PCI-MPG24

4-CH MPEG4 Hardware

Video Compression Card

Programming Guide

Manual Rev. 2.01

Revision Date: January 28, 2008

Part No: 50-15035-1040

Recycled Paper

Advance Technologies; Automate the World.

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

Trademarks

Microsoft®, Windows NT®, Windows 98®, Windows 2000®, and Windows XP® are registered trademarks of Microsoft Corporation. Borland C++ Builder® is a registered trademark of Borland International, Inc.

Other product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Getting Service from ADLINK

Customer Satisfaction is top priority for ADLINK Technology Inc. Please contact us should you require any service or assistance.

ADLINK TECHNOLOGY INC.

Web Site:	http://www.adlinktech.com
Sales & Service:	Service@adlinktech.com
TEL:	+886-2-82265877
FAX:	+886-2-82265717
Address:	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan

Please email or FAX this completed service form for prompt and satisfactory service.

Company Information	
Company/Organization	
Contact Person	
E-mail Address	
Address	
Country	
TEL	FAX:
Web Site	
Product Information	
Product Model	
Environment	OS: M/B: CPU: Chipset: BIOS:

Please give a detailed description of the problem(s):

Table of Contents

List of Tables

List of Figures

1 DirectX Programming Guide

1.1 DirectShow Application Programming Introduction

A complete documentation on DirectShow application programming can be found at **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directX/htm/introductiontodirectshow.asp**. If a DirectX 9.0 is installed, this documentation is also available from DirectX SDK Help.

The purpose of writing a DirectShow Application is to build a filter graph by connecting several filters together to perform a given task such as previewing video/audio, capturing video/audio and multiplexing them to write into a file. Each filter performs a single operation and pass data from its output pin to the input pin of the next filter in the graph.

To build a capture graph using a program, first obtain the interface pointer of the capture filter. The ADLink Bt878 Video Capture filter and the ADLINK Hardware MPEG4 Device filter can be obtained through the **system device enumerator**. After holding an interface pointer to the capture filter object, use method **IGraphBuilder::AddSourceFilter** to add the source filter object to the filter graph. Use **IFilterGraph::AddFilter** to add other downstream filters to the filter graph. After filters are added, call **IFilterGraph::ConnectDirect** or **IGraphBuilder::Connect** methods to connect output pins from upstream filters to the input pins of the downstream filters. Calling methods such as **IMediaControl::Run**, **IMediaControl::Pause**, or **IMediaControl::Stop** will change filter state to running, paused or stopped.

The filters needed for capturing MPEG4 streams are listed in section 1.2, along with a detailed description for each filter and its pins. Example filter graphs for previewing/capturing MPEG4 streams are illustrated in section 1.3. Section 1.4 provides examples of two ways of controlling the device driver.

1.2 Descriptions of Filters

This chapter lists filters needed to build a filter graph for capturing MPEG4 video stream and previewing video stream.

Source Filter

ADLink Bt878 Video Capture

ADLink Bt878 Video Capture Filter belongs to the family of WDM Streaming Capture Devices. It is a kernel-mode KsProxy plug-in, where an application can treat it simply as a filter. Use the System Device Enumerator to add this filter to a filter graph.

Filter Name	ADLink Bt878 Video Capture
Filter CLSID	Not applicable
Filter Category Name	WDM Streaming Capture Devices
Filter Category	AM_KSCATEGORY_CAPTURE
Video Capture Pin Supported Media Types	MEDIATYPE_Video Subtypes: ▶ MEDIASUBTYPE_YUY2 ▶ MEDIASUBTYPE_YVU9 ▶ MEDIASUBTYPE_UYVY ▶ MEDIASUBTYPE_YV12 ▶ MEDIASUBTYPE_I420 ▶ MEDIASUBTYPE_Y41P ▶ MEDIASUBTYPE_RGB24 ▶ MEDIASUBTYPE_RBG32 ▶ MEDIASUBTYPE_RBG565 ▶ MEDIASUBTYPE_RBG555

Video Preview Pin Supported Media Types	MEDIATYPE_Video Subtypes: ▶ MEDIASUBTYPE_YUY2 ▶ MEDIASUBTYPE_YVU9 ▶ MEDIASUBTYPE_UYVY ▶ MEDIASUBTYPE_YV12 ▶ MEDIASUBTYPE_I420 ▶ MEDIASUBTYPE_Y41P ▶ MEDIASUBTYPE_RGB24 ▶ MEDIASUBTYPE_RBG32 ▶ MEDIASUBTYPE_RBG565 ▶ MEDIASUBTYPE_RBG555
Merit	MERIT_DO_NOT_USE

Table 1-1: ADLink Bt878 Video Capture

ADLINK Hardware MPEG4 Device

ADLINK Hardware MPEG4 Device belongs to the family of WDM Streaming Capture Devices. It is a kernel-mode KsProxy plug-in where an application can treat it simply as a filter. Use the System Device Enumerator to add this filter to a filter graph.

Filter Name	ADLINK Hardware MPEG4 Device
Filter CLSID	Not applicable
Filter Category Name	WDM Streaming Capture Devices
Filter Category	AM_KSCATEGORY_CAPTURE
Video Capture Pin Supported Media Types	DIVX_MPEG4, MICROSOFT_MPEG4, MPEG2, MPEG1, H.263, MJPG (For detailed definition of each media type, please refer to DirectX Reference Manual chapter 2.2: Media Types)

Video Preview Pin Supported Media Types	DIVX_MPEG4, MICROSOFT_MPEG4, MPEG2, MPEG1, H.263, MJPG (For detailed definition of each media type, please refer to DirectX Reference Manual chapter 2.2: Media Types)
Merit	MERIT_DO_NOT_USE

Table 1-2: ADLINK Hardware MPEG4 Device

Video Renderer Filter

Filter Name	Video Renderer
Filter CLSID	CLSID_VideoRenderer
Filter Category Name	DirectShow Filters
Filter Category CLSID	CLSID_LegacyAmFilterCategory
Input Pin Media Types	MEDIATYPE_Video
Output Pin Media Types	Not Applicable
Merit	MERIT_DO_NOT_USE

Table 1-3: Video Renderer Filter

MPEG4 AVI Mux Filter

Filter Name	AVI Mux
Filter CLSID	CLSID_AviDest
Filter Category Name	DirectShow Filters
Filter Category CLSID	CLSID_LegacyAmFilterCategory
Input Pin Media Types	Any major type that corresponds to an old-style FOURCC, or MEDIATYPE_AUXLine21Data. ▶ If the major type is MEDIATYPE_Audio, the format must be FORMAT_WaveFormatEx. ▶ If the major type is MEDIATYPE_Video, the format must be FORMAT_VideoInfo or FORMAT_DvInfo. ▶ If the major type is MEDIATYPE_Interleaved, the format must be FORMAT_DvInfo.
Output Pin Media Types	MEDIATYPE_Stream, MEDIASUBTYPE_Avi
Merit	MERIT_DO_NOT_USE

Table 1-4: MPEG4 AVI Mux Filter

MPEG4 File Writer

Filter Name	File writer
Filter CLSID	CLSID_FileWriter
Filter Category Name	DirectShow Filters
Filter Category CLSID	CLSID_LegacyAmFilterCategory
Input Pin Media Types	MEDIATYPE_Stream, MEDIASUBTYPE_NULL
Output Pin Media Types	Not Applicable
Merit	MERIT_DO_NOT_USE

Table 1-5: MPEG4 File Writer

CrossBar Filter

If the device is a capture board, a CrossBar filter is needed for switching video source.

Filter Name	ADLink Bt878 CrossBar
Filter Category Name	WDM_Streaming Crossbar Devices

Table 1-6: CrossBar Filter

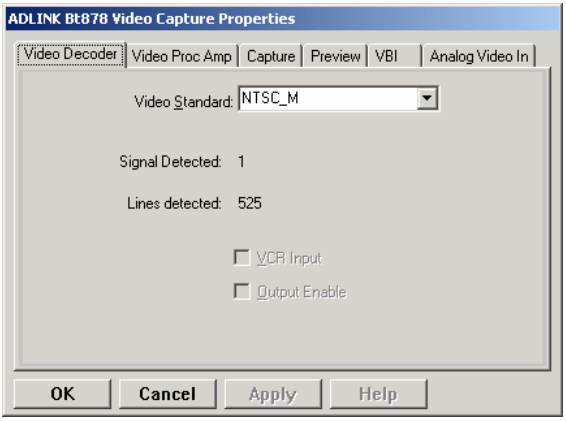
1.3 Example Graphs

Microsoft DirectX SDK provides a very useful debugging utility - GraphEdit, which can be used to simulate graph building. From the **Graph** menu of the GraphEdit application, click **Insert Filters…** and choose the filters required. Filters are organized by categories. Click **Insert Filter** to add filters to a graph. Connect the two filters' pins by dragging the mouse from one filter's output pin to another filter's input pin. An arrow will be drawn if these two pins agree on the connection.

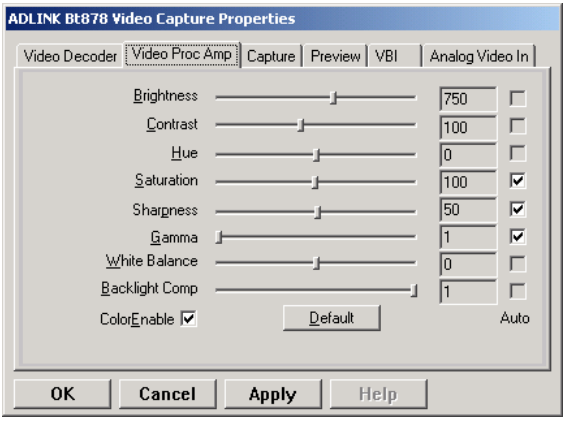
After inserting the ADLINK Bt878 Video Capture filter, the ADLink Bt878 Crossbar filter, and/or ADLINK Hardware MPEG4 Device filter, right-click on the rectangle and click **Filter Properties….** The filter properties dialogue will appear. Use the property pages to set video settings before connecting video pins to other filters. The property pages are shown below:

ADLink Bt878 Video Capture filter

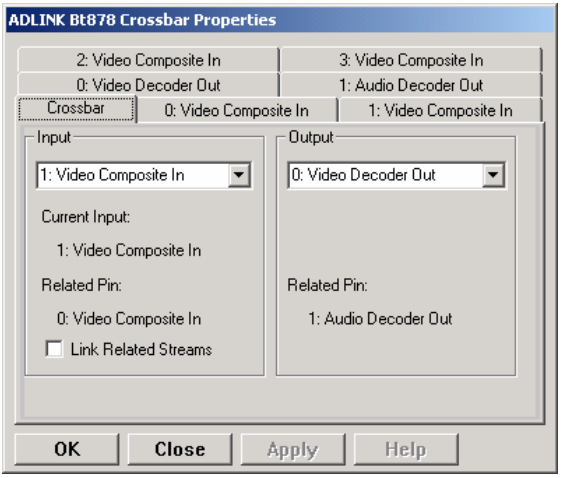
Video Decoder:



Video Proc Amp:



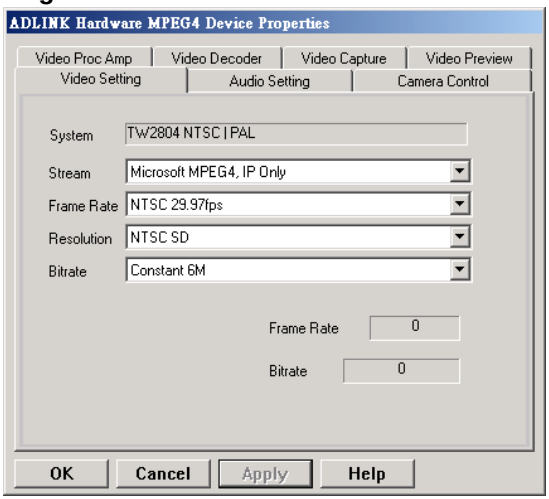
ADLink Bt878 Crossbar filter



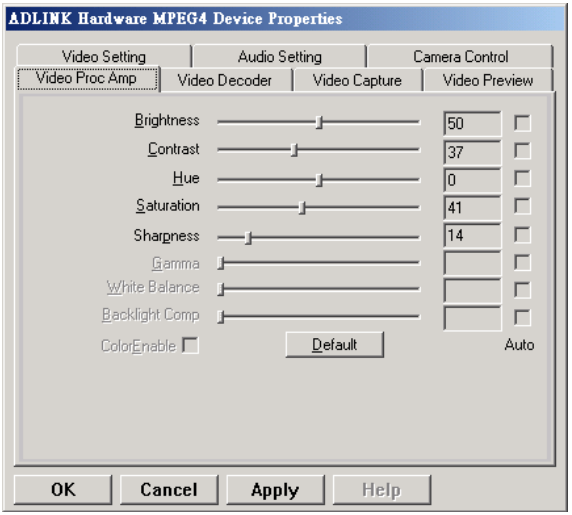
Select video input before or during video previewing.

ADLINK Hardware MPEG4 Device

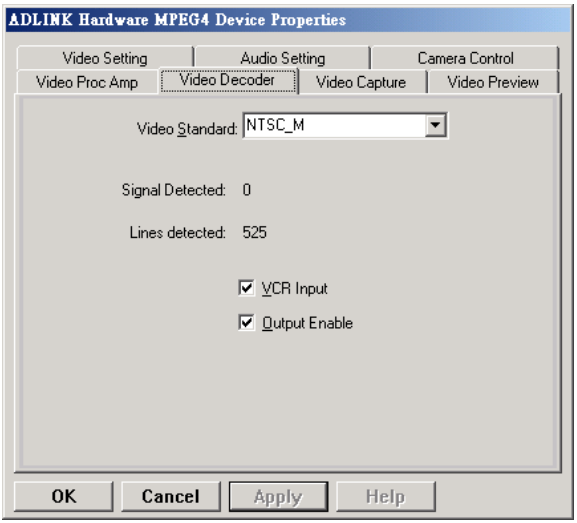
Video Setting:



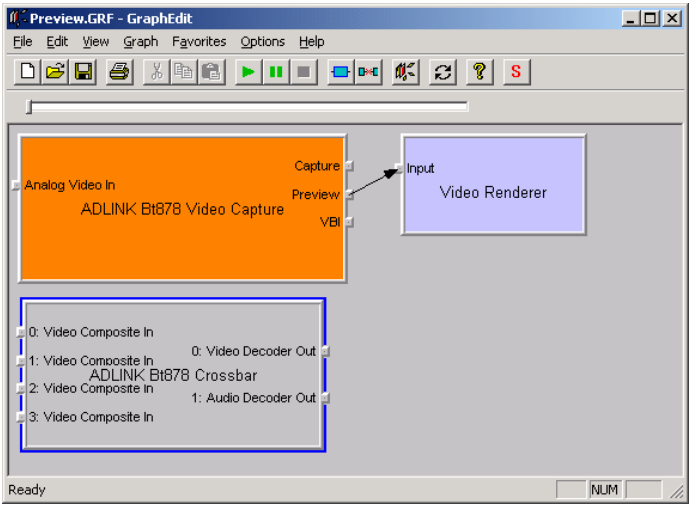
Video Proc Amp:



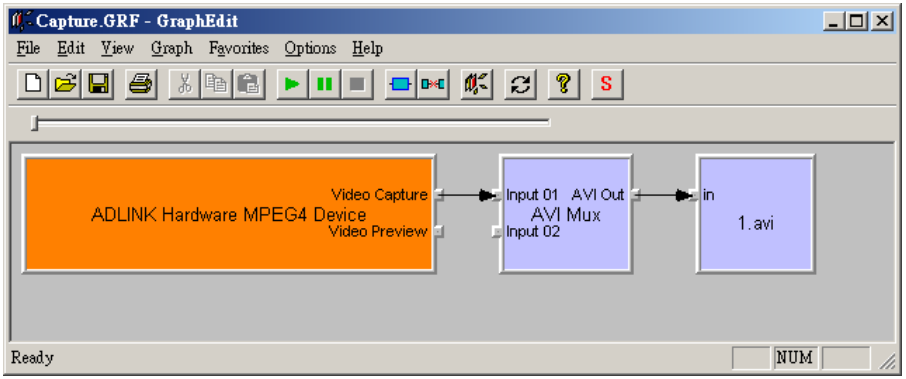
Video Decoder



Preview



Capture



1.4 Controlling Driver

Preview

ADLINK Bt878 Video Capture

The ADLINK Bt878 Video Capture Filter provides property pages and exposes COM interfaces to control video. Hence, an application has two ways to control video configurations: via property pages or via the COM interfaces.

Use Property Pages

There are two embedded property pages in the driver. To show these property pages, use Windows API: OleCreateProperty-Frame.

Documentation on Displaying a Filter's Property Page can be found on Microsoft MSDN homepage.

Below is an example code for adding property pages:

```
// pFilter points to the capture filter

ISpecifyPropertyPages *pSpecify;
HRESULT hr;
hr = pFilter-
>QueryInterface(IID_ISpecifyPropertyPages, (void
**)&pSpecify);
if (SUCCEEDED(hr))
{
FILTER_INFO FilterInfo;
pFilter->QueryFilterInfo(&FilterInfo);
FilterInfo.pGraph->Release();

CAUUID caGUID;
pSpecify->GetPages(&caGUID);
pSpecify->Release();
```

```
OleCreatePropertyFrame(
NULL,                    // Parent window
0,                       // x (Reserved)
0,                       // y (Reserved)
FilterInfo.achName,      // Caption for the
dialog box
1,                       // Number of filters
(IUnknown **)&m_pFilter, // Pointer to the filter
caGUID.cElems,           // Number of property
pages
caGUID.pElems,           // Pointer to property
page CLSIDs
0,                       // Locale identifier
0,                       // Reserved
NULL                     // Reserved
);
CoTaskMemFree(caGUID.pElems);
}
```

Use COM interfaces

Use the methods of the **IAMVideoProvAmp** interface of standard DirectShow Interface to retrieve or set the qualities of an incoming video signal.

ADLINK Bt878 Crossbar

The ADLink Bt878 Crossbar filter implements an IAMCrossbar interface. It routes signals from an analog or digital source to a video capture filter.

The routing definition of PCI-MPG24 card is shown in the following figure:

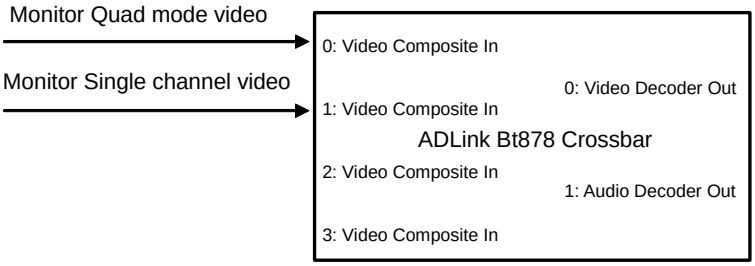


Figure 1-1: ADLink Bt878 Crossbar

For single video port selection please refer to the Bt878 GPIO pin definition in chapter 2.1.

Capture

ADLINK Hardware MPEG4 Device

The ADLINK Hardware MPEG4 Device Filter provides property pages and exposes COM interfaces to control video. Hence, an application can have two ways to control video configurations: via the property pages or via the COM interfaces.

Use Property Pages

There are three embedded property pages in the driver. To show these property pages, use Windows API: **OleCreatePropertyFrame**.

Documentation on Displaying a Filter's Property Page can be found on Microsoft MSDN homepage.

The example code for adding property pages is the same as that of the ADLINK Bt878 Video Capture.

Use COM interfaces

It is standard practice to use the standard DirectShow interfaces defined for A/V capture filter and output pin to retrieve and set video configurations. However, due to a known issue in the ADLINK Hardware MPEG4 device driver, the programmer has to use a proprietary interface, IGOChip in addition to the standard interfaces. Details about the IGOChip interface and a sample code are provided in the DirectX Reference Manual.

2 DirectX Reference Manual

2.1 Preview

GPIO Access

The GPIO provides a method to read board information, select input channel, and control digital inputs/digital outputs.

Sample:

```
#define INSTANCE_DATA_OF_PROPERTY_PTR(x) (
  (PKSPROPERTY ((x)) ) + 1 )
#define INSTANCE_DATA_OF_PROPERTY_SIZE(x) (
  sizeof((x)) – sizeof (KSPROPERTY) )


void GPIOWrite(IBaseFilter* pFilter,DWORD
  bit,DWORD value)
{
IKsPropertySet *pKs = NULL;
DWORD TypeSupport = 0;
KSPROPERTY_CUSTOMBT848_GPIO_S rc;
HRESULT hr;
ULONG ret=0;


if (pFilter->QueryInterface(IID_IKsPropertySet,
  (void **)&pKs) == S_OK)
{
hr = pKs-
  >QuerySupported(PROPSETID_CUSTOMBT848,KSPROPERTY
  _CUSTOMBT848_GPIO,&TypeSupport);
if(TypeSupport & KSPROPERTY_SUPPORT_GET)
{
ZeroMemory(&rc,sizeof(rc));
rc.dwOperation=BT848_CUSTPROP_GPIO_SETGPDATABITS;
```

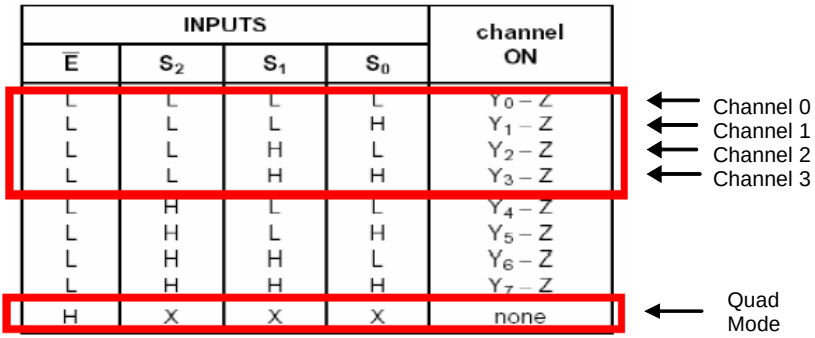
```
rc.dwFromBit = bit;
rc.dwToBit = bit;
rc.dwValue = value;
rc.dwOffset =0;
hr = pKs->Get(
PROPSETID_CUSTOMBT848,//idetificador del driver
KSPROPERTY_CUSTOMBT848_GPIO,
INSTANCE_DATA_OF_PROPERTY_PTR(&rc),
INSTANCE_DATA_OF_PROPERTY_SIZE(rc),
&rc,// variable a rellenar con los datos
sizeof(rc),
&ret);
}
pKs->Release();
}
}

DWORD GPIORead(IBaseFilter* pFilter,DWORD bit)
{
IKsPropertySet *pKs = NULL;
DWORD TypeSupport = 0;
KSPROPERTY_CUSTOMBT848_GPIO_S rc;
HRESULT hr;
ULONG ret=0;
DWORD ReturnValue=0;

if (pFilter->QueryInterface(IID_IKsPropertySet,
  (void **)&pKs) == S_OK)
{
hr = pKs-
  >QuerySupported(PROPSETID_CUSTOMBT848,KSPROPERTY
  _CUSTOMBT848_GPIO,&TypeSupport);
if(TypeSupport & KSPROPERTY_SUPPORT_GET)
```

```
  {
  ZeroMemory(&rc,sizeof(rc));
  rc.dwOperation =
    BT848_CUSTPROP_GPIO_GETGPDATABITS;
  rc.dwFromBit = bit;
  rc.dwToBit = bit;
  rc.dwOffset =0;
  hr = pKs->Get(
  PROPSETID_CUSTOMBT848,//idetificador del driver
  KSPROPERTY_CUSTOMBT848_GPIO,
  INSTANCE_DATA_OF_PROPERTY_PTR(&rc),
  INSTANCE_DATA_OF_PROPERTY_SIZE(rc),
  &rc,// variable a rellenar con los datos
  sizeof(rc),
  &ret);
  ReturnValue = rc.dwValue;
  }
  pKs->Release();
  }
  return ReturnValue;
}
```

Bt878 GPIO PIN Definition

Pin	Type	Function
GPIO0	Output	--
GPIO1	Output	Set watchdog timer enable / disable Set “1” => disable (default), set ”0” => enable
GPIO2	Output	Control the watch dog timer count down time
GPIO3	Output	
GPIO4	Input	Card ID bit 0 (setting by dip switch)
GPIO5	Input	Card ID bit 1 (setting by dip switch)
GPIO6	Input	Card ID bit 2 (setting by dip switch)
GPIO7	Input	--
GPIO8	Input	Port 1 DI
GPIO9	Input	Port 2 DI
GPIO10	Input	Port 3 DI
GPIO11	Input	Port 4 DI
GPIO12	Output	Port 1 DO
GPIO13	Output	Port 2 DO
GPIO14	Output	Port 3 DO
GPIO15	Output	Port 4 DO
GPIO16	Output	Monitor single channel , data enable (low active) **(E#)**
GPIO17	Output	Monitor single channel , data 0 **(S0)**
GPIO18	Output	Monitor single channel , data 1 **(S1)**
GPIO19	Output	Monitor single channel , data 2 **(S2)**

Table 2-1: Bt878 GPIO PIN Definition



INPUTS				channel ON
$\overline{E}$	S_2	S_1	S_0	
L	L	L	L	$Y_0 = Z$
L	L	L	H	$Y_1 = Z$
L	L	H	L	$Y_2 = Z$
L	L	H	H	$Y_3 = Z$
L	H	L	L	$Y_4 = Z$
L	H	L	H	$Y_5 = Z$
L	H	H	L	$Y_6 = Z$
L	H	H	H	$Y_7 = Z$
H	X	X	X	none

Table 2-2: Function Table

EEPROM Access

ADLink Bt878 Video Capture provides a method for accessing the I2C register. The interface can store a few data, for example, board identification.

Sample

```
#define INSTANCE_DATA_OF_PROPERTY_PTR(x) (
  (PKSPROPERTY ((x))) + 1 )


#define INSTANCE_DATA_OF_PROPERTY_SIZE(x) (
  sizeof((x)) - sizeof(KSPROPERTY) )
BYTE EEPROMRead(IBaseFilter *pFilter, BYTE offset)
{
IKsPropertySet *pKs = NULL;
DWORD TypeSupport = 0;
KSPROPERTY_CUSTOMBT848_I2C_S I2C;
BYTE uAddress;
HRESULT hr;
ULONG ret=0;
```

```
if((hr=pFilter->QueryInterface(IID_IKsPropertySet,
  (void **)&pKs)) == S_OK)
{
hr = pKs-
  >QuerySupported(PROPSETID_CUSTOMBT848,KSPROPERTY
  _CUSTOMBT848_I2C,&TypeSupport);
if(TypeSupport & KSPROPERTY_SUPPORT_GET)
{
uAddress = 0xa0;
ZeroMemory(&I2C,sizeof(I2C));
I2C.bDontWaitACK = true;
I2C.dwOperation = BT848_CUSTPROP_I2C_SETFREQ;
I2C.dwFreq = 100000;
hr = pKs->Get(
PROPSETID_CUSTOMBT848,
KSPROPERTY_CUSTOMBT848_I2C,
INSTANCE_DATA_OF_PROPERTY_PTR(&I2C),
INSTANCE_DATA_OF_PROPERTY_SIZE(I2C),
&I2C,
sizeof(I2C),
&ret);
I2C.dwOperation=BT848_CUSTPROP_I2C_R3;
I2C.ucAddress= uAddress;
I2C.ucInBuf[0] = offset;
I2C.dwOutLen = 0;
I2C.dwInLen  =  1;
I2C.bDontWaitACK = TRUE;
hr = pKs->Get(
PROPSETID_CUSTOMBT848,
KSPROPERTY_CUSTOMBT848_I2C,
  INSTANCE_DATA_OF_PROPERTY_PTR(&I2C),
INSTANCE_DATA_OF_PROPERTY_SIZE(I2C),
&I2C,
```

```
sizeof(I2C),
&ret);
}
pKs->Release();
}
return I2C.ucInBuf[1];
}
```

2.2 Capture

WDM Streaming Capture Filter

Filter Interfaces	**Microsoft DirectShow Interfaces:** IAMAnalogVideoDecoder, IAMCameraControl, IAMDroppedFrames, IAMVideoProcAmp, IBaseFilter, IKsPropertySet, ISpecifyPropertyPages **Capture Interfaces:** IGOChip, IGOChipConfig, IGOInfo, IAccessFunc, IAdvanced
Video Capture Pin Supported Media Types	DIVX_MPEG4, MICROSOFT_MPEG4, MPEG2, MPEG1, H263, MJPG
Video Capture Pin Interfaces	Microsoft DirectShow Interfaces: IAMStreamConfig, IKsPin, IKsPropertySet, IPin
Video Preview Pin Supported Media Types	DIVX_MPEG4, MICROSOFT_MPEG4, MPEG2, MPEG1, H263, MJPG
Video Preview Pin Interfaces	Microsoft DirectShow Interfaces: IAMStreamConfig, IKsPin, IKsPropertySet, IPin
Audio Capture Pin Supported Media Types	MEDIATYPE_Audio, MEDIASUBTYPE_PCM, MEDIASUBTYPE_ADPCM, MEDIASUBTYPE_IMA_ADPCM
Audio Capture Pin Interfaces	Microsoft DirectShow Interfaces: IAMBufferNegotiation, IAMStreamConfig, IAMStreamControl, IKsPin, IKsPropertySet, IStreamBuilder, IMediaSeeking, IPin, IQualityControl
Filter CLSID	Not applicable.
Property Page CLSID	**Video Control Property Page:** 35D3656A-6C20-46B0-B44A-DC8E861F1205 **Audio Control Property Page:** 8ED37ED7-477B-4764-B72E-DFC2D1899C6C
Merit	MERIT_DO_NOT_USE
Filter Category	AM_KSCATEGORY_CAPTURE CLSID_VideoInputDeviceCategory CLSID_AudioInputDeviceCategory

Table 2-3: WDM Streaming Capture Filter

For Microsoft DirectShow interfaces, follow the links for online reference. Alternatively, please visit http://msdn.microsoft.com/library/ and from the left panel navigation, select Graphics and Multimedia -> DirectX -> SDK Documentation -> DirectX 9.0 (C++) -> DirectShow -> DirectShow Reference -> Interfaces for a complete list of Microsoft DirectShow filter interfaces references.

Media Types

DIVX_MPEG4

Major type	MEDIATYPE_Video
Subtype	‘D’, ‘X’, ‘5’, ‘0’, 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71
Format Type	FORMAT_Videoinfo

Table 2-4: DIVX_MPEG4

MICROSOFT_MPEG4

Major type	MEDIATYPE_Video
Subtype	‘M’, ‘P’, ‘4’, ‘S’, 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71
Format Type	FORMAT_Videoinfo

Table 2-5: MICROSOFT_MPEG4

MPEG2

Major type	MEDIATYPE_Video
Subtype	MEDIASUBTYPE_MPEG2_VIDEO
Format Type	FORMAT_MPEG2Video

Table 2-6: MPEG2

MPEG1

Major type	MEDIATYPE_Video
Subtype	MEDIASUBTYPE_MPEG1Payload
Format Type	FORMAT_MPEGVideo

Table 2-7: MPEG1

H263

Major type	MEDIATYPE_Video
Subtype	'W', 'M', 'P', 0x3, 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71
Format Type	FORMAT_Videoinfo

Table 2-8: H263

MJPG

Major type	MEDIATYPE_Video
Subtype	'M', 'J', 'P', 'G', 0x0000, 0x0010, 0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71
Format Type	FORMAT_Videoinfo

Table 2-9: MJPG

Data Structures

TCFG_HEADER Structure

The TCFGHEADER structure will be used in the following structures: TCFGSYSTEM, TCFGSTREAM, TCFGFRAMERATE, TCFGRESOLUTION, TCFGBRCTRL, and TCFGMISC. It provides general information about the structure it resides in.

Syntax

```
typedef struct
{
char  name[MAX_NAME];
char  desc[MAX_DESC];
```

```
unsigned long flags;
unsigned long size;
} TCFG_HEADER;
```

Members

name

The name of the configuration. It uses a string less than MAX_NAME (64) characters included in quotes. For example: “MPEG2, IPB” for a stream setting and “4M” for a bitrate setting.

desc

The description of the configuration. Use a string of less than MAX_DESC (256) characters included in quotes.

flags

The flags member provides information on what fields are provided in the structure where the TCFG_HEADER is located. Each bit in flags corresponds to one field. Each TCFGxxxx structure (except for TCFGSYSTEM) has a corresponding FLAGS_xxxx enumeration that shows the relationship between field and bit position. The enumeration also has a FLAGS_xxxx_MANDETORY field indicating which fields have to be provided. Set a bit as 1 if the corresponding field value is provided. Take the TCFGFRAMERATE structure as an example, if only the mandatory fields are provided, which are frame rate and tv_standard, then the flags should be 0x9 (1001).

size

The size of the structure is where the TCFGHEADER is located. For example, in a TCFGSTREAM structure, the size in TCFG_HEADER is the size of TCFGSTREAM.

TCFGSYSTEM Structure

The TCFGSYSTEM structure describes settings of the sensor (image sensor or video decoder) being used to capture video and some general settings of the MPEG4 chip.

Syntax

```
typedef struct
{
TCFG_HEADER  header;

TV_STANDARDtv_standard;
long  framerate;
long  sensor_h;
long  sensor_v;
char  format;
char  pformat;

char  sensor_656_mode;
char  valid_enable;
char  valid_polar;
char  href_polar;
char  vref_polar;
char  field_id_polar;
char  sensor_bit_width;
char  hv_resync_enable;

long  reserved;
} TCFGSYSTEM;
```

Members

header

Header information about the structure.

tv_standard

tv_standard can only be set as TVStandard_NTSC_Mask or TVStandard_PAL_Mask.

framerate

The real frame rate will be the value of frame rate divided by 1001. For example: if frame rate = 30000, then the real frame rate = 30000 / 1001 = 29.97.

sensor_h

The horizontal resolution of the sensor source input, in pixels.

sensor_v

The vertical resolution of the sensor source input, in pixels.

sensor_h and sensor_v constitute the source video size.

The common source video sizes are:

- 320 * 240 (QVGA)
- 352 * 288 (CIF)
- 640 * 480 (VGA)
- 720 * 480 (Full D1, NTSC)
- 720 * 576 (Full D1, PAL)

Format

Sensor pixel format:

- 0: YUV progressive
- 1: YUV interlace
- 2: RGB Bayer

pformat

Sensor pixel format. Valid only if format is RGB Bayer.

- 2: RGB-Bayer in GB-format
- 3: RGB-Bayer in GR-format
- 4: RGB-Bayer in BG-format
- 5: RGB-Bayer in RG-format

sensor_656_mode

Specifies if the sensor input is in 656 mode.

- 0: The sensor is not in 656 mode
- 1: The sensor is in 656 mode

valid_enable

If valid_enable is set, then input image data is valid only if valid signal is active.

- 0: Disable valid signal.
- 1: Enable valid signal.

valid_polar

Specifies the polarity of the valid signal provided by the sensor or emulator.

- 0: Vvalid signal polarity will be active-high
- 1: Valid signal polarity will be active-low

href_polar

Specifies the polarity of the horizontal reference signal provided by the sensor or emulator. Only valid in non-656 mode.

- 0: H reference polarity will be active-high
- 1: H reference polarity will be active-low

vref_polar

Specifies the polarity of the vertical reference signal provided by the sensor or emulator. Only valid in non-656 mode.

- 0: V reference polarity will be active-high
- 1: V reference polarity will be active-low

field_id_polar

Specifies the polarity of the field ID.

- 0: Top field ID = 0; Bottom field ID = 1
- 1: Top field ID = 1; Bottom field ID = 0

sensor_bit_width

The bit width of the image data provided by sensor or emulator.

- 0: Data is 8-bit wide
- 1: Data is 10-bit wide

hv_resync_enable

By enabling this option, the video compression process pauses when the input video signal is temporarily out-of-sync. It will resume and re-synchronize to the video input signal when regular synchronization signals are recovered. Video signals from some image sensors have constantly changing horizontal and vertical periods. This option needs to be disabled when working with such sensors.

reserved

Reserved for future use.

TCFGSTREAM Structure

The TCFGSTREAM structure describes settings of the output video stream of the MPEG4 chip.

Syntax

```
typedef struct
{
TCFG_HEADER  header;

EVideoFormat  compress_mode;
ESequenceMode  sequence;

unsigned char  gop_mode;
unsigned char  gop_size;
unsigned char  mpeg4_mode;
unsigned char  DVD_compliant;
unsigned char  deinterlace_mode;

unsigned char  search_range;
unsigned char  gop_head_enable;
unsigned char  seq_head_enable;
unsigned char  aspect_ratio;
long  reserved;
} TCFGSTREAM;
```

Members

header

Header information about the structure.

compress_mode

The stream's compression mode. Refer to Eumeration: EVideoFormat for details.

- MPEG1: 0x00
- MPEG2: 0x01
- H263: 0x03
- MPEG4: 0x04
- MOTIONJPEG: 0x08

sequence

The sequence mode of the encoding stream. There are three types of frames used in a stream sequence: Intra frames (I), Predictive frames (P), and Bi-directional frames (B). This member indicates the types of frames being used in a stream sequence. Refer to Enumeration: ESequenceMode for details.

- IONLY: Only I-frames in the stream sequence (1)
- IPONLY: Only I and P frames in the stream sequence (2)
- IPB: All types of frames (I, P and B) in the stream sequence (3)

gop_mode

The GOP (Group Of Picture) mode of the encoding stream sequence. Encoding DivX format MPEG4 stream requires closed GOP mode.

- GOP_MODE_OPEN: 0
- GOP_MODE_CLOSE: 1

gop_size

The Group of Pictures (GOP) size. This value is the key frame interval. Note that in the IPB sequence mode, the value must be a multiple of 3. If the stream is preferred to be DVD compliant, the GOP size must be less than or equal to 18.

mpeg4_mode

MPEG4 stream mode. Valid only when compress_mode is four. Refer to Enumeration: MPEG4_MODE for details.

- WIS_MPEG4: 0
- DIVX_MPEG4: 1
- MICROSOFT_MPEG4: 2

DVD_compliant

Specifies if the stream is to be DVD compliant. Valid only when the compression mode is MPEG2.

- 0: Disable DVD_compliant
- 1: Enable DVD_compliant

deinterlace_mode

- 0: Use one field only
- 1: Use MPEG4 deinterlace algorithm
- 2: Interlace coding is used; no de-interlacing performed

search_range

The searching range for motion vectors. Typical values are 16, 32, 64, or 128. Microsoft format MPEG4 stream requires that the search range be 64. H.263 format stream requires the search range to be 32.

gop_head_enable

Only encoding the Microsoft format MPEG4 stream requires disabling the GOP head. All other format streams requires enabling the GOP head.

- 0: Disable GOP head
- 1: Enable GOP head

seq_head_enable

Only encoding Microsoft format MPEG4 stream requires disabling the sequence head. All other format streams require enabling sequence head.

- 0: Disable sequence head
- 1: Enable sequence head

aspect_ratio

The ratio between the width and the height of the picture. This information is included in the sequence header.

- 1: 1:1
- 2: 4:3
- 3: 16:9

reserved

Reserved for future use.

TCFGFRAMERATE Structure

The TCFGFRAMERATE structure describes frame rate related settings of the output video stream of MPEG4 chip.

Syntax

```
typedef struct
{
TCFG_HEADER  header;

TV_STANDARD  tv_standard;
unsigned  longframe_rate;
unsigned  long   drop_frame;
unsigned  charivtc_enable;
long  reserved;
```

```
} TCFGFRAMERATE;
```

Members

header

Header information about the structure

tv_standard

tv_standard can only be set as TVStandard_NTSC_Mask or TVStandard_PAL_Mask.

frame_rate

Output frame rate

drop_frame

- 0: Keep original frame rate. No frames dropped
- 1: Keep 1/2 original frame rate
- 2: Keep 1/3 original frame rate
- n: Keep 1/(n+1) original frame rate

ivtc_enable

IVTC (InVerse TeleCine) is a process where video editing tools reverse the Telecine process. Basically IVTC brings back movie's original frame rate from NTSC's 29.97fps to 24fps.

- 0: Disable IVTC
- 1: Enable IVTC

reserved

Reserved for future use

TCFGRESOLUTION Structure

The TCFGRESOLUTION structure describes the resolution of an encoded stream.

Syntax

```
typedef struct
{
TCFG_HEADER  header;
TV_STANDARD  tv_standard;

unsigned long width;
unsigned long height;

unsigned char h_sub_window;
unsigned char v_sub_window;
unsigned long h_sub_offset;
unsigned long v_sub_offset;

unsigned char h_scale_enb;
unsigned char v_scale_enb;
unsigned char sub_sample;

unsigned long max_bitrate;
unsigned long min_bitrate;

long reserved;
} TCFGRESOLUTION;
```

Members

header

Header information about the structure.

tv_standard

tv_standard can only be set as TVStandard_NTSC_Mask or TVStandard_PAL_Mask.

width

The desired output stream resolution: horizontal size.

height

The desired output stream resolution: vertical size.

h_sub_window

Specify if performing sub-window (cropping) in the horizontal direction.

- 0: Disable sub-window
- 1: Enable sub-window

v_sub_window

Specify if performing sub-window (cropping) in the vertical direction.

- 0: Disable sub-window
- 1: Enable sub-window

h_sub_offset

If h_sub_window is performed, this parameter specifies a relative offset between the leftmost pixel of the output stream and the leftmost pixel of the source stream, in pixels.

v_sub_offset

If v_sub_window is performed, this parameter specifies a relative offset between the topmost pixel of the output stream and the topmost pixel of the source stream, in pixels.

h_scale_enb

Specify if it will perform ½ scaling in the horizontal direction.

- 0: Disable scaling
- 1: Enable scaling

v_scale_enb

Specify if it will perform ½ scaling in the vertical direction.

- 0: Disable scaling
- 1: Enable scaling

sub_sample

Specify if it is performing sub_sampling. Sub-sampling will perform ½ scaling down to the stream in both horizontal and vertical directions.

- 0: Disable sub_sampling
- 1: Enable sub_sampling

Note:	Sub-sampling, sub-windowing, and scaling are three different methods provided by the chip to reduce the source stream size, and performed in the sequence above. If sub-sampling and sub-windowing are both enabled, the sub-offset for sub-window will be relative to the leftmost pixel of the stream AFTER sub-sampling is performed.

max_bitrate

The maximum bit rate allowed for this resolution.

min_bitrate

The minimum bit rate allowed for this resolution.

reserved

Reserved for future use.

TCFGBRCTRL Structure

The TCFGBRCTRL structure describes the bit-rate control setting of encoded stream.

Syntax

```
typedef struct
{
TCFG_HEADER  header;

unsigned long target_bitrate;
unsigned long peak_bitrate;
unsigned long vbv_buffer;
unsigned char converge_speed;
unsigned char lambda;

unsigned long Q;
unsigned char IQ;
unsigned char PQ;
unsigned char BQ;

long reserved;
} TCFGBRCTRL;
```

Members

header

Header information about the structure.

target_bitrate

The desired average target bit rate of the encoded stream, in bits per second (bps).

- 0: If Q>0, apply the variable bitrate control (VBR) using the value of Q. If Q=0, no bitrate control algorithm is applied. Bitrate will be determined by values of IQ, PQ, and BQ provided by the user.
- >0: Apply constant bitrate control (CBR) using the value of target_bitrate.

peak_bitrate

The highest bit rate allowed in the encoded stream, in bits per second (bps). This parameter is only valid when applying constant bitrate control (both Q and target_bitrate are greater than 0).

vbv_buffer

Specifies VBV buffer size. Video Buffering Verifier (VBV) is a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

converge_speed

Specifies the converging speed of bit rate control process. Its value range is [0, 100]. The larger the value, the faster the converging speed.

lambda

The factor determining stream quality. Its value range is [0, 100]. The larger the value, the smoother the stream however, the quality of each frame decreases. This is inversely true for smaller values. However, due to frame drops, the entire video stream would appear “jumpy”.

Q

Initial quantizer. This value is divided by four.

- 0: If target_bitrate>0, apply constant bitrate control. The initial quantizer value is calculated. If target_bitrate=0, no bitrate control algorithm is applied. Use IQ, PQ, and BQ values provided by the user to determine the bitrate value.
- >0: If target_bitrate is set to 0, apply VBR (variable bitrate) using the value of Q. If target_bitrate is greater than 0, apply CBR (constant bitrate) using the value of target_bitrate.

IQ

The fixed quantized scale for I-frames during the entire encoding session. This member is valid only when Q and target_bitrate are both set to '0'.

PQ

The fixed quantized scale for P-frames during the entire encoding session. This member is valid only when Q and target_bitrate are both set to '0'.

BQ

The fixed quantized scale for B-frames during the entire encoding session. This member is valid only when Q and target_bitrate are both set to '0'.

reserved

Reserved for future use.

TCFGMISC Structure

The TCFGSTREAM structure describes miscellaneous settings of the output video stream of MPEG4 chip.

Syntax

```
typedef struct
{
TCFG_HEADER  header;

unsigned char   av_sync_enable;
unsigned char   iip_enable;
unsigned char   vbi_enable;
unsigned char   four_channel_enable;

FilterMode   h_filter_mode;
FilterMode   v_filter_mode;
char   filter_nAX;
char   filter_nBX;
char   filter_nCX;
char   filter_nAY;
char   filter_nBY;
char   filter_nCY;

long  reserved;
} TCFGMISC;
```

Members

Header

Header information about the structure.

av_sync_enable

- 0: Disable WIS Audio/Video Synchronization algorithm
- 1: Enable WIS Audio/Video Synchronization algorithm

iip_enable

Specifies if enabling Input Image Processing. Only valid when sensor pixel format is RGB Bayer.

- 0: Disable IIP
- 1: Enable IIP

vbi_enable

- 0: Disable VBI
- 1: Enable VBI

four_channel_enable

If four_channel_enable is set, each frame of the encoded stream will be divided into four quadrants. Motion search will be confined in each quadrant and will not be performed in other quadrants.

- 0: Disable four channel feature
- 1: Enable four channel feature

h_filter_mode

The mode of pre-filtering in a horizontal direction.

- 0: No pre-filtering in the horizontal direction before encoding
- 1: Median filter applied in the horizontal direction before encoding
- 2: Linear filter applied in the horizontal direction before encoding

v_filter_mode

The mode of pre-filtering in a vertical direction.

- 0: No pre-filtering in the vertical direction before encoding
- 1: Median filter applied in the vertical direction before encoding
- 2: Linear filter applied in the vertical direction before encoding

filter_nAX

filter_nBX

filter_nCX

The coefficients of linear filter in horizontal direction. Valid only if the h_filter_mode equals to two.

filter_nAX, filter_nBX, and filter_nCX correspond to precedent pixel, current pixel, and following pixel respectively. These three coefficients are all 5-bit values. filter_nBX is an unsigned value. filter_nAX and filter_nCX are signed values, with the highest bit indicating the sign and the rest four bits indicating the absolute value.

Typically, if filter_nAX and filter_nCX are positive, the filter will be a low-pass filter. Otherwise, if filter_nAX and filter_nCX are negative, the filter is a high-pass filter. A typical requirement for the coefficients is:

filter_nAX + filter_nBX + filter_nCX = 16.

filter_nAY

filter_nBY

filter_nCY

The coefficients of linear filter in vertical direction. Valid only if v_filter_mode is 2.

filter_nAY, filter_nBY and filter_nCY correspond to precedent pixel, current pixel, and following pixel respectively. These three coefficients are all 5-bit values. filter_nBY is an unsigned value. filter_nAY and filter_nCY are signed values, with the highest bit indicating the sign and the remaining four bits indi-

cating the absolute value.

Typically, if filter_nAY and filter_nCY are positive, the filter will be a low-pass filter. If filter_nAY and filter_nCY are negative, the filter is a high-pass filter. A typical requirement for the coefficients is:

filter_nAY + filter_nBY + filter_nCY = 16.

reserved

Reserved for future use.

TCFGVIDEO Structure

The TCFGVIDEO structure describes a complete video setting, including miscellaneous setting, stream setting, resolution setting, frame rate setting, and bitrate control setting.

Syntax

```
typedef struct
{
TCFGMISC  misccfg;
TCFGSTREAM  strcfg;
TCFGRESOLUTION rescfg;
TCFGFRAMERATE fpscfg;
TCFGBRCTRL  ctlcfg;
} TCFGVIDEO;
```

Members

misccfg

A TCFGMISC structure for miscellaneous settings.

strcfg

A TCFGSTREAM structure for stream settings.

rescfg

A TCFGRESOLUTION structure for resolution settings.

fpscfg

A TCFGFRAMERATE structure for frame rate settings.

ctlcfg

A TCFGBRCTRL structure for bitrate control settings.

TCFGVIDEOEX Structure

The TCFGVIDEOEX structure describes both system and video settings.

Syntax

```
typedef struct
{
TCFGSYSTEM  syscfg;
TCFGMISC  misccfg;
TCFGSTREAM  strcfg;
TCFGRESOLUTION rescfg;
TCFGFRAMERATE fpscfg;
TCFGBRCTRL  ctlcfg;
} TCFGVIDEOEX;
```

Members

syscfg

A TCFGSYSTEM structure for system settings.

misccfg

A TCFGMISC structure for miscellaneous settings.

strcfg

A TCFGSTREAM structure for stream settings.

rescfg

A TCFGRESOLUTION structure for resolution settings.

fpscfg

A TCFGFRAMERATE structure for frame rate settings.

ctlcfg

A TCFGBRCTRL structure for bit rate control settings.

TCFG_FORMAT_EXTENSION Structure

An extension to be appended to format information that will be set to video pin.

Syntax

```
typedef struct
{
TCFGSTREAM  strcfg;
TCFGFRAMERATE fpscfg;
TCFGRESOLUTION rescfg;
TCFGBRCTRL  ctlcfg;
} TCFG_FORMAT_EXTENSION;
```

Members

strcfg

A TCFGSTREAM structure for stream settings.

fpscfg

A TCFGFRAMERATE structure for frame rate settings.

rescfg

A TCFGRESOLUTION structure for resolution settings.

ctlcfg

A TCFGBRCTRL structure for bitrate control settings.

_VIDEO_CAPABILITIES Structure

Syntax

```
typedef struct
{
unsigned long  _num_of_system_configs;
TCFGSYSTEM_system  _configs[MAX_SYSTEM_CONFIG];

unsigned long  _num_of_stream_configs;
TCFGSTREAM  _stream_configs[MAX_STREAM_CONFIG];

unsigned long  _num_of_resolution_configs;
TCFGRESOLUTION
  _resolution_configs[MAX_RESOLUTION_CONFIG];

unsigned long  _num_of_framerate_configs;
```

```
TCFGFRAMERATE
  _framerate_configs[MAX_FRAMERATE_CONFIG];

unsigned long  _num_of_associations;
TCFGASSOCIATION  _associations[MAX_ASSOCIATION];

unsigned long  _num_of_configurations;
TVCFG_ENTRY*  _configurations;
} _VIDEO_CAPABILITIES;
```

Members

_num_of_system_configs

The count of all system configurations.

_system_configs

An array of TCFGSYSTEM structures to hold all system configurations.

_num_of_stream_configs

The count of all stream configurations.

_stream_configs

An array of TCFGSTREAM structures to hold all stream configurations.

_num_of_resolution_configs

The count of all resolution configurations.

_resolution_configs

An array of TCFGRESOLUTION structures to hold all resolu-

tion configurations.

_num_of_framerate_configs

The count of all frame rate configurations.

_framerate_configs

An array of TCFGFRAMERATE structures to hold all frame rate configurations.

_num_of_associations

The count of all associations.

_associations

An array of TCFGASSOCIATION structures to hold all associations.

_num_of_configurations

The count of all video configuration entries.

_configurations

A pointer to TVCFG_ENTRY structures.

TCFGASSOCIATION Structure

The TCFGASSOCIATION structure allows users to define relationship between any two types of settings from system setting, stream setting, resolution setting, frame rate setting, and bitrate control setting, if any.

Syntax

```
typedef struct
{
Enum ASSOCIATION_TYPE_master_type;
unsigned  long  _master_id;
Enum ASSOCIATION_TYPE_slave_type;
unsigned long  _slave_id;
unsigned char  _associate_type;
} TCFGASSOCIATION;
```

Members

_master_type

Type of master video setting.

_master_id

ID of the specific master setting.

_slave_type

Type of slave video setting.

_slave_id

ID of the specific slave setting.

_associate_type

Type of this association. Refer to the Enumeration: ASSOCIATION_TYPE.

TVCFG_ENTRY Structure

The TVCFG_ENTRY structure describes one entry for video configuration.

Syntax

```
typedef struct
{
unsigned long stream_index;
unsigned long resolution_index;
unsigned long framerate_index;
} TVCFG_ENTRY;
```

Members

stream_index

Index of stream configuration.

resolution_index

Index of resolution configuration.

framerate_index

Index of frame rate configuration.

AUDIO_CONFIG Structure

Syntax

```
typedef struct _AUDIO_CONFIG
{
unsigned long Format;
unsigned long SampleRate;
unsigned long Channels;
unsigned long SampleBits;

unsigned short BlockAlign;
unsigned long AvgBytesPerSec;
unsigned short SamplesPerBlock;
unsigned short ExtSize;
} AUDIO_CONFIG;
```

Members

sormat

Audio format. Possible values are included in the Enumeration: AUDIO_FORMAT.

sampleRate

Audio sample rate, in byte. Possible values are 44100, 48000, etc for PCM, 48000 for ADPCM.

channels

Audio channels. Possible values are 1 for Mono and 2 for Stereo.

SampleBits

Audio sample bits. Possible values are 8 bits and 16 bits for PCM, 4 bits for ADPCM.

STATISTIC Structure

Syntax

```
typedef struct _STATISTIC
{
UINT32  VideoByte;
UINT32  FrameCount;
} STATISTIC;
```

Members

VideoByte

Total video bytes obtained since starting capturing.

FrameCount

Total video frames obtained since starting capturing.

REVISION_INFO Structure

Syntax

```
typedef struct {
int DriverMajor;
int DriverMinor;
int BoardRevision;
char BoardName[MAX_NAME];
int BoardCapability;
int MaxBandWidth;
int SourceWidth;
int SourceHeight;
} REVISION_INFO;
```

Members

DriverMajor

Major revision number of driver.

DriverMinor

Minor revision number of driver.

BoardRevision

Revision number of reference board.

BoardName

Name of reference board.

BoardCapability

An integer with each bit representing one kind of capability of board, using values in Enumeration: BOARD_CAP.

MaxBandWidth

Reserved for future use.

SourceWidth

Width of source video.

SourceHeight

Height of source video.

Enumerations

EVideoFormat Enumeration

Syntax

```
typedef enum
{
MPEG1 = 0x00,
MPEG2 = 0x01,
H261 = 0x02,
H263 = 0x03,
MPEG4 = 0x04,
MPEG4XGO = 0x05,
MPEG2X4 = 0x06,
MOTIONJPEG = 0x08,
DV = 0x09,
H26L = 0x20,
GO = 0x40
} EVideoFormat;
```

ESequenceMode Enumeration

Syntax

```
typedef enum
{
IONLY = 1,
IPONLY = 2,
IPB = 3,
IPBDROP = 4
} ESequenceMode;
```

TV_STANDARD Enumeration

Syntax

```
typedef enum
{
    TVStandard_None= 0x00000000,
    TVStandard_NTSC_M= 0x00000001,
    TVStandard_NTSC_M_J= 0x00000002,
    TVStandard_NTSC_433= 0x00000004,

    TVStandard_PAL_B= 0x00000010,
    TVStandard_PAL_D= 0x00000020,
    TVStandard_PAL_G= 0x00000040,
    TVStandard_PAL_H= 0x00000080,
    TVStandard_PAL_I= 0x00000100,
    TVStandard_PAL_M= 0x00000200,
    TVStandard_PAL_N= 0x00000400,

    TVStandard_PAL_60= 0x00000800,

    TVStandard_SECAM_B= 0x00001000,
    TVStandard_SECAM_D= 0x00002000,
    TVStandard_SECAM_G= 0x00004000,
    TVStandard_SECAM_H= 0x00008000,
    TVStandard_SECAM_K= 0x00010000,
    TVStandard_SECAM_K1= 0x00020000,
    TVStandard_SECAM_L= 0x00040000,
    TVStandard_SECAM_L1= 0x00080000
} TV_STANDARD;
```

FilterMode Enumeration

Syntax

```
typedef enum
{
GO7007SB_MIDIAN= 1,
GO7007SB_LOWPASS= 2,
GO7007SB_NOFILTER= 0
} FilterMode;
```

MPEG4_MODE Enumeration

Syntax

```
enum MPEG4_MODE
{
WIS_MPEG4= 0,
DIVX_MPEG4= 1,
MICROSOFT_MPEG4= 2,
};
```

FLAGS_STREAM Enumeration

Syntax

```
enum FLAGS_STREAM
{
FLAGS_STREAM_COMPRESS_MODE= 0x00000001,
FLAGS_STREAM_SEQUENCE_MODE= 0x00000002,
FLAGS_STREAM_GOP_MODE= 0x00000004,
FLAGS_STREAM_GOP_SIZE= 0x00000008,
FLAGS_STREAM_MPEG4_MODE= 0x00000010,
FLAGS_STREAM_DEINTERLACE_MODE= 0x00000020,
FLAGS_STREAM_SEARCH_RANGE= 0x00000040,
```

```
FLAGS_STREAM_GOPHEAD_ENABLE= 0x00000080,
FLAGS_STREAM_SEQHEAD_ENABLE= 0x00000100,
FLAGS_STREAM_ASPECT_RATIO= 0x00000200,
FLAGS_STREAM_DVD_COMPLIANT= 0x00000400,
FLAGS_STREAM_MPEG4_MANDETORY=
  FLAGS_STREAM_COMPRESS_MODE +

  FLAGS_STREAM_MPEG4_MODE,
};
```

FLAGS_FRAMERATE Enumeration

Syntax

```
enum FLAGS_FRAMERATE
{
FLAGS_FRAMERATE_FRAMERATE = 0x00000001,
FLAGS_FRAMERATE_IVTC_ENABLE = 0x00000002,
FLAGS_FRAMERATE_DROP_FRAME = 0x00000004,
FLAGS_FRAMERATE_TVSTANDARD = 0x00000008,
FLAGS_FRAMERATE_MANDETORY =
  FLAGS_FRAMERATE_FRAMERATE +

  FLAGS_FRAMERATE_TVSTANDARD,
};
```

FLAGS_RESOLUTION Enumeration

Syntax

```
enum FLAGS_RESOLUTION
{
FLAGS_RESOLUTION_WIDTH=0x00000001,
FLAGS_RESOLUTION_HEIGHT=0x00000002,
FLAGS_RESOLUTION_H_SUBWINDOW=0x00000004,
FLAGS_RESOLUTION_V_SUBWINDOW=0x00000008,
FLAGS_RESOLUTION_SCALE_OFFSET=0x00000010,
FLAGS_RESOLUTION_SUBSAMPLE=0x00000100,
FLAGS_RESOLUTION_TVSTANDARD=0x00000200,
FLAGS_RESOLUTION_MAX_BITRATE=0x00000400,
FLAGS_RESOLUTION_MIN_BITRATE=0x00000800,
FLAGS_RESOLUTION_H_SUBOFFSET=0x00001000, // used
  only in parser
FLAGS_RESOLUTION_V_SUBOFFSET=0x00002000, // used
  only in parser
FLAGS_RESOLUTION_H_SCALE_ENABLE =0x00004000, //
  used only in parser
FLAGS_RESOLUTION_V_SCALE_ENABLE =0x00008000, //
  used only in parser


FLAGS_RESOLUTION_MANDETORY =
  FLAGS_RESOLUTION_WIDTH

  +FLAGS_RESOLUTION_HEIGHT

  +FLAGS_RESOLUTION_TVSTANDARD

  +FLAGS_RESOLUTION_MAX_BITRATE

  + FLAGS_RESOLUTION_MIN_BITRATE,
};
```

FLAGS_BITRATE Enumeration

Syntax

```
enum FLAGS_BITRATE
{
FLAGS_BITRATE_TARGET= 0x00000004,
FLAGS_BITRATE_PEAK= 0x00000008,
FLAGS_BITRATE_VBV_BUFFER= 0x00000010,
FLAGS_BITRATE_CONVERGE_SPEED= 0x00000020,
FLAGS_BITRATE_LAMBDA= 0x00000040,
FLAGS_BITRATE_Q= 0x00000080,
FLAGS_BITRATE_IPBQ= 0x00000100,
FLAGS_BITRATE_IQ= 0x00000200, // used only in
  parser
FLAGS_BITRATE_PQ= 0x00000400, // used only in
  parser
FLAGS_BITRATE_BQ= 0x00000800, // used only in
  parser

FLAGS_BITRATE_MANDETORY= FLAGS_BITRATE_TARGET +

  FLAGS_BITRATE_Q
};
```

FLAGS_MISC Enumeration

Syntax

```
enum FLAGS_MISC
{
FLAGS_MISC_AV_SYNC_ENABLE= 0x00000001,
FLAGS_MISC_IIP_ENABLE= 0x00000002,
FLAGS_MISC_VBI_ENABLE= 0x00000004,
FLAGS_MISC_FOUR_CHANNEL_ENABLE= 0x00000008,
```

```
FLAGS_MISC_FILTER= 0x00000010,

FLAGS_MISC_MANDETORY= 0
};
```

SENSOR_CAPABILITIES Enumeration

Syntax

```
enum SENSOR_CAPABILITIES
{
CAP_SENSOR_VIDEO_SOURCE= 0x00000001,

CAP_SENSOR_VIDEO_BRIGHTNESS= 0x00000004,
CAP_SENSOR_VIDEO_BRIGHTNESS_AUTO= 0x00000008,

CAP_SENSOR_VIDEO_CONTRAST= 0x00000010,
CAP_SENSOR_VIDEO_CONTRAST_AUTO= 0x00000020,

CAP_SENSOR_VIDEO_HUE= 0x00000040,
CAP_SENSOR_VIDEO_HUE_AUTO= 0x00000080,

CAP_SENSOR_VIDEO_SATURATION= 0x00000100,
CAP_SENSOR_VIDEO_SATURATION_AUTO= 0x00000200,

CAP_SENSOR_VIDEO_SHARPNESS= 0x00000400,
CAP_SENSOR_VIDEO_SHARPNESS_AUTO= 0x00000800,

CAP_SENSOR_VIDEO_GAMMA= 0x00001000,
CAP_SENSOR_VIDEO_GAMMA_AUTO= 0x00002000,

CAP_SENSOR_VIDEO_WHITEBALANCE= 0x00004000,
```

```
CAP_SENSOR_VIDEO_WHITEBALANCE_AUTO= 0x00008000,

CAP_SENSOR_VIDEO_BACKLIGHT_COMPENSATION=
  0x00010000,
CAP_SENSOR_VIDEO_BACKLIGHT_COMPENSATION_AUTO =
  0x00020000,

CAP_SENSOR_VIDEO_COLORENABLE= 0x00040000,
};
```

Remark

A DWORD with each bit represents one kind of sensor capability.

AUDIO_CAPS Enumeration

Syntax

```
enum AUDIO_CAPS
{
CAP_AUDIO_FORMAT_PCM= 0x00000001,
CAP_AUDIO_FORMAT_ADPCM_MS= 0x00000002,
CAP_AUDIO_FORMAT_ADPCM_IMA= 0x00000004,
CAP_AUDIO_FORMAT_ALAW= 0x00000008,
CAP_AUDIO_FORMAT_ULAW= 0x00000010,
CAP_AUDIO_FORMAT_MP3= 0x00000020,

CAP_AUDIO_SAMPLERATE_8K= 0x00000100,
CAP_AUDIO_SAMPLERATE_11025= 0x00000200,
CAP_AUDIO_SAMPLERATE_16K= 0x00000400,
CAP_AUDIO_SAMPLERATE_22050= 0x00000800,
CAP_AUDIO_SAMPLERATE_32K= 0x00001000,
CAP_AUDIO_SAMPLERATE_44100= 0x00002000,
```

```
CAP_AUDIO_SAMPLERATE_48K= 0x00004000,

CAP_AUDIO_CHANNEL_MONO= 0x00010000,
CAP_AUDIO_CHANNEL_STEREO= 0x00020000,

CAP_AUDIO_SAMPLE_8BIT= 0x00040000,
CAP_AUDIO_SAMPLE_16BIT= 0x00080000,
};
```

Remark

A DWORD with each bit represents one kind of audio capability.

AUDIO_FORMAT Enumeration

Syntax

```
enum AUDIO_FORMAT
{
AUDIO_FORMAT_PCM=1,
AUDIO_FORMAT_ADPCM_MS=2,
AUDIO_FORMAT_ADPCM_IMA=11,
AUDIO_FORMAT_ALAW,
AUDIO_FORMAT_ULAW,
AUDIO_FORMAT_MP3=0x55
};
```

Remark

This enumeration lists various kinds of audio formats.

ASSOCIATION_TYPE Enumeration

Syntax

```
enum ASSOCIATION_TYPE
{
TYPE_SYSTEM_CONFIG,
TYPE_STREAM_CONFIG,
TYPE_RESOLUTION_CONFIG,
TYPE_BITRATE_CONFIG,
TYPE_FRAMERATE_CONFIG
};
```

BOARD_CAP Enumeration

Syntax

```
typedef enum
{
BC_VIDEO= 0x00000001,
BC_AUDIO= 0x00000002,
BC_TVTUNER= 0x00000004,
BC_XBAR= 0x00000008,
BC_VBI= 0x00000010,
} BOARD_CAP;
```

Filter Interfaces

Included in this chapter are descriptions of the interfaces exposed by the WDM streaming capture filter (ADLINK Hardware MPEG4 Device filter).

For Microsoft DirectShow interfaces follow these links: IAMAnalogVideoDecoder, IAMCameraControl, IAMDroppedFrames, IAMVideoProcAmp, IBaseFilter, IKsPropertySet, and

ISpecifyPropertyPages.

Alternatively, visit http://msdn.microsoft.com/library/ and from the left panel navigation, select Graphics and Multimedia -> DirectX -> SDK Documentation -> DirectX 9.0 (C++) -> DirectShow -> DirectShow Reference -> Interfaces for a complete list of standard DirectShow filter interfaces references.

The ADLINK Hardware MPEG4 Device private interfaces are described in this chapter.

IGOChip Interface

IGOChip::SetVideoConfig

The SetVideoConfig method sets the video configurations.

Syntax

```
HRESULT SetVideoConfig(
TCFG_FORMAT_EXTENSION* pConfig,
unsigned int* pError
);
```

Parameters

pConfig: [Out] Pointer to a structure TCFG_FORMAT_EXTENSION that contains video configurations.

pError: [Out] Error information

Return Value

HRESULT

Remarks

Normally, format information for DirectShow applications is configured via the IAMStreamConfig interface. This interface is

exposed by both video and audio pins of WIS driver. The video and audio capabilities of the driver, the mean time, and the default format of these capabilities can be retrieved by using this interface. It is common to have multiple capabilities for both audio and video. Follow the instructions below to configurate:

Inspect all capabilities to check which capability is the one you want to set, using IAMStreamConfig::GetStreamCaps.

The default format for this capability can be modified as needed. This format is the second out parameter of the Get-StreamCaps.

Use IAMStreamConfig->SetFormat to set the modified format to the driver.

The following sample code - SetPinFormat function shows the video configuration setup process. The audio configuratation can also be set in a similar way.

The proprietary interface IGOChip::SetVideoConfig is necessary here is due to a known issue on the VideoInfoHeader format, preventing the format information to be set in the standard way. The code in the SetPinFormat function indicates this patch. After this problem is solved, there will be no need to use any non-standard interface.

The private format information is appended as an extension (TCFG_FORMAT_EXTENSION structure) to the normal format information of DirectShow, (AM_MEDIA_TYPE->pbFormat). The format size of cbFormat reflects this extension.

Sample Code

```
void
  CVideoControlPropertyPage::SetPinFormat(IAMStrea
  mConfig* stream_config,

  TCFGVIDEOEX* video_config)
{
    AM_MEDIA_TYPE* pmt;
    VIDEO_STREAM_CONFIG_CAPS caps;

    if ( stream_config == NULL ) return;

    int caps_count = 0, caps_size = 0;
    stream_config-
  >GetNumberOfCapabilities(&caps_count,
  &caps_size);

    char szDebugInfo[1000];

    for ( int i = 0 ; i < caps_count ; i ++ )
    {
        HRESULT hr = stream_config-
  >GetStreamCaps(i, &pmt, (BYTE*)&caps);
        if ( FAILED(hr) ) {
  OutputDebugString("[wisproxy]: GetSteamCaps
  Failed!"); continue; }

        unsigned long normal_format_size;
      switch ( video_config->strcfg.compress_mode
  )
     {
       case MPEG1:
       {
if ( pmt->subtype != MEDIASUBTYPE_MPEG1Payload )
  goto next_stream_caps;
```

```
if ( pmt->formattype != FORMAT_MPEGVideo ) goto
next_stream_caps;

MPEG1VIDEOINFO* format = (MPEG1VIDEOINFO*)pmt-
>pbFormat;

if ( format->hdr.bmiHeader.biWidth !=
(int)video_config->rescfg.width )

   goto next_stream_caps;

if ( format->hdr.bmiHeader.biHeight !=
(int)video_config->rescfg.height )

   goto next_stream_caps;


normal_format_size =
SIZE_MPEG1VIDEOINFO(format);

format->hdr.AvgTimePerFrame =
(ULONGLONG)(10010000000) / video_config-
>fpscfg.frame_rate;

format->hdr.bmiHeader.biWidth = video_config-
>rescfg.width;

format->hdr.bmiHeader.biHeight = video_config-
>rescfg.height;

format->hdr.bmiHeader.biSizeImage =
video_config->rescfg.width * video_config-
>rescfg.height * 3 / 2;

format->hdr.dwBitRate = video_config-
>ctlcfg.target_bitrate;


if ( pmt->cbFormat > normal_format_size )

{

   assert( pmt->cbFormat == normal_format_size +


sizeof(TCFG_FORMAT_EXTENSION) );

TCFG_FORMAT_EXTENSION* extension =
(TCFG_FORMAT_EXTENSION*)(pmt->pbFormat +
normal_format_size);

                       extension->_stream =
video_config->strcfg;
```

```
 extension->_framerate = video_config->fpscfg;
 extension->_resolution = video_config->rescfg;
 extension->_bitrate = video_config->ctlcfg;
             }

sprintf(szDebugInfo, "MPEG1 width: %d height: %d
fps: %d bps: %d",
      format->hdr.bmiHeader.biWidth,
      format->hdr.bmiHeader.biHeight,
      long(format->hdr.AvgTimePerFrame),
      format->hdr.dwBitRate);

   OutputDebugString(szDebugInfo);

   break;
 }
case MPEG2:
 {
   if ( pmt->subtype != MEDIASUBTYPE_MPEG2_VIDEO
) goto next_stream_caps;
   if ( pmt->formattype != FORMAT_MPEG2Video )
goto next_stream_caps;
  MPEG2VIDEOINFO* format = (MPEG2VIDEOINFO*)pmt-
>pbFormat;
   if ( format->hdr.bmiHeader.biWidth !=
(int)video_config->rescfg.width )
      goto next_stream_caps;
   if ( format->hdr.bmiHeader.biHeight !=
(int)video_config->rescfg.height )
      goto next_stream_caps;

   normal_format_size =
SIZE_MPEG2VIDEOINFO(format);
   format->hdr.AvgTimePerFrame =
```

```
(ULONGLONG)(10010000000) / video_config-
>fpscfg.frame_rate;
        format->hdr.bmiHeader.biWidth =
video_config->rescfg.width;
        format->hdr.bmiHeader.biHeight =
video_config->rescfg.height;
        format->hdr.bmiHeader.biSizeImage =
video_config->rescfg.width * video_config-
>rescfg.height * 3 / 2;
        format->hdr.dwBitRate = video_config-
>ctlcfg.target_bitrate;
        format->hdr.dwPictAspectRatioX =
video_config->rescfg.width;
        format->hdr.dwPictAspectRatioY =
video_config->rescfg.height;


       if ( pmt->cbFormat > normal_format_size )
        {
            assert( pmt->cbFormat ==
normal_format_size +

sizeof(TCFG_FORMAT_EXTENSION) );
        TCFG_FORMAT_EXTENSION* extension =
(TCFG_FORMAT_EXTENSION*)(pmt->pbFormat +
normal_format_size);
        extension->_stream = video_config-
>strcfg;
        extension->_framerate = video_config-
>fpscfg;
        extension->_resolution = video_config-
>rescfg;
        extension->_bitrate = video_config-
>ctlcfg;
    }


   sprintf(szDebugInfo, "MPEG2 width: %d height:
%d fps: %d bps: %d",
```

```
            format->hdr.bmiHeader.biWidth,
            format->hdr.bmiHeader.biHeight,
            long(format->hdr.AvgTimePerFrame),
            format->hdr.dwBitRate);

        OutputDebugString(szDebugInfo);

        break;
      }
case MPEG4:
case H263:
case MOTIONJPEG:
   {
        if ( pmt->formattype != FORMAT_VideoInfo )
goto next_stream_caps;
         VIDEOINFOHEADER* format =
(VIDEOINFOHEADER*)pmt->pbFormat;
         if ( format->bmiHeader.biWidth !=
(int)video_config->rescfg.width ) goto
next_stream_caps;
         if ( format->bmiHeader.biHeight !=
(int)video_config->rescfg.height ) goto
next_stream_caps;
        if ( video_config->strcfg.compress_mode ==
MPEG4 )
         {
            switch ( video_config-
>strcfg.mpeg4_mode )
            {
              case DIVX_MPEG4:
                 if ( format-
>bmiHeader.biCompression !=
FCC_FORMAT_DIVX_MPEG4)
                 goto next_stream_caps;
              break;
```

```
          case MICROSOFT_MPEG4:
            if ( format->bmiHeader.biCompression
!= FCC_FORMAT_MICROSOFT_MPEG4 )
                goto next_stream_caps;
            break;
          case WIS_MPEG4:
            if ( format->bmiHeader.biCompression
!= FCC_FORMAT_WIS_MPEG4 )
                goto next_stream_caps;
            break;
          default:
            assert(false);
      }
  }
 else if ( video_config->strcfg.compress_mode ==
H263 )
  {
       if ( format->bmiHeader.biCompression !=
FCC_FORMAT_H263 )
          goto next_stream_caps;
                }
   else if ( video_config->strcfg.compress_mode
== MOTIONJPEG )
                {
        if ( format->bmiHeader.biCompression !=
FCC_FORMAT_MOTION_JPEG )
          goto next_stream_caps;
                }


          normal_format_size = format-
>bmiHeader.biSize + SIZE_PREHEADER –

sizeof(TCFG_FORMAT_EXTENSION);
          format->AvgTimePerFrame =
```

```
(ULONGLONG)(10010000000) / video_config-
>fpscfg.frame_rate;
       format->bmiHeader.biWidth = video_config-
>rescfg.width;
        format->bmiHeader.biHeight =
video_config->rescfg.height;
        format->bmiHeader.biSizeImage =
video_config->rescfg.width * video_config-
>rescfg.height * 3 / 2;
        format->dwBitRate = video_config-
>ctlcfg.target_bitrate;


       if ( pmt->cbFormat > normal_format_size )
        {
           assert( pmt->cbFormat ==
normal_format_size +

sizeof(TCFG_FORMAT_EXTENSION) );
           TCFG_FORMAT_EXTENSION* extension =
(TCFG_FORMAT_EXTENSION*)(pmt->pbFormat +
normal_format_size);
           extension->_stream = video_config-
>strcfg;
           extension->_framerate = video_config-
>fpscfg;
          extension->_resolution = video_config-
>rescfg;
           extension->_bitrate = video_config-
>ctlcfg;


           if ( video_config->strcfg.mpeg4_mode
== MICROSOFT_MPEG4 )
           {
              char* seq_header;
              UINT32 seq_length =
FormatMPEG4StreamSEQHeader(&m_VideoCaps,
extension, &seq_header);
```

```
                memcpy( pmt->pbFormat +
normal_format_size - seq_length, seq_header,
seq_length);
            }

            if ( m_pIGOChipConfig ) // patch
            {
               IGOChip* pIGOChip;
               m_pIGOChipConfig-
>QueryInterface(IID_IGOChip,
reinterpret_cast<void**>(&pIGOChip));
               unsigned int error;
               pIGOChip-
>SetVideoConfig(extension, &error);
               pIGOChip->Release();
            }
       }

      sprintf(szDebugInfo, "videoinfo width: %d
height: %d fps: %d bps: %d",
               format->bmiHeader.biWidth,
               format->bmiHeader.biHeight,
               long(format->AvgTimePerFrame),
               format->dwBitRate);

       OutputDebugString(szDebugInfo);

       break;
   }
default:
   assert(false);
   DeleteMediaType(pmt);
   return;
}
```

```
AM_MEDIA_TYPE* pmt1;

hr = stream_config->GetFormat(&pmt1);
DeleteMediaType(pmt1);

hr = stream_config->SetFormat(pmt);
if ( FAILED(hr) ) { OutputDebugString("wisproxy:
 set pin format failed"); };
DeleteMediaType(pmt);

hr = stream_config->GetFormat(&pmt);
DeleteMediaType(pmt);

return;

next_stream_caps:
 DeleteMediaType(pmt);
}
}
```

IGOChipConfig Interface

Note:	This interface has now been phased out. It will continue to be supported for backward compatibility with existing applications, but new applications and filters should not use this interface. The functionality of this interface can be achieved by using Microsoft DirectShow interfaces.

1. IGOChipConfig::GetVideoConfig

The GetVideoConfig method retrieves the video configurations.

Syntax

```
HRESULT GetVideoConfig(
     TCFGVIDEOEX *pVal
);
```

Parameters

pVal: [Out] Pointer to a structure TCFGVIDEOEX to receive video configurations.

Return Value

HRESULT

Related Items

IGOChip::SetVideoConfig()

2. IGOChipConfig::GetVideoSource

The **GetVideoSource** method retrieves the video source that is in use.

Syntax

```
HRESULT GetVideoSource(
     unsigned int *pVal
);
```

Parameters

pVal: [Out] Pointer to an integer that represents video source in use. 0 represents S-video and 1 represents composite.

Return Value

HRESULT

3. IGOChipConfig::SetVideoSource

The SetVideoSource method sets the video source as either S-video or composite.

Syntax

```
HRESULT SetVideoSource(
     unsigned int newVal
);
```

Parameters

newVal: [In]Specifies what kind of video source is in use. 0 represents S-video and 1 represents composite.

Return Value

HRESULT

4. IGOChipConfig::GetSensorCapability

The GetSensorCapability method retrieves the sensor capabilities.

Syntax

```
HRESULT GetSensorCapability(
     unsigned int *pVal
);
```

Parameters

pVal: [Out] Pointer to an unsigned integer that is Enumeration: SENSOR_CAPABILITIES.

Return Value

HRESULT

5. IGOChipConfig::GetStatisticInfo

The GetStatisticInfo method retrieves the statistical information about video bytes and frames obtained since starting the capture.

Syntax

```
HRESULT GetStatisticInfo(
        STATISTIC *pVal
);
```

Parameters

pVal: [Out] Pointer to a STATISTIC structure to receive statistic info.

Return Value

HRESULT

6. IGOChipConfig::GetVideoCapabilities

The GetVideoCapabilities method retrieves the information about video capabilities.

Syntax

```
HRESULT GetVideoCapabilities(
    _VIDEO_CAPABILITIES* pCaps
        );
```

Parameters

pCaps: [Out] Pointer to a _VIDEO_CAPABILITIES structure to receive video capabilities.

Return Value

HRESULT

7. IGOChipConfig::GetAudioConfig

The GetAudioConfig method retrieves the audio configurations.

Syntax

```
HRESULT GetAudioConfig(
    AUDIO_CONFIG *pConfig
      );
```

Parameters

pConfig: [Out] Pointer to a structure AUDIO_CONFIG to receive audio configurations.

Return Value

HRESULT

Related Items

IGOChipConfig::SetAudioConfig()

8. IGOChipConfig::SetAudioConfig

The SetAudioConfig method sets the audio configurations.

Syntax

```
HRESULT SetAudioConfig(
    AUDIO_CONFIG *pConfig
        );
```

Parameters

pConfig: [Out] Pointer to a structure AUDIO_CONFIG that contains audio configurations.

Return Value

HRESULT

Related Items

IGOChipConfig::GetAudioConfig()

9. IGOChipConfig::GetAudioCapability

The GetAudioCapability method retrieves the audio capabilities.

Syntax

```
HRESULT GetAudioCapability(
   unsigned int *pAudioCap
    );
```

Parameters

pAudioCap: [Out] Pointer to an unsigned integer that is Enumeration: AUDIO_CAPS.

Return Value

HRESULT

IGOInfo Interface

1. IGOInfo::GetRevisionInfo

The GetRevisoinInfo method retrieves revision information of driver and board.

Syntax

```
HRESULT GetRevisionInfo(
REVISION_INFO *pRevInfo
);
```

Parameters

pRevInfo: [In] A pointer to REVISION_INFO structure to hold driver and board revision information.

Return Value

HRESULT

2. IGOInfo::GetMacrovision

The GetMacrovision method ascertains whether the video stream is protected by Macrovision.

Syntax

```
HRESULT GetMacrovision(
int *pMacrovision
);
```

Parameters

pMacrovision: [In] 1 indicates the stream is protected. 0 indicates the stream is not protected.

Return Value

HRESULT

IAccessFunc Interface

This interface provides methods for accessing I2C, SPI and

GPIO.

1. IAccessFunc::I2C_WriteRegister

The I2C_WriteRegister method writes a single I2C register.

Syntax

```
HRESULT I2C_WriteRegister(
unsigned char DevAddr,
int AddrWidth,
unsigned short RegAddr,
unsigned char RegValue,
int I2CMode
);
```

Parameters

DevAddr: [In] Device address.

AddrWidth: [In] Length of register address, in bits. Typical values can either be 8 or 16.

RegAddr: [In] Register address. If its higher byte is 0, the register is considered to have an 8-bit address; otherwise, the address length is 16 bits.

RegValue: [In] Value to be written to the I^2C register.

I2Cmode: [In] I^2C mode. The value can be set at:

- 0x0000: Use I^2C protocol via on chip I^2C controller.
- 0x0001: Use SCCB protocol via on chip I^2C controller.
- 0x8000: Use I^2C protocol via Cypress I^2C controller.

Return Value

HRESULT

2. IAccessFunc::I2C_ReadRegister

The I2C_ReadRegister method reads a single I2C register.

Syntax

```
HRESULT I2C_ReadRegister(
Unsigned char DevAddr,
int AddrWidth,
unsigned short RegAddr,
unsigned char *pRegValue,
int I2CMode
);
```

Parameters

DevAddr: [In] Device address.

AddrWidth: [In] Length of register address, in bits. Typical values can be either 8 or 16.

RegAddr: [In] Register address. If its higher byte is 0, the register is considered to have an 8-bit address; otherwise, the address length is 16 bits.

pRegValue: [Out] Value read from the I^2C register.

I2CMode: [In] I^2C mode. The value can be set at:

- 0x0000: Use I^2C protocol via on chip I^2C controller.
- 0x0001: Use SCCB protocol via on chip I^2C controller.
- 0x8000: Use I^2C protocol via Cypress I^2C controller.

Return Value

HRESULT

3. IAccessFunc::I2C_BurstWriteRegister

The I2C_BurstWriteRegister method writes multiple continuous I2C registers (burst mode).

Syntax

```
HRESULT I2C_BurstWriteRegister(
unsigned char DevAddr,
int AddrWidth,
unsigned short StartRegAddr,
int RegNum,
unsigned char *pRegValue,
int I2CMode
);
```

Parameters

DevAddr: [In] Device address.

AddrWidth: [In] Length of register address, in bits. Typical values can be either 8 or 16.

StartRegAddr: [In] Address of the first register. If its higher byte is 0, the register is considered to have an 8-bit address; otherwise, the address length is 16 bits.

RegNum: [In] Number of registers to be written.

pRegValue: [In] Pointer to an array of values to be written to the I^2C registers.

I2CMode: [In] I^2C mode. The value can be set at:

- 0x0000:Use I^2C protocol via on chip I^2C controller.
- 0x0001:Use SCCB protocol via on chip I^2C controller.
- 0x8000:Use I^2C protocol via Cypress I^2C controller.

Return Value

HRESULT

4. IAccessFunc::I2C_BurstReadRegister

The I2C_BurstReadRegister method reads multiple continuous I2C registers (burst mode).

Syntax

```
HRESULT I2C_BurstReadRegister(
unsigned char DevAddr,
int AddrWidth,
unsigned short StartRegAddr,
int RegNum,
unsigned char *pRegValue,
int I2CMode
);
```

Parameters

DevAddr: [In] Device address.

AddrWidth: [In] Length of register address, in bits. Typical values can either be 8 or 16.

StartRegAddr: [In] Address of the first register. If its higher byte is 0, the register is considered to have an 8-bit address; otherwise the address length is 16 bits.

RegNum: [In] Number of registers to be read.

pRegValue: [Out] Pointer to an array which will receive the values read from I^2C registers.

I2CMode: [In] I^{2}C mode. The value can be set at:

- 0x0000:Use I^{2}C protocol via on chip I^{2}C controller.
- 0x0001:Use SCCB protocol via on chip I^{2}C controller.
- 0x8000:Use I^{2}C protocol via Cypress I^{2}C controller.

Return Value

HRESULT

5. IAccessFunc::SPI_WriteRegister

The SPI_WriteRegister method writes a single SPI register.

Syntax

```
HRESULT SPI_WriteRegister(
int OpLen,
unsigned char OpCode,
int AddrLen,
unsigned short RegAddr,
int DataLen,
unsigned short RegData,
int SPIMode
);
```

Parameters

OpLen: [In] Operation code length, in bits. The typical range is 1 - 8.

OpCode: [In] Operation code.

AddrLen: [In] Length of register address, in bits. The typical range is 1 - 16.

RegAddr: [In] Register address.

DataLen: [In] Length of data, in bits. The typical range is 0 - 16.

RegData: [In] Value to be written to the SPI register.

SPI_mode: [In] SPI mode, a 16-bit data. Refer to the following table for definitions for each bit.

Bit	Name	Type	Default Value	Description
15:11	Reserved			
10	three_wire_en	RW	1'b0	1 = 3-wire is enabled; 0 = 3-wire is not enabled;
9	bst_end	RW	1'b0	1 = the next R/W access is the last access of a burst access; 0 = the next R/W access is not the last access of a burst access (only valid for burst mode)
8	sdo_separate	RW	1'b0	1 = pin sdi and pin sdo are separated; 0 = pin sdi and pin sdo are shared;
7	cs1_en_value	RW	1'b0	1 = the chip-select enable logic value for cs1 is 1 0 = the chip-select enable logic value for cs1 is 0
6	cs0_en_value	RW	1'b0	1 = the chip-select enable logic value for cs0 is 1 0 = the chip-select enable logic value for cs0 is 0
5	cs1_en	RW	1'b0	1 = output cs1 is enabled; 0 = output cs1 is disabled;
4	cs0_en	RW	1'b1	1 = output cs0 is enabled; 0 = output cs0 is disabled;
3	read	RW	1'b0	1 = do read access; 0 = do non-read access (write or other operation like erase, erase_write_enable/disable);

Bit	Name	Type	Default Value	Description
2	bst_rw	RW	1'b0	1 = burst R/W mode (burst R for 3-wire device is not supported, the read data of 3-wire device is half spi clock cycle later than that of spi device); 0 = single R/W mode;
1:0	spi_mode	RW	2'h0	2'h0 = spi mode 0; 2'h1 = spi mode 1; 2'h2 = spi mode 2; 2'h3 = spi mode 3; Note: for 3-wire, mode 0 should be used.

Table 2-10: SPI Control Register Definition

Return Value

HRESULT

6. IAccessFunc::SPI_ReadRegister

The SPI_ReadRegister method reads a single SPI register.

Syntax

```
HRESULT SPI_ReadRegister(
int OpLen,
unsigned char OpCode,
int AddrLen,
unsigned short RegAddr,
int DataLen,
unsigned short *pRegData,
int SPIMode
);
```

Parameters

OpLen: [In] Operation code length, in bits. The typical range is 1 - 8.

OpCode: [In] Operation code.

AddrLen: [In] Length of register address, in bits. The typical range is 1 - 16.

RegAddr: [In] Register address.

DataLen: Length of data, in bits. The typical range is 0 – 16.

pRegData: [Out] Value to be written to the SPI register.

SPIMode: [In] SPI mode, a 16-bit data. Refer to Table 1 SPI Control Register Definition for definitions of each bit.

Return Value

HRESULT

7. IAccessFunc::GPIO_WritePins

The GPIO_WritePins method toggles the signal on one or multiple GPIO pins.

Syntax

```
HRESULT GPIO_WritePins(
int WriteNum,
int *Index,
int *Value,
int Mode
);
```

Parameters

WriteNum: [In] Number of pins to write.

Index: [In] Indexes of GPIO pins.

Value: [In] The signal written to the GPIO pins. The value must be either 0 or 1.

Mode: [In] 0: On chip GPIO controller; 1: Cypress GPIO controller.

Return Value

HRESULT

8. IAccessFunc::GPIO_ReadPins

The GPIO_ReadPins method reads the signal on one or multiple GPIO pins.

Syntax

```
HRESULT GPIO_ReadPins(
int ReadNum,
int *Index,
int *Value,
int Mode
);
```

Parameters

ReadNum: [In] Number of pins to read.

Index: [In] Indexes of GPIO pins.

Value: [In] The signal read from the GPIO pins.

Mode: [In] 0: via on chip GPIO controller; 1: via Cypress GPIO controller.

Return Value

HRESULT

ADLINK Hardware MPEG4 Device GPIO Pin Definition

PIN	Type	FUNCTION
GPIO0	Input	Channel ID bit 0
GPIO1	Input	Channel ID bit 1
GPIO2	Input	Card ID bit 0 (setting by dip switch)
GPIO3	Input	Card ID bit 1 (setting by dip switch)
GPIO4	Input	Card ID bit 2 (setting by dip switch)

Table 2-11: ADLINK Hardware MPEG4 Device GPIO Pinout

IAdvanced Interface

This interface provides advanced access to CBUS registers and HPI registers. Some methods are for internal testing use purposes only, and hence are not documented in this document.

1. IAdvanced::ReadCBusRegFW

The ReadCBusRegFW method reads a single CBus register.

Syntax

```
HRESULT ReadCBusRegFW(
unsigned short Addr,
unsigned short *pData,
);
```

Parameters

Addr: [In] CBus register address.

pData: [Out] pointer to an unsigned short to hold the register value read.

Return Value

HRESULT

2. IAdvanced::WriteCBusRegFW

The WriteCBusRegFW method writes a single CBus register.

Syntax

```
HRESULT WriteCBusRegFW(
unsigned short Addr,
unsigned short Data,
);
```

Parameters

Addr: [In] CBus register address.

Data: [In] CBus register value.

Return Value

HRESULT

IMotionDetection Interface

WIS Windows SDK supports Motion Detection(MD). This IMotionDetection interface let upper application to send the configurations of MD to WIS firmware.

1. IMOTIONDETECTION::SetUpMDInterace

The SetUpMDInterace method sends MD configurations to firmware and starts MD. Data of MD will be gotten by interruption of WIS firmware after MD starts.

Note: The chip supports four regions when we use MD, but only three regions can be used and the fourth region is invalid temporarily.

Syntax

```
HRESULT SetUpMDInterface( [in] MDConfig_t* pData
  );
```

```
Parameters
pData: [In] structure contains MD input
  information.
#define MAX_REGIONS_OF_INTEREST 4
MDConfig_t structure:
typedef struct sPointCoords
{
int CoordsX;
int CoordsY;
} PointCorrds_t;

typedef struct sMDConfig
{
PointCorrds_t ULPoint[MAX_REGIONS_OF_INTEREST];
PointCorrds_t BRPoint[MAX_REGIONS_OF_INTEREST];
unsigned short
  u32SADThresholdValues[MAX_REGIONS_OF_INTEREST];
unsigned short
  u32MVThresholdValues[MAX_REGIONS_OF_INTEREST];
unsigned short
  u32SensitivityValues[MAX_REGIONS_OF_INTEREST];
}MDConfig_t;
```

Description

ULPoint Upper-left coordinate of region.

BRPoint Bottom-right coordinate of region.

U32SADThresholdValues The value of SADThreshold. This range of SADThreshold is from 0 to 32767. Higher the value, lower the sensitivity.

U32MVThresholdValues The value of MVThreshold. This range of MVThreshold is from 0 to 32767. Higher the value, lower the sensitivity.

u32SensitivityValues The value of Sensitivity. This range of Sensitivity is from 0 to 100. Higher the value, lower the sensitivity.

Return Value

HRESULT

2. IMOTIONDETECTION::StopMDInterface

The StopMDInterface method sends a group of default configurations to WIS firmwire and close MD.

Syntax

```
HRESULT StopMDInterface( );
```

Parameters

Null

Return Value

HRESULT

INotify Interface

The Interface of INotify provides a field that upper application communicates with lower driver by event.

1. INotify::EnableEvent

The EnableEvent method may notify application whenever a specific event action occurs.

Syntax

```
HRESULT EnableEvent(
ULONG event_id,
ULONG event_handle,
ULONG *cookie
);
```

Parameters

Event_id: [in] contain id of event that application want to enable it.

Event_handle: [in] a handle is defined by application.

Cookie: [out] point to an array that is used to store the event data

Return Value

HRESULT

2. INotify::GetEventData

The GetEventData method gets EventData from event which is produced by firmwire.

Syntax

```
HRESULT GetEventData(
ULONG event_id,
ULONG cookie,
ULONG buf_len,
unsigned char * buf,
ULONG *data_size
);
```

Parameters

Event_id: [in] contain id of event that application want to enable it.

cookie: [in] contain event data.

Buf_len: [in] Size, in bytes, of the buffer at buf

Buf: [in] Pointer to a buffer that receive data for the operation.

Data_size: [out] Pointer to a variable that receives the size, in bytes, of the data that stores in the buffer at buf

Return Value

HRESULT

3. INotify::DisableEvent

The DisableEvent method informs the KS object specified by Handle to stop notifying the

application whenever a specific event action occurs.

Syntax

```
HRESULT DisableEvent(
ULONG event_id,
ULONG cookie
);
```

Parameters

Event_id: [in] contain id of event that application want to enable it.

Cookie: [in] Pointer to a buffer that contains data that specifies the operation to perform

Return Value

HRESULT

IOSD Interfaces

ADLINK Hardware MPEG4 Device supports up to 94 Unicode character On Screen Display (OSD). This IOSD interface let upper application to send On Screen Display (OSD) string to the firmware, which will generate a corresponding OSD effect. The OSD string font size is limited to 94.

1. IOSD::Textout

The Textout method sends OSD frame to firmware to display. An OSD frame is composed multiple OSD strings. See OSD programming for details.

Syntax

HRESULT Textout(

OSDTextoutInfo info

);

Parameters

info: [In] structure contains OSD output information.

- OSDTextoutInfo structure:

typedef struct

{

unsigned short TotalLength;

unsigned short text[MAX_OSDSTRING_LEN];

} OSDTextoutInfo;

Description: TotalLength: Describes the length of OSD frame contained in text[].

Text[]: OSD frame in word, refer to OSD programming chapter for OSD frame structure.

Return Value

HRESULT

2. IOSD::Show

The Show method sends the firmware the command to start OSD.

Syntax

HRESULT Textout(

);

Parameters

Null

Return Value

HRESULT

Pin Interfaces

The pins of ADLINK Hardware MPEG4 Device filter expose Microsoft DirectShow interfaces: IAMBufferNegotiation, IAMStreamConfig, IAMStreamControl, IKsPin, IKsPropertySet, IStreamBuilder, IMediaSeeking, IPin, and IQualityControl. Follow the links of the interfaces for further detail.

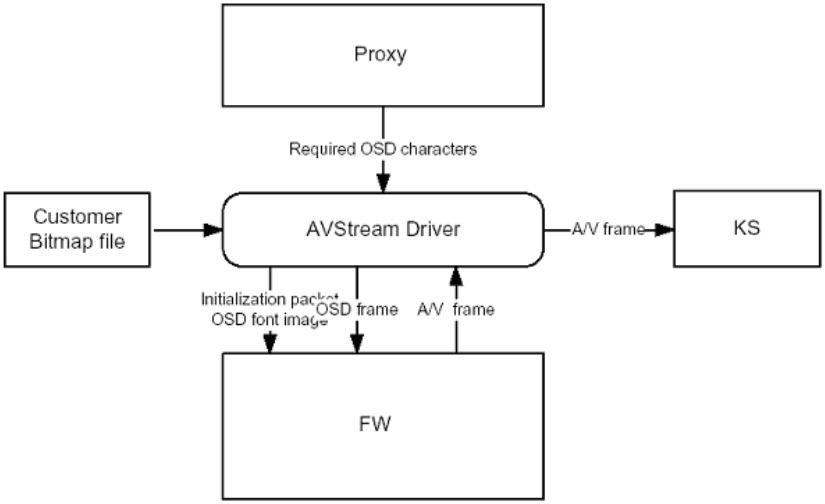
Alternatively, please visit http://msdn.microsoft.com/library/ and from the left panel navigation, select Graphics and Multimedia -> DirectX -> SDK Documentation -> DirectX 9.0 (C++) -> DirectShow -> DirectShow Reference -> Interfaces for a complete list of standard DirectShow filter interfaces references.

2.3 OSD Programming

Introduction to OSD

OSD (On-Screen-Display) is supported by the ADLINK Hardware MPEG4 Device. During the Motion Estimation and Compensation (MEC) stage, firmware can load the OSD bitmap and overlay with video, thus modifying output bitmap. By default, the ASCII bitmaps are prepared at boot-up and saved in DRAM. Customers can also choose or generate their own bitmaps, such as UNICODE. However, font base address space is limited to no more than 16-bit. Drivers can download the customized bitmap font file or the default font bitmap file into firmware, and enabled firmware to do OSD bitmap overlap to show user required characters.

Context



OSD Font Bitmaps

All OSD font bitmaps are stored in the off-chip DRAM. The address is from 0xA0100 to 0xAF8FF (if 8M SDRAM and IP_ONLY), or from 0x140100 to 0x14F8FF (if 8M SDRAM and IPB). Each font occupies 32 consecutive DWORD, supporting up to 1984 font bitmaps. After downloading to DRAM, the bitmaps are never changed.

OSD Fonts Display

There is a 192-WORD font index buffer in the on-chip SRAM. It is separated into two 96-WORD buffers: index_buffer1 (0x3A00-0x3A5F) and index_buffer2 (0x3A60-0x3ABF). At any time, one of them contains an OSD frame which is currently in use; the other is open for editing the next OSD frame. Here, an OSD frame means a layout of OSD fonts. A series of consecutive video frames may share one single OSD frame.

OSD Frame

An OSD frame is made up with at least one OSD string and one OSD_EOF. An OSD string starts with a ST_CD which is followed by a series of font base addresses, and ends with an OSD_EOS. An example is shown as follows:

ST_CD	ADDR0	ADDR1	...	ADDRn	OSD_EOS	ST_CD	ADDR0	...	OSD_EOS	OSD_EOF

ST_CD: 16-bit, as the macroblock coordinate (x, y) for the first font of this string.

ST_CD[15:8] = y, ST_CD[7:0] = x.

Example: Preview window is 720*480, max value of X is 720/16-1=44, max value of Y is 480/16-1 = 29.

ADDRx: 16-bit, as the base address for the its font of this string

Notes: n in ADDRn is from 0 to 93.

OSD_EOS: 0x0000

OSD_EOF: 0xAAAA.

OSD Algorithm

Before encoding each macroblock, the firmware makes an OSD function call. In this call, the chip decides if the current macroblock has an OSD font over it. If so, it obtains this font's base address and sends it to the DMA controller. Then, a 32-DWORD bitmap of this font will be overlapped to that macroblock. At the first macroblock (0, 0) of each video frame, the firmware determines which buffer (index_buffer1 or index_buffer2) is in use. The firmware keeps comparing the ST_CD of each OSD string with current macroblock's coordinate until they match. After getting ST_CD, the firmware reads the next base address and sends it to the DMA controller, until an OSD_EOS is met.

Bitmap Stored in SDRAM

For every pixel in the bitmap, 4 bit data will be used to describe OSD behavior of the pixel. Format of BITMAP in SDRAM is:

bit 0							bit 31	
dword 0	P(0,0)	P(0,1)	P(0,2)	P(0,3)	P(0,4)	P(0,5)	P(0,6)	P(0,7)
.	P(1,0)	P(1,1)	P(1,2)	P(1,3)	P(1,4)	P(1,5)	P(1,6)	P(1,7)
.	…	…	…	…	…	…	…	…
.	P(15,0)	P(15,1)	P(15,2)	P(15,3)	P(15,4)	P(15,5)	P(15,6)	P(15,7)
.	P(0,8)	P(0,9)	P(0,10)	P(0,11)	P(0,12)	P(0,13)	P(0,14)	P(0,15)
.	P(1,8)	P(1,9)	P(1,10)	P(1,11)	P(1,12)	P(1,13)	P(1,14)	P(1,15)
.	…	…	…	…	…	…	…	…

P(x, y) means 4 bit data of pixel at column x and line y.

OSD Pixel Color (4-bit OSD Data)

For every pixel, there are 4 bits to present the OSD behavior. It means:

Bit(s)	Description
[3]	OSD mode
	0 – Background blending
	1 – Foreground blending
[2:0]	Alpha blending level (0 - 7)

For mode 0, the algorithm is:

$$X_d = \frac{C_0 \cdot \alpha + X_s \cdot (8 - \alpha) + 4}{8}$$

For mode 1, the algorithm is:

$$X_d = \frac{C_0 \cdot \alpha + C_1 \cdot (8 - \alpha) + 4}{8}$$

In the previous equations, X_s is the source data (Y or U or V), X_d is the result (Y or U or V), α is the alpha blending level which is defined in bit 2 to bit 0. C_0 is the background color and C_1 is the foreground color. As there are three channels (YUV), C_0 and C_1 could be programmable from C-Bus for every channel.

So there are two ways to change the OSD color:

1. Change the YUV value in Fix_setting.txt

// osd setting

osdcfg.DoOSD = 1

osdcfg.OSDyc0 = 0

osdcfg.OSDyc1 = 255

osdcfg.OSDuc0 = 0

osdcfg.OSDuc1 = 128

osdcfg.OSDvc0 = 0

osdcfg.OSDvc1 = 128

2. Change the alpha blending level (a) as the above algorithm definition.

Refer the following YUV to RGB converting algorithm for detail.

Y = 0.299R + 0.587G + 0.114B

U = - 0.147R- 0.289G + 0.436B

V = 0.615R - 0.515G - 0.100B

Or

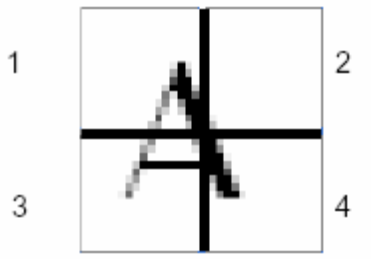
$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Know Limitations

1. Each OSD frame can only contain up to 90 fonts in the single OSD string case.
2. Font size can only be 16*16, which is macro block based.

How to display customer defined size bitmap

The firmware only can display 16*16 OSD bitmaps. However, customers can still show bigger bitmaps by proper software programming skills. For example, a 32*32 pixel bitmap, software can display 32*32's bitmap by dividing it into 4 small 16*16 bitmap, first download the divided 16*16 bitmap to the firmware, then display the small bitmap corresponding to their X,Y position. See following description:



To display a 32*32 pixel "A", we can divide a 32*32 "A" into 4 16*16 sub bitmap, 1, 2, 3, 4, then create a OSD frame containing 2 (or 4) OSD string:

2 OSD string frame

X1, Y1, address1, address 2, 0x0000, X3, Y3, address 3, address 4, 0x0000, 0xAAAA

4 OSD string frame

X1, Y1, address1, 0x0000, X2, Y2, address 2, X3, Y3, address 3, X4, Y4, address 4, 0x0000, 0xAAAA

Notes:

- Xn, Yn means the 1, 2, 3, 4 sub bitmap's X,Y position, for example, sub bitmap 1's X/Y is (0,0), sub bitmap 3's X/Y is (0,1)
- Address[n] means 1,2,3,4 sub bitmap's address in firmware, this address is decided when bitmap is downloaded.
- 32*32 bitmap are software features, customer application should know the bitmap address map in firmware and create the corresponding OSD frame which contains multiple OSD string, with corresponding X/Y value. The Avstream driver only provides API functions to download bitmap and write OSD frames to the firmware.

OSD features in Demo application

Customer scan develop their own font bitmap for their certain application.

We provided the following features by providing 6 demo font files:

Eng16_1.osd 16*16 ASCII characters, default color

Eng16_2.osd 16*16 ASCII characters, customized color

Eng_hollow.osd 32*32 ASCII characters, two colors (body/boarder)

Chn16_1.osd 16*16 Chinese characters, default

Chn16_2.osd 16*16 Chinese characters, customized color

Chn_hollow.osd 32*32 Chinese characters, two colors (body/

boarder)

- Basic font bitmap display
  - 16*16 bitmap size, customer defined font style and size
  - Customer defined alpha blending level and YUV (Can be converted to RGB) color
  - Single OSD string in OSD frame
- Advanced font bitmap display
  - 32*32 bitmap size, customer defined font style and size, which is simulated by software, to display the 4 sub-bitmaps for composing 32*32 bitmap.
  - Multiple OSD strings in OSD frame
  - Hollow font with different colors between font boarder and body.

OSD Data Structure

A filter interface (IOSD) is described to use OSD, through this interface, upper application send structured OSD frame to driver, driver writes the frame to the firmware. OSD data structure is defined as following fields:

```
#define MAX_OSDSTRING_LEN 96

typedef struct

{

unsigned short TotalLength; //Total length of OSD frame

unsigned short text[MAX_OSDSTRING_LEN]; //OSD frame

} OSDTextoutInfo;
```

3 Windows API Functions

3.1 Introduction to Windows API Functions

The Application Program Interface (API) functions are based on DirectX 9.0. DirectX 9.0 need to be installed, and then the API functions can work.

The main goal of API functions is to simplify the programming. DirectShow is good at developing media stream program. DirectX SDK is a powerful developer kit and provides many programming interfaces to developer. But most applications don't need the full range of DirectShow's capabilities; in fact, very few do. The API Functions reduce the interfaces and pack them to easy understanding functions. These functions are dedicated to PCI-MPG24. Users can use them to develop applications under Visual C++, Visual Basic, C++ Builder, and Delphi.

3.2 Function List

Some key words in the following table mean:

Preview: Displays video images coming from 'ADLINK Bt878 Video Capture' device.

Encoder: Saves media samples into a file. The media data come from 'ADLINK Hardware MPEG4 Device' device and its format is MPEG4.

PlayFile: Display images from a video file.

Category	Section	Function
System	3.4	Mpg24_PreviewOpen(CardNo)
		Mpg24_PreviewClose(CardNo)
		Mpg24_EncoderOpen(CardNo, PortNo)
		Mpg24_EncoderClose(CardNo, PortNo)
		Mpg24_PlayFileClose(Index)
		Mpg24_ReadSerial(CardNo, HighByte, LowByte)

Category	Section	Function
Configuration	3.5	Mpg24_PreviewGetImageRange(CardNo, Property, Min, Max, SteppingDelta, Default)
		Mpg24_PreviewSetImageConfig(CardNo, Property, Value)
		Mpg24_PreviewGetImageConfig(CardNo, Property, Value)
		Mpg24_PreviewSetVideoFormat(CardNo, FormatIndex, Value)
		Mpg24_PreviewGetVideoFormat(CardNo, FormatIndex, Value)
		Mpg24_PreviewSetDisplay(CardNo, Handle, X, Y, AutoShow)
		Mpg24_PreviewSetCustomSize(CardNo, Width, Height)
		Mpg24_EncoderGetImageRange(CardNo, PortNo, Property, Min, Max, SteppingDelta, Default)
		Mpg24_EncoderSetImageConfig(CardNo, PortNo, Property, Value)
		Mpg24_EncoderGetImageConfig(CardNo, PortNoProperty, Value)
		Mpg24_EncoderSetVideoFormat(CardNo, PortNo, FormatIndex, Value)
		Mpg24_EncoderGetVideoFormat(CardNo, PortNo, FormatIndex, Value)
		Mpg24_EncoderSetFile(CardNo, PortNo, FileName)
		Mpg24_PlayFileSetFile(Index, FileName)
		Mpg24_PlayFileSetDisplay(Index, Handle, X, Y, Width, Height, AutoShow)
Action	3.6	Mpg24_PreviewRun(CardNo)
		Mpg24_PreviewPause(CardNo)
		Mpg24_PreviewStop(CardNo)
		Mpg24_PreviewSaveImage(CardNo, FileName)
		Mpg24_PreviewSelectChannel(CardNo, PortNo, Mode)
		Mpg24_PreviewShow(CardNo, Visible)
		Mpg24_EncoderSetOSD(CardNo, PortNo, OSDText, Len)
		Mpg24_EncoderRun(CardNo, PortNo)
		Mpg24_EncoderStop(CardNo, PortNo)
		Mpg24_PlayFileRun(Index)
		Mpg24_PlayFilePause(Index)
		Mpg24_PlayFileStop(Index)
		Mpg24_PlayFileShow(Index, Visible)
		Mpg24_PlayFileSaveImage(Index, FileName)
Watchdog	3.7	Mpg24_WatchdogConfig(CardNo, TriggerInterval)
		Mpg24_WatchdogEnable(CardNo)
		Mpg24_WatchdogDisable(CardNo)
		Mpg24_WatchdogTrigger(CardNo)
IO	3.8	Mpg24_SetGPIO(CardNo, PortNo, Status)
		Mpg24_GetGPIO(CardNo, PortNo, Status)
		Mpg24_WriteEEPROM(CardNo, Offset, Value)
		Mpg24_ReadEEPROM(CardNo, Offset, Value)

Category	Section	Function
Misc.	3.9	Mpg24_PreviewCallback(CardNo, CallbackProc)
		Mpg24_PreviewGetStatus(CardNo, Status)
		Mpg24_EncoderCallback(CardNo, PortNo, CallbackProc)
		Mpg24_EncoderGetStatus(CardNo, PortNo, Status)
		Mpg24_PlayFileCallback(Index, CallbackProc)
		Mpg24_PlayFileGetStatus(Index, Status)
		Mpg24_GetLastErrorInfo(ErrorInfo)

3.3 Setting Up the Build Environment

Include Files

All applications using the APIs need include the file shown the following table.

Include File	Description
PCI_MPG24.h	The header file required for all C/C++ applications.
PCI_MPG24.bas	The function definitions required for all VB applications.

These files are located on the directory [Program Files]\ADLINK\PCI-MPG24\Include\.

Library File

All C/C++ applications using the APIs need the library file shown in the following table.

Library File	Description
PCI_MPG24.lib	Exports API function definitions. Required for all C/C++ applications.

These files are located on the directory [Program Files]\ADLINK\PCI-MPG24\Include\.

DLL Files

All applications using the APIs need the DLL file shown in the following table.

Library File	Description
PCI_MPG24.dll	Dynamic link library. Required for all applications.

These files are located on the directory [WINDOWS]\system32\.

3.4 System Functions

Mpg24_PreviewOpen(CardNo)

Mpg24_EncoderOpen(CardNo, PortNo)

The Open function initialize the device and open it for later use. Application need call this function before using any other functions. One device only can be opened once before you close it.

There is no Mpg24_PlayFileOpen() function. The system will automatically allocate the resources when application call other PlayFile functions.

Syntax

C/C++

```
int Mpg24_PreviewOpen(short CardNo)
int Mpg24_EncoderOpen(short CardNo, short PortNo)
```

VB

```
Mpg24_ PreviewOpen (ByVal CardNo as Integer) as
     Long
Mpg24_ EncoderOpen (ByVal CardNo as Integer,
     ByVal PortNo as Integer) as Long
```

Delphi

```
Mpg24_PreviewOpen(CardNo:Smallint):Smallint
Mpg24_EncoderOpen(CardNo:Smallint, PortNo:Small-
     int) : Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7. The card ID can be got from S1 dip
     switch. Please refer to Hardware Reference
     chapter.
```

PortNo: The video input channel which value is between 0 and 3.

Return Value

O : No error.
<0 : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_PreviewClose(CardNo)

Mpg24_EncoderClose(CardNo, PortNo)

Mpg24_PlayFileClose(Index)

The Close function closes the opened device and release the resources the device occupied.

Syntax

C/C++

```
int Mpg24_PreviewClose(short CardNo)
int   Mpg24_EncoderClose(short   CardNo,   short PortNo)
```

VB

```
Mpg24_ PreviewClose (ByVal CardNo as Integer) as Long
Mpg24_ EncoderClose (ByVal CardNo as Integer, ByVal PortNo as Integer) as Long
```

Delphi

```
Mpg24_PreviewClose(CardNo:Smallint):Smallint
Mpg24_EncoderClose(CardNo:Smallint, PortNo:Smallint) : Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.
PortNo: The video input channel which value is between 0 and 3.
Index: The index of playing back file which value is between 0 and 15.

Return Value

O : No error.

```
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_ReadSerial(CardNo, HighByte, LowByte)

The ReadSerial function reads the unique 48 bits serial number. Each PCI-MPG24 has a ROM what stores a unique serial number. Application can check this number to enable your software function.

Syntax

C/C++

```
int Mpg24_ReadSerial(short CardNo, UINT *High-
     Byte, UINT *LowByte)
```

VB

```
Mpg24_ReadSerial(ByVal CardNo as Integer, ByRef
     HighByte as Long, ByRef LowByte as Long) as
     Long
```

Delphi

```
Mpg24_ReadSerial(CardNo:Smallint,   Var   Hight-
     Byte:Longint, Var LowByte:Longint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7. The card ID can be got from S1 dip
     switch. Please refer to Hardware Reference
     chapter.
HighByte: The upper 16-bit of serial number.
LowByte: The lower 32-bit of serial number.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

3.5 Configuration Functions

Mpg24_PreviewGetImageRange(CardNo, Property, Min, Max, SteppingDelta, Default)

Mpg24_EncoderGetImageRange(CardNo, PortNo, Property, Min, Max, SteppingDelta, Default)

The GetImageRange function retrieves the range and default value of a specified video property such as brightness, contrast, hue, saturation, gamma, and sharpness.

Syntax

C/C++

```
int Mpg24_PreviewGetImageRange(short CardNo, int
    Property, int *Min, int *Max, int *Step-
    pingDelta, int *Default)
int  Mpg24_EncoderGetImageRange(short  CardNo,
    short PortNo, int Property, int *Min, int
    *Max, int *SteppingDelta, int *Default)
```

VB

```
Mpg24_PreviewGetImageRange(ByVal CardNo As Inte-
    ger, ByVal Property As Long, ByRef Min As
    Long, ByRef Max As Long, ByRef SteppingDelta
    As Long, ByRef Default As Long) As Long
Mpg24_EncoderGetImageRange(ByVal CardNo As Inte-
    ger, ByVal PortNo as Integer, ByVal Property
    As Long, ByRef Min As Long, ByRef Max As
    Long, ByRef SteppingDelta As Long, ByRef
    Default As Long) As Long
```

Delphi

```
Mpg24_PreviewGetImageRange(CardNo:Smallint,
    Property:Longint, Var Min :Longint, Var
    Max:Longint, Var SteppingDelta:Longint, Var
    Default:Longint):Longint
Mpg24_EncoderGetImageRange(CardNo:Smallint,
    PortNo:Smallint, Property:Longint, Var Min
    :Longint, Var Max:Longint, Var Step-
    pingDelta:Longint, Var Default:Long-
    int):Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

PortNo: The video input channel which value is between 0 and 3.

Property

Property	Enum Value	Preview	Encoder
Brightness	0	O	O
Contrast	1	O	O
Hue	2	O	O
Saturation	3	O	O
Sharpness	4	O	O
Gamma	5	O	X
Color Enable	6	O	X
White Balance	7	O	X
Backlight Compensation	8	O	X

O: Support, X: Not support

Min: Minimum value of the property.

Max: Maximum value of the property.

SteppingDelta: Step size for the property. The step size is the smallest increment by which the property can change.

Default: Default value of the property.

Return Value

0 : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_PreviewGetImageConfig(CardNo, Property, Value)

Mpg24_EncoderGetImageConfig(CardNo, PortNo, Property, Value)

The GetImageConfig function retrieves video quality of a specified property.

Syntax

C/C++

```
int Mpg24_PreviewGetImageConfig(short CardNo, int
    Property, int *Value)
int  Mpg24_EncoderGetImageConfig(short  CardNo,
    short PortNo, int Property, int *Value)
```

VB

```
Mpg24_PreviewGetImageConfig(ByVal CardNo as Inte-
    ger, ByVal Property as Long, ByRef Value as
    Long) as Long
Mpg24_EncoderGetImageConfig(ByVal CardNo as Inte-
    ger, ByVal PortNo as Integer, ByVal Property
    as Long, ByRef Value as Long) as Long
```

Delphi

```
Mpg24_PreviewGetImageConfig(CardNo:Smallint,
    Property:Longint,  Var Value:Longint):Long-
    int
Mpg24_EncoderGetImageConfig(CardNo:Smallint,
    PortNo:Smallint,  Property:Longint,  Var
    Value:Longint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
    7. The card ID can be got from S1 dip
    switch. Please refer to Hardware Reference
    chapter.
PortNo: The video input channel which value is
    between 0 and 3.
Property: Please refer to section 7.5.1 for the
    meaning of property.
Value: The value of the property.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
    the detail description of error code.
```

Mpg24_PreviewSetImageConfig(CardNo, Property, Value)

Mpg24_EncoderSetImageConfig(CardNo, PortNo, Property, Value)

The SetImageConfig function sets video quality of a specified property.

Syntax

C/C++

```
int Mpg24_PreviewSetImageConfig(short CardNo, int
    Property, int Value)
int  Mpg24_EncoderSetImageConfig(short  CardNo,
    short PortNo, int Property, int Value)
```

VB

```
Mpg24_PreviewSetImageConfig(ByVal CardNo as Inte-
    ger, ByVal Property as Long, ByVal Value as
    Long) as Long
Mpg24_EncoderSetImageConfig(ByVal CardNo as Inte-
    ger, ByVal PortNo as Integer, ByVal Property
    as Long, ByVal Value as Long) as Long
```

Delphi

```
Mpg24_PreviewSetImageConfig(CardNo:Smallint,
    Property:Longint,  Value:Longint):Longint
Mpg24_EncoderSetImageConfig(CardNo:Smallint,
    PortNo:Smallint,          Property:Longint,
    Value:Longint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
    7.  The  card  ID  can  be  got  from  S1  dip
    switch. Please refer to Hardware Reference
    chapter.
PortNo: The video input channel which value is
    between 0 and 3.
Property:
Please refer to section 7.5.1 for the meaning of
    porperty.
Value: The value of the property.
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
    the detail description of error code.
```

Mpg24_PreviewGetVideoFormat(CardNo, FormatIndex, Value)

Mpg24_EncoderGetVideoFormat(CardNo, PortNo, FormatIndex, Value)

The GetVideoFormat function retrieves the value of a specified video configuration such as video standard, video size, and frame rate.

Syntax

C/C++

```
int   Mpg24_PreviewGetVideoFormat(short   CardNo,
      short FormatIndex, short *Value)
int   Mpg24_EncoderGetVideoFormat(short   CardNo,
      short   PortNo,   short   FormatIndex,   short
      *Value)
```

VB

```
Mpg24_PreviewGetVideoFormat(ByVal CardNo as Inte-
      ger,  ByVal  FormatIndex  as  Integer,  ByRef
      Value as Integer) as Long
Mpg24_EncoderGetVideoFormat(ByVal CardNo as Inte-
      ger, ByVal PortNo as Integer, ByVal Format-
      Index as Integer, ByRef Value as Integer) as
      Long
```

Delphi

```
Mpg24_PreviewGetVideoFormat(CardNo:Smallint,
      FormatIndex:Smallint,     Var   Value:Small-
      int):Longint
Mpg24_EncoderGetVideoFormat(CardNo:Smallint,
      PortNo:Smallint, FormatIndex:Smallint,  Var
      Value:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
      7.  The  card  ID  can  be  got  from  S1  dip
      switch. Please refer to Hardware Reference
      chapter.
PortNo: The video input channel which value is
      between 0 and 3.
```

FormatIndex:

Type	Enum Value	Value	Default	
			Preview	Encoder
Standard	0	0: NTSC 1: PAL	NTSC	NTSC
Size	1	0: FULL D1 1: CIF 2: QCIF 3: VGA 4: QVGA	CIF	FULL D1 (QCIF size not supported)
Frame Rate	2	0: 30 fps(NTSC), 25 fps(PAL) 1: 15 fps(NTSC), 12.5 fps(PAL) 2: 10 fps(NTSC), 8.3 fps(PAL) 3: 5 fps(NTSC), 5 fps(PAL) 4: 1 fps(NTSC), 1fps(PAL)	30 or 25 fps	30 or 25 fps
Target Bitrate	3	0: 4M bps 1: 2M bps 2: 1.5M bps 3: 1M bps 4: 750k bps 5: 500k bps 6: 384k bps	Not supported	4M bps
Codec	4	0: DivX 1: Microsoft	Not supported	DivX

P.S. Video Size:

- FULL D1: 720 x 480(NTSC), 720 x 576(PAL)
- CIF: 352 x 240(NTSC), 352 x 288(PAL)
- QCIF: 176 x 144(NTSC), 176 x 144(PAL)
- VGA: 640 x 480(NTSC), PAL doesn't support this format.
- QVGA: 320 x 240(NTSC), PAL doesn't support this format.

Value: The value of the specified format.

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewSetVideoFormat(CardNo, FormatIndex, Value)

Mpg24_EncoderSetVideoFormat(CardNo, PortNo, FormatIndex, Value)

The SetVideoFormat function sets the value of a specified video configuration. You can't set video format while preview or encoder is running.

Syntax

C/C++

```
int  Mpg24_PreviewSetVideoFormat(short  CardNo,
     short FormatIndex, short Value)
int  Mpg24_EncoderSetVideoFormat(short  CardNo,
     short  PortNo,  short  FormatIndex,  short
     Value)
```

VB

```
Mpg24_PreviewSetVideoFormat(ByVal CardNo as Inte-
     ger,  ByVal  FormatIndex  as  Integer,  ByVal
     Value as Integer) as Long
Mpg24_EncoderSetVideoFormat(ByVal CardNo as Inte-
     ger, ByVal PortNo as Integer, ByVal Format-
     Index as Integer, ByVal Value as Integer) as
     Long
```

Delphi

```
Mpg24_PreviewSetVideoFormat(CardNo:Smallint,
     FormatIndex:Smallint,          Value:Small-
     int):Longint
Mpg24_EncoderSetVideoFormat(CardNo:Smallint,
     PortNo:Smallint,      FormatIndex:Smallint,
     Value:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch. Please refer to Hardware Reference
     chapter.
PortNo: The video input channel which value is
     between 0 and 3.
FormatIndex: Please refer to section 7.5.4 for
     the meaning of FormatIndex.
Value: The value of the specified format.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewSetCustomSize(CardNo, Width, Height)

The SetCustomSize function sets a customizing size of video image. You also can use the Mpg24_PreviewSetVideoFormat function to set a predefined video size.

Syntax

C/C++

```
int   Mpg24_PreviewSetCustomSize(short   CardNo,
     short Width, short Height)
```

VB

```
Mpg24_PreviewSetCustomSize(ByVal CardNo as Inte-
     ger, ByVal Width as Integer, ByVal Height as
     Integer) as Long
```

Delphi

```
Mpg24_PreviewSetCustomSize(CardNo:Smallint,
     Width:Smallint,  Height:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch.  Please  refer  to  Hardware  Reference
     chapter.
Width: The width of video window.
Height: The height of video window.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewSetDisplay(CardNo, Handle, X, Y, AutoShow)

Mpg24_PlayFileSetDisplay(Index, Handle, X, Y, Width, Height, AutoShow)

The SetDisplay function sets properties on the video window. Application can use it to set the window owner, the position and dimensions of the window, and automatically showing the video window.

Syntax

C/C++

```
int  Mpg24_PreviewSetDisplay(short  CardNo,  int
     Handle, short X, short Y, short AutoShow)
int  Mpg24_PlayFileSetDisplay(short  Index,  int
     Handle, short X, short Y, short Width, short
     Height, short AutoShow)
```

VB

```
Mpg24_PreviewSetDisplay(ByVal CardNo as Integer,
     ByVal Handle as Long, ByVal X as Integer,
     ByVal Y as Integer, ByVal AutoShow as Inte-
     ger) as Long
Mpg24_PlayFileSetDisplay(ByVal Index as Integer,
     ByVal Handle as Long, ByVal X as Integer,
     ByVal Y as Integer, ByVal Width as Integer,
     ByVal Height as Integer, ByVal AutoShow as
     Integer) as Long
```

Delphi

```
Mpg24_PreviewSetDisplay(CardNo:Smallint,     Han-
     dle:Longint,     X:Smallint,     Y:Smallint,
     AutoShow:Smallint):Longint
Mpg24_PlayFileSetDisplay(Index:Smallint,     Han-
     dle:Longint,     X:Smallint,     Y:Smallint,
     Width:Smallint,            Height:Smallint,
     AutoShow:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch. Please refer to Hardware Reference
     chapter.
```

```
Index: The index of playing back file which value
       is between 0 and 15.
Handle: Specifies a handle to the parent window
       that the video image show on it. If Handle
       is 0, the video image will show on a new
       window.
X: Sets the video window's x-coordinate.
Y: Sets the video window's y-coordinate.
Width: The width of video window.
Height: The height of video window.
AutoShow: Specifies whether the video renderer
       automatically shows the video window when it
       receives video data.
0: FALSE; Hide the video window.
other: TRUE; Show the video window.
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
       the detail description of error code.
```

Mpg24_EncoderSetFile(CardNo, PortNo, FileName)

Mpg24_PlayFileSetFile(Inex, FileName)

The SetFile function sets the name of the file into which media samples will be written or from which media samples will be read.

Syntax

C/C++

```
int   Mpg24_EncoderSetFile(short   CardNo,   short
      PortNo, char *FileName)
int   Mpg24_PlayFileSetFile(short   Index,   char
      *FileName)
```

VB

```
Mpg24_EncoderSetFile(ByVal  CardNo  as  Integer,
      ByVal PortNo as Integer, ByVal FileName as
      String) as Long
Mpg24_PlayFileSetFile(ByVal  Index  as  Integer,
      ByVal FileName as String) as Long
```

Delphi

```
Mpg24_EncoderSetFile(CardNo:Smallint,
      PortNo:Smallint, FileName:String):Longint
```

Mpg24_PlayFileSetFile(Index:Smallint, FileName:String):Longint

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

PortNo: The video input channel which value is between 0 and 3.

Index: The index of playing back file which value is between 0 and 15.

FileName: The name of the media file.

Return Value

O : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

3.6 Action Functions

Mpg24_PreviewRun(CardNo)

Mpg24_EncoderRun(CardNo, PortNo)

Mpg24_PlayFileRun(Index)

The Run function starts to display the video window or starts to save the media samples into a file.

Syntax

C/C++

```
int Mpg24_PreviewRun(short CardNo)
int Mpg24_EncoderRun(short CardNo, short PortNo)
int Mpg24_PlayFileRun(short Index)
```

VB

```
Mpg24_PreviewRun(ByVal CardNo as Integer) as Long
Mpg24_EncoderRun(ByVal CardNo as Integer, ByVal
     PortNo as Integer) as Long
Mpg24_PlayFileRun(ByVal Index as Integer) as Long
```

Delphi

```
Mpg24_PreviewRun(CardNo:Smallint) :Longint
Mpg24_EncoderRun(CardNo:Smallint,   PortNo:Small-
     int):Longint
Mpg24_PlayFileRun(Index:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch.  Please  refer  to  Hardware  Reference
     chapter.
PortNo: The video input channel which value is
     between 0 and 3.
Index: The index of playing back file which value
     is between 0 and 15.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewPause(CardNo)

Mpg24_PlayFilePause(Index)

The Pause function pauses playing the video window.

Syntax

C/C++

```
int Mpg24_PreviewPause(short CardNo)
int Mpg24_PlayFilePause(short Index)
```

VB

```
Mpg24_PreviewPause(ByVal CardNo as Integer) as
     Long
Mpg24_PlayFilePause(ByVal Index as Integer) as
     Long
```

Delphi

```
Mpg24_PreviewPause(CardNo:Smallint) :Longint
Mpg24_PlayFilePause(Index:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7. The card ID can be got from S1 dip
     switch. Please refer to Hardware Reference
     chapter.
Index: The index of playing back file which value
     is between 0 and 15.
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewStop(CardNo)

Mpg24_EncoderStop(CardNo, PortNo)

Mpg24_PlayFileStop(Index)

The Stop function stops displaying the video window or closes the saving media file.

Syntax

C/C++

```
int Mpg24_PreviewStop(short CardNo)
int Mpg24_EncoderStop(short CardNo, short PortNo)
int Mpg24_PlayFileStop(short Index)
```

VB

```
Mpg24_PreviewStop(ByVal CardNo as Integer) as
     Long
Mpg24_EncoderStop(ByVal CardNo as Integer, ByVal
     PortNo as Integer) as Long
Mpg24_PlayFileStop(ByVal Index as Integer) as
     Long
```

Delphi

```
Mpg24_PreviewStop(CardNo:Smallint) :Longint
Mpg24_EncoderStop(CardNo:Smallint, PortNo:Small-
     int):Longint
Mpg24_PlayFileStop(Index:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7. The card ID can be got from S1 dip
     switch. Please refer to Hardware Reference
     chapter.
PortNo: The video input channel which value is
     between 0 and 3.
Index: The index of playing back file which value
     is between 0 and 15.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewShow(CardNo, Visible)

Mpg24_PlayFileShow(Index, Visible)

The Show function shows or hides displaying video window. This function take effect only when video window runs.

Syntax

C/C++

```
int  Mpg24_PreviewShow(short  CardNo,  short  Visi-
     ble)
int  Mpg24_PlayFileShow(short  Index,  short  Visi-
     ble)
```

VB

```
Mpg24_PreviewShow(ByVal CardNo as Integer, ByVal
     Visible as Integer) as Long
Mpg24_PlayFileShow(ByVal Index as Integer, ByVal
     Visible as Integer) as Long
```

Delphi

```
Mpg24_PreviewShow(CardNo:Smallint,          Visi-
     ble:Smallint):Longint
Mpg24_PlayFileShow(Index:Smallint,          Visi-
     ble:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch.  Please  refer  to  Hardware  Reference
     chapter.
Index: The index of playing back file which value
     is between 0 and 15.
Visible:  Specifies  whether  to  show  or  hide  the
     window. The value can be:
0: FALSE; Hide the video window.
other: TRUE; Show the video window.
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_PreviewSelectChannel(CardNo, PortNo, Mode)

The SelectChannel function routes the input sources to the video window.

Syntax

C/C++

```
int   Mpg24_PreviewSelectChannel(short   CardNo,
      short PortNo, short Mode)
```

VB

```
Mpg24_PreviewSelectChannel(ByVal CardNo as Inte-
      ger, ByVal PortNo as Integer, ByVal Mode as
      Integer) as Long
```

Delphi

```
Mpg24_PreviewSelectChannel(CardNo:Smallint,
      PortNo:Smallint, Mode:Smallint):Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

PortNo: The video input channel which value is between 0 and 3.

Mode: Specifies whether to display QUAD video or single channel video. The value can be:

0: QUAD mode; Hide the video window. The value of PortNo doesn't matter.

1: Single mode; According to the value of PortNo, shows the input source.

Return Value

0 : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_PreviewSaveImage(CardNo, FileName)

Mpg24_PlayFileSaveImage(Index, FileName)

The SaveImage function captures one frame from the video samples and writes it to a bitmap (.bmp) file.

Syntax

C/C++

```
int   Mpg24_PreviewSaveImage(short  CardNo,  char
      *FileName)
int   Mpg24_PlayFileSaveImage(short  Index,  char
      *FileName)
```

VB

```
Mpg24_PreviewSaveImage(ByVal CardNo as Integer,
      ByVal FileName as String) as Long
Mpg24_PlayFileSaveImage(ByVal Index as Integer,
      ByVal FileName as String) as Long
```

Delphi

```
Mpg24_PreviewSaveImage(CardNo:Smallint,File-
      Name:String):Longint
Mpg24_PlayFileSaveImage(Index:Smallint,File-
      Name:String):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
      7. The card ID can be got from S1 dip
      switch. Please refer to Hardware Reference
      chapter.
Index: The index of playing back file which value
      is between 0 and 15.
FileName: The bitmap file name.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
      the detail description of error code.
```

Mpg24_EncoderSetOSD(CardNo, PortNo, OSDText, Len)

The SetOSD function blends text and video to the saving file.

Syntax

C/C++

```
int  Mpg24_EncoderSetOSD(short  CardNo,  short
     PortNo, BYTE *OSDText, short Len)
```

VB

```
Mpg24_EncoderSetOSD(ByVal  CardNo  as  Integer,
     ByVal PortNo as Integer, ByRef OSDText As
     Byte, ByVal Len As Integer) as Long
```

Delphi

```
Mpg24_EncoderSetOSD(CardNo:Smallint,
     PortNo:Smallint,     Var     OSDText:Byte,
     Len:Smallint):Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

PortNo: The video input number which value is between 0 and 3.

OSDText: The OSD string contains header, Unicode text, and EOF. Please refer to chapter 2.3 for detail description.

Len: How many words the OSD text has.

Return Value

0 : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

3.7 Watchdog Functions

Mpg24_WatchdogConfig(CardNo, TriggerInterval)

The WatchdogConfig function sets the maximum time that application need to re-trigger Watchdog register to prevent computer from restarting.

Syntax

C/C++

```
int Mpg24_WatchdogConfig(short CardNo, short TriggerInterval)
```

VB

```
Mpg24_WatchdogConfig(ByVal CardNo as Integer, ByVal TriggerInterval As Integer) as Long
```

Delphi

```
Mpg24_WatchdogConfig(CardNo:Smallint, TriggerInterval:Smallint):Longint
```

Parameters

`CardNo`: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

`TriggerInterval`: The trigger time. The value can be:

- ▷ 0: 0 second; Reset computer immediately.
- ▷ 1: 8 seconds
- ▷ 2: 16 seconds
- ▷ 3: 32 seconds

Return Value

`0` : No error.

`<0` : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_WatchdogEnable(CardNo)

The WatchdogEnable function enables watch dog.

Syntax

C/C++

```
int Mpg24_WatchdogEnable(short CardNo)
```

VB

```
Mpg24_WatchdogEnable(ByVal CardNo as Integer) as
     Long
```

Delphi

```
Mpg24_WatchdogEnable(CardNo:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7. The card ID can be got from S1 dip
     switch. Please refer to Hardware Reference
     chapter.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_WatchdogDisable(CardNo)

The WatchdogDisable function disables watch dog.

Syntax

C/C++

```
int Mpg24_WatchdogDisable(short CardNo)
```

VB

```
Mpg24_WatchdogDisable(ByVal CardNo as Integer) as
     Long
```

Delphi

```
Mpg24_WatchdogDisable(CardNo:Smallint):Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

Return Value

0 : No error.
<0 : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_WatchdogTrigger(CardNo)

The WatchdogTrigger function triggers watch dog to prevent computer from restarting.

Syntax

C/C++

```
int Mpg24_WatchdogTrigger(short CardNo)
```

VB

```
Mpg24_WatchdogTrigger(ByVal CardNo as Integer) as
     Long
```

Delphi

```
Mpg24_WatchdogTrigger(CardNo:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7. The card ID can be got from S1 dip
     switch. Please refer to Hardware Reference
     chapter.
```

Return Value

```
0 : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

3.8 IO Functions

Mpg24_SetGPIO(CardNo, PortNo, Status)

The SetGPIO function sets the GPIO pin to high voltage or low voltage.

Syntax

C/C++

```
int  Mpg24_SetGPIO(short  CardNo,  short  PortNo,
     BYTE Status)
```

VB

```
Mpg24_SetGPIO(ByVal  CardNo  as  Integer,  ByVal
     PortNo as Integer, ByVal Status as Byte) as
     Long
```

Delphi

```
Mpg24_SetGPIO(CardNo:Smallint,   PortNo:Smallint,
     Status:Byte):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch.  Please  refer  to  Hardware  Reference
     chapter.
PortNo: The GPIO number which value is between 0
     and 3.
Status: Sets the voltage level of output pin. The
     vale is:
0: Low voltage
1: High voltage
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_GetGPIO(CardNo, PortNo, Status)

The GetGPIO function retrieves the status of GPIO pin.

Syntax

C/C++

```
int  Mpg24_GetGPIO(short  CardNo,  short  PortNo,
     BYTE *Status)
```

VB

```
Mpg24_GetGPIO(ByVal  CardNo  as  Integer,  ByVal
     PortNo as Integer, ByRef Status as Byte) as
     Long
```

Delphi

```
Mpg24_GetGPIO(CardNo:Smallint,   PortNo:Smallint,
     Var Status:Byte):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
     7.  The  card  ID  can  be  got  from  S1  dip
     switch. Please refer to Hardware Reference
     chapter.
PortNo: The GPIO number which value is between 0
     and 3.
Status: The voltage level of output pin. The vale
     is:
0: Low voltage
1: High voltage
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
     the detail description of error code.
```

Mpg24_WriteEEPROM(CardNo, Offset, Value)

The WriteEEPROM function writes the onboard EEPROM.

Syntax

C/C++

```
int Mpg24_WriteEEPROM(short CardNo, BYTE Offset,
    BYTE Value)
```

VB

```
Mpg24_WriteEEPROM(ByVal CardNo as Integer, ByVal
    Offset as Byte, ByVal Value as Byte) as Long
```

Delphi

```
Mpg24_WriteEEPROM(CardNo:Smallint,   Offset:Byte,
    Value:Byte):Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

Offset: The address in EEPROM which value is between 0 and 127.

Value: The value of the address which value is between 0 and 255.

Return Value

O : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_ReadEEPROM(CardNo, Offset, Value)

The WriteEEPROM function read the value of onboard EEPROM.

Syntax

C/C++

```
int Mpg24_ReadEEPROM(short CardNo, BYTE Offset,
    BYTE *Value)
```

VB

```
Mpg24_ReadEEPROM(ByVal CardNo as Integer, ByVal
    Offset as Byte, ByRef Value as Byte) as Long
```

Delphi

```
Mpg24_ReadEEPROM(CardNo:Smallint,   Offset:Byte,
    Var Value:Byte):Longint
```

Parameters

CardNo: The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

Offset: The address in EEPROM which value is between 0 and 127.

Value: The value of the address which value is between 0 and 255.

Return Value

0 : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

3.9 Miscellaneous Functions

Mpg24_PreviewCallback(CardNo, CallbackProc)

Mpg24_EncoderCallback(CardNo, PortNo, CallbackProc)

Mpg24_PlayFileCallback(Index, CallbackProc)

The Callback function sets which procedure to be called when the frame data ready. User need write a callback procedure following the predefined format.

Syntax

C/C++

```
int   Mpg24_PreviewCallback(short   CardNo,   void
      (__stdcall  *CallbackProc)(BYTE  *  pBuffer,
      int lBufferSize))
int   Mpg24_EncoderCallback(short   CardNo,   short
      PortNo, void (__stdcall *CallbackProc)(BYTE
      * pBuffer, int lBufferSize))
int   Mpg24_PlayFileCallback(short   Index,   void
      (__stdcall  *CallbackProc)(BYTE  *  pBuffer,
      int lBufferSize))
```

VB

```
Mpg24_PreviewCallback(ByVal  CardNo  As  Integer,
      ByVal CallbackProc As Long) as Long
Mpg24_EncoderCallback(ByVal  CardNo  As  Integer,
      ByVal PortNo As Integer, ByVal CallbackProc
      As Long) as Long
Mpg24_PlayFileCallback(ByVal  Index  As  Integer,
      ByVal CallbackProc As Long) as Long
```

Delphi

```
Mpg24_PreviewCallback(CardNo:Smallint,  Callback-
      Proc:CallbackFunc):Longint
Mpg24_EncoderCallback(CardNo:Smallint,
      PortNo:Smallint      CallbackProc:Callback-
      Func):Longint
Mpg24_PlayFileCallback(Index:Smallint,  Callback-
      Proc:CallbackFunc):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
      7.  The  card  ID  can  be  got  from  S1  dip
```

switch. Please refer to Hardware Reference chapter.

PortNo: The video input channel which value is between 0 and 3.

Index: The index of playing back file which value is between 0 and 15.

CallbackProc: The prototype of the callback function is CallbackProc(BYTE * pBuffer, int lBufferSize).

pBuffer: The pointer of frame data.

lBufferSize: The size of frame data.

Return Value

0 : No error.

<0 : Error occurred. Please see section 3.10 for the detail description of error code.

Mpg24_PreviewGetStatue(CardNo, Status)

Mpg24_EncoderGetStatus(CardNo, PortNo, Status)

Mpg24_PlayFileGetStatus(Index, Status)

The GetStatus function retrieves the status – running, paused, or stopped and detects the state of signal input – video present or video loss.

Syntax

C/C++

```
int  Mpg24_PreviewGetStatus(short  CardNo,  short
     *Status)
int  Mpg24_EncoderGetStatus(short  CardNo,  short
     PortNo, short *Status)
int  Mpg24_PlayFileGetStatus(short  Index,  short
     *Status)
```

VB

```
Mpg24_PreviewGetStatus(ByVal  CardNo  As  Integer,
     ByRef Status As Integer) as Long
Mpg24_EncoderGetStatus(ByVal  CardNo  As  Integer,
     ByVal  PortNo  As  Integer,  ByRef  Status  As
     Integer) as Long
Mpg24_PlayFileGetStatus(ByVal  Index  As  Integer,
     ByRef Status As Integer) as Long
```

Delphi

```
Mpg24_PreviewGetStatus(CardNo:Smallint, Var Sta-
    tus:Smallint):Longint
Mpg24_EncoderGetStatus(CardNo:Smallint,
    PortNo:Smallint Var Status:Smallint):Long-
    int
Mpg24_PlayFileGetStatus(Index:Smallint, Var Sta-
    tus:Smallint):Longint
```

Parameters

```
CardNo: The card ID which value is between 0 and
    7. The card ID can be got from S1 dip
    switch. Please refer to Hardware Reference
    chapter.
PortNo: The video input channel which value is
    between 0 and 3.
Index: The index of playing back file which value
    is between 0 and 15.
Status: The Value can be:
Bit 0-1: 0: stopped, 1: paused, 2:running
Bit 2: 0: video loss, 1: video present
```

Return Value

```
O : No error.
<0 : Error occurred. Please see section 3.10 for
    the detail description of error code.
```

Mpg24_GetLastErrorInfo(ErrorInfo)

The GetLastErrorInfo function retrieves the last error message.

Syntax

C/C++

```
int Mpg24_GetLastErrorInfo(TCHAR *ErrorInfo)
```

Parameters

```
ErrorInfo: The last error message. Please refer
    to section 3.10 for detailed.
```

Return Value

```
O : No error.
E_UNKOWN_ERROR : Please see section 3.10 for the
    detail description of error code.
```

3.10 Error Codes

When application gets a non-zero code, you can query its meaning from below table.

Error constant	Value	Description
	0	No error.
E_INVALID_ARGUMENT	-1	Invalid argument. Maybe wrong CardNo , wrong PortNo, or the arguments exceed its range.
E_IN_USE	-2	This device is already in use. You need to close it and then you can use it again.
E_DEVICE_NOT_FOUND	-3	Can't find this device.
E_DEVICE_NOT_READY	-4	Device is not initialized.
E_FILE_NOT_FOUND	-5	This file cannot be found.
E_DIRECTX_NOT_INSTALL	-6	DirectX Runtime or DirectX SDK doesn't be installed.
E_UNKOWN_FORMAT	-7	The video format is not the predefined one that the error is returned by calling Mpg24_xxxxGetVideoFormat function. xxxx is Preview or Encoder.
E_SERIAL_READ	-8	Error occurs when read the serial number.
E_NON_STOP	-9	Cannot set video format while running.
E_DIRTECTX_INTERNAL	-10	Other error. Please call Mpg24_LastErrorInfo function to get the error message.
E_UNKNOWN_ERROR	-11	This is an unknown error.

4 ActiveX Control

4.1 PCI-MPG24 ActiveX Control Introduction

The PCI-MPG24 ActiveX Control (ocx) are based on API library and Directx 9.0. PCI_MPG24.dll and DirectX 9.0 need to be installed, and then the ActiveX Control can work.

ActiveX control is a reusable object what includes visual elements and codes. It can be used in many different container applications such as VB application, MS Office document, and web pages. PCI-MPG24 ActiveX control has two components in it, MPG24Preview and MPG24Encoder. They implement almost all ability of preview and encoder API functions.

4.2 Setting Up the Build Environment

All applications using the ActiveX control need the OCX file shown in the following table.

Library File	Description
PCI_MPG24.ocx	PCI-MPG24 ActiveX control. Required for all applications.

The file is located on [WINDOWS]\system32\.

If you are a VB user, please add this file into your VB project. The component name is 'ADLink PCI-MPG24 ActiveX Control'. After you add the control into your project, there are two controls,MPG24Preview and MPG24Encoder, on VB toolbox.

If you are a HTML user, add the following code to your HTML page:

```
<OBJECT ID="MPG24Preview"
CLASSID="CLSID:70B4054C-CC5B-435F-B0F4-
     A7A12F9238C2"
CODEBASE="PCI_MPG24.OCX#version=1,2,0,0">
</OBJECT>

<OBJECT ID="MPG24Encoder"
CLASSID="CLSID:476167C5-8310-4209-8039-
     E1E71A3C6971"
CODEBASE="PCI_MPG24.OCX#version=1,2,0,0">
```

</OBJECT>

4.3 Properties and Methods

Preview control

Property:

CardNo, Brightness, Contrast, Hue, Saturation, Sharpness, Gamma, ColorEnable, WhiteBalance, BacklightCompensation, VideoStandard, VideoSize, FrameRate, AutoShow, Status, DI0, DI1, DI2, DI3, DO0, DO1, DO2, DO3, Show, CustomWidth, CustomHeight

Method:

OpenDevice(), CloseDevice(), GetImageRange(Property, Max, Min, SteppingDelta, Default), Play(), PausePlay(), StopPlay(), SetCallback(CallbackProc), SaveImage(FileName), WriteEEPROM(Offset, Value), ReadEEPROM(Offset), ReadSerial(HighByte, LowByte)

Meaning

CardNo: A read/write integer value. The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

Brightness: A read/write long value. Default value is 750.

Contrast: A read/write long value. Default value is 100.

Hue: A read/write long value. Default value is 100.

Saturation: A read/write long value. Default value is 100.

Sharpness: A read/write long value. Default value is 50.

Gamma: A read/write long value. Default value is 140.

ColorEnable: A read/write boolean value. Default value is true.

WhiteBalance: A read/write long value. Default value is 0.

BacklightCompensation: A read/write boolean value. Default value is true.

VideoStandard: A read/write integer value. Default value is 0. The value can be:

- 0 – NTSC
- 1 – PAL

VideoSize: A read/write integer value. Default value is 1. The value can be:

- 0 – FULL D1
- 1 – CIF
- 2 – QCIF
- 3 – VGA
- 4 – QVGA
- 5 – Custom (The size depends on CustomWidth and CustomHeight)

Please refer to chapter 3.5 for the value of video size.

FrameRate: A read/write integer value. Default value is 0. The value can be:

- 0 – NTSC_30fps or PAL_25fps
- 1 – NTSC_15fps
- 2 – NTSC_10fps
- 3 – NTSC_5fps

AutoShow: A read/write boolean value. Default value is true. Specifies whether the video renderer automatically shows the video window when it receives video data.

Status: A read only integer value. The Value can be:

Bit 0-1: 0: stopped, 1: paused, 2:running

Bit 2: 0: video loss, 1: video present

DI0~DI3: Read only boolean values. A digital input.

DO0~DO3: Read only boolean values. Default value is false. A digital output.

Show: A read/write boolean value. Default value is true. Shows/Hides video window.

CustomWidth: A read/write integer value. Default value is 352. Sets the width of video window.

CustomHeight: A read/write integer value. Default value is 240. Sets the height of video window.

OpenDevice() as Long: Initializes the device and opens it for later use. Application need call this function before using any other methods. One device only can be opened once before you close it. This function returns an error code.

CloseDevice() as Long: Closes the opened device and releases the resources it occupied. This function returns an error code.

GetImageRange(Property As Long, ByRef Min As Long, ByRef Max As Long, ByRef SteppingDelta As Long, ByRef Default As Long) As Long: The GetImageRange function retrieves the range and default value of a specified video property such as brightness, contrast, hue, saturation, gamma, and sharpness. This function returns an error code.

Please refer to chapter 3.5 for the meaning of arguments.

Play() as Long: Starts to display the video window. This function returns an error code.

PausePlay() as Long: Pauses displaying the video window. This function returns an error code.

StopPlay() as Long: Stops displaying the video window. This function returns an error code.

SetCallback(CallbackProc As Long) As Long: Sets which procedure to be called when the frame data ready. User need write a callback procedure following the predefined format. This function returns an error code. The prototype of the callback function is CallbackProc(ByRef pBuffer as Byte, ByVal lBufferSize as Long).

pBuffer: The buffer of frame data.

lBufferSize: The size of frame data.

SaveImage(FileName As String) As Long: Captures one frame from the video samples and writes

it to a bitmap (.bmp) file. This function returns an error code.

ReadEEPROM(Offset As Byte) As Byte: This function return the offset address value of the on-board EEPROM.

WriteEEPROM(Offset As Byte, Value As Byte) As Long: Writes a value to offset address of the on-board EEPROM. This function returns an error code.

ReadSerial(ByRef HighByte As Long, ByRef LowByte As Long) As Long: This function reads the unique 48 bits serial number. Each PCI-MPG24 has a ROM what stores a unique serial number. The function stores the number at the two variables, HighByte and LowByte, and returns an error code. HighByte is the upper 16-bit of serial number. LowByte is the lower 32-bit of serial number.

Please refer to 3.10 for the meaning of error code.

Encoder Control

Property:

CardNo, PortNo, Brightness, Contrast, Hue, Saturation, Sharpness, VideoStandard, VideoSize, FrameRate, TargetBitrate, MPEG4Mode, FileName, Status

Method:

OpenDevice(), CloseDevice(), GetImageRange(Property, Max, Min, SteppingDelta, Default), Play(), StopPlay(), SetCallback(CallbackProc)

Meaning

CardNo: A read/write integer value. The card ID which value is between 0 and 7. The card ID can be got from S1 dip switch. Please refer to Hardware Reference chapter.

PortNo: The video input channel which value is between 0 and 3.

Brightness: A read/write long value. Default value is 32.

Contrast: A read/write long value. Default value is 57.

Hue: A read/write long value. Default value is 0.

Saturation: A read/write long value. Default value is 50.

Sharpness: A read/write long value. Default value is 14.

VideoStandard: A read/write integer value. Default value is 0. The value can be:

- 0 – NTSC
- 1 – PAL

VideoSize: A read/write integer value. Default value is 3. The value can be:

- 0 – FULL D1
- 1 – CIF
- 2 – QCIF (Not support)
- 3 – VGA
- 4 – QVGA
- 5 – Custom (Not support)

Please refer to chapter 3.5 for the value of video size.

FrameRate: A read/write integer value. Default value is 0. The value can be:

- 0 – NTSC_30fps or PAL_25fps
- 1 – NTSC_15fps
- 2 – NTSC_10fps
- 3 – NTSC_5fps

TargetBitrate: A read/write integer value. Default value is 0. The value can be:

- 0 – Bitrate_4M
- 1 – Bitrate_2M
- 2 – Bitrate_1500K
- 3 – Bitrate_1M
- 4 – Bitrate_750K
- 5 – Bitrate_500K
- 6 – Bitrate_384K

MPEG4Mode: A read/write integer value. Default value is 0. The value can be:

- 0 – DivX-Codec
- 1 – Microsoft_Codec

FileName: A read only integer value. sets the name of the file into which media samples will be written.

Status: A read only integer value. The Value can be:

Bit 0-1: 0: stopped, 1: paused, 2:running

Bit 2: 0: video loss, 1: video present

OpenDevice() as Long: Initializes the device and opens it for later use. Application need call this function before using any other methods. One device only can be opened once before you close it. This function returns an error code.

CloseDevice() as Long: Closes the opened device and releases the resources it occupied. This function returns an error code.

GetImageRange(Property As Long, ByRef Min As Long, ByRef Max As Long, ByRef SteppingDelta As Long, ByRef Default As Long) As Long: The GetImageRange function retrieves the range and default value of a specified video property such as brightness, contrast, hue, saturation, gamma, and sharpness. This function returns an error code.

Please refer to chapter 3.5 for the meaning of arguments.

Play() as Long: Starts to display the video window. This function returns an error code.

StopPlay() as Long: Stops displaying the video window. This function returns an error code.

SetCallback(CallbackProc As Long) As Long: Sets which procedure to be called when the frame data ready. User need write a callback procedure following the predefined format. This function returns an error code.

The prototype of the callback function is CallbackProc(ByRef pBuffer as Byte, ByVal lBufferSize as Long).

pBuffer: The buffer of frame data.

```
lBufferSize: The size of frame data.
```

Please refer to 3.10 for the meaning of error codes.

5 Linux Programming Guide

5.1 Overview

PCI-MPG24 Linux SDK is a LINUX 2.6.9 based video capture and MPEG encoder development package. It includes different levels of APIs and objects and developers can use different interface to meet user's requirements.

The PCI-MPG24 software includes the following different level interfaces:

Preview:

- Video4Linux: Providing a SDK for video capture/overlay. Please refer to the document of Video for Linux API. You can download this manual from http://linux.bytesex.org/v4l2/
- V4lsample: An application developed on and distributed with LINUX SDK. This application is developed based on Video for Linux Two API.

Encoder:

- CVideoCapture: Providing an interface separating server from encoder driver. This interface enables an application to invoke I/O actions on an instance of a videocapture through an I/O packet.
- Go-server: An application developed on and distributed with LINUX SDK. This application is developed based on CServer. It can be discarded or customized if users want to develop their own application.

5.2 Encoder Linux SDK Architecture

This chapter describes the architecture of Encoder Linux SDK. First, we introduce the application based on Encoder Linux SDK; then describe the Data Flow Diagram (DFD) of Encoder Linux SDK.

Application on Encoder LINUX SDK

Application is a user application implemented by customer.

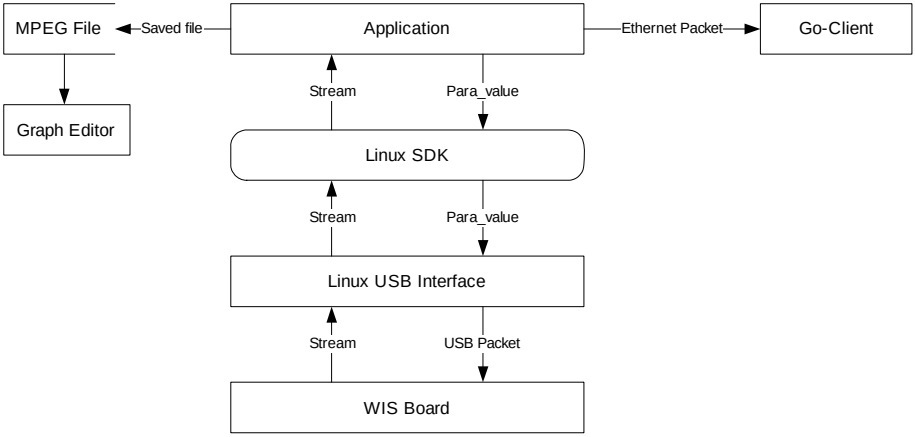


Figure 5-1: Application Based on LinuxSDK

DFD of Encoder LINUX SDK

The following graph describes the DFD of Encoder Linux SDK. User can use different level APIs to implement their own application according to their requirement.

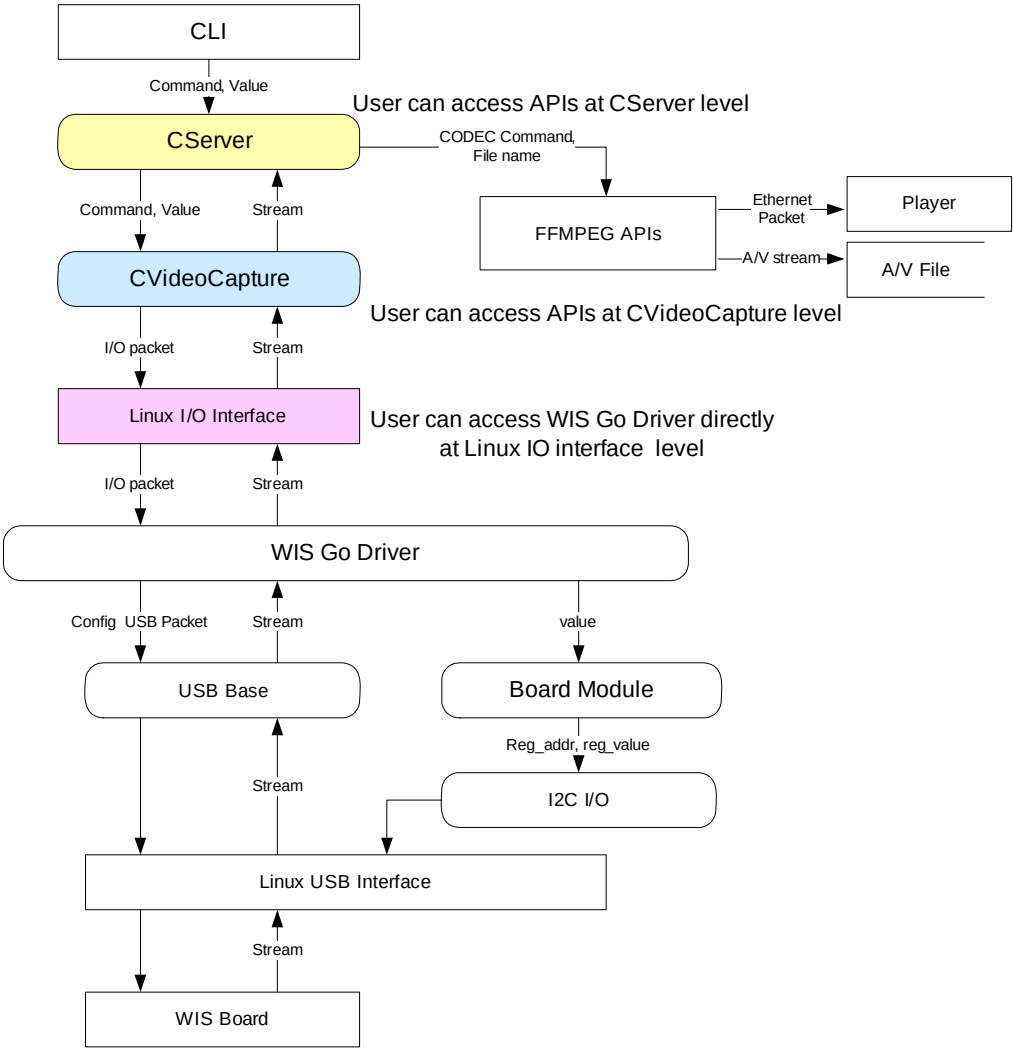


Figure 5-2: DFD of Encoder Linux SDK

5.3 Encoder LINUX SDK Interfaces

This chapter describes the class or APIs.

CVideoCapture Interface

Purpose

This class provides an interface separating server from encoder driver. This interface enables an application to invoke I/O actions on an instance of a videocapture through an I/O packet.

Method Summary

Method	Description
Initialization Operation Method	
CreateInstance	Create an instance for this interface.
Release	Release the instance for this interface.
DeviceExist	Specify if a device is present
GetBoardInfo	Get board information
Video Operation Method	
SetVideoConfig	Set the video config. The setting will not take effect until next call of StartCapturing.
StartCapturing	start capturing video from chip using the pre-set stream and bitrate setting
StopCapturing	Stop capturing
GetOneFrame	Get one video frame from chip, this function works in block mode, it will not return until a frame is unavailable or StopCapturing is called.
SetVideoSource	Set video input source.
GetVideoSource	Get video input source.
SetTVStandard	Specify TV standard.
ChangePFrameRate	Change frame rate only for P frame.
SetFPS	Set frame rate for I, P and B frame.
BitrateControl	Specify bitrate of the output video.
SetResolution	Specify the output video stream resolution.
ChangeIFrameQuantizer	Change quantized scale only for I frame.
ChangePFrameQuantizer	Change quantized scale only for P frame.
ChangeBrightness	Specify brightness value of output video.
ChangeContrast	Specify contrast value of output video.

ChangeHue	Specify hue value of output video.
ChangeSaturation	Specify saturation value of output video.
ForceIFrame	Insert an Iframe.
InsertNewSequenceHeader	Insert a new sequence header.
sigDetect	Monitor signal lost
SetMDRegions	Set motion detection region.
SetMDThresholdsAndSensitivities	Set threshold and sensitivity value for motion detection
ResetMotionDetection	Reset motion detection.
InitMotionDetection	Initialize motion detection
osd_show	Show osd
Debugging Operation Method	
ReadCBusReg	Read from c-bus registers for debugging.
WriteCBusReg	Write to c-bus registers for debugging.
ReadCBus	Read from c-bus register
WriteCBus	Write data to c-bus register
I2C_WriteRegister	Write peripheral chip registers via I2C bus
GPIO_Read	Read the status of GPIO pins.

Initialization Methods

CreateInstance

Synopsis:

STATIC SINT32 CreateInstance (VOID **pp, SINT8 instance)

Description:

This function creates an instance for CVideoCapture interface.

Arguments:

VOID **pp – [out] Pointer to this new instance. It should be allocated memory by the caller.

SINT8 instance – [in] Index number of the connected device. The range should be from 0 to the return value of GetDeviceNumber () – 1.

Returns:

SINT32 – Status of whether the new instance is created successfully.

SUCCESS (0x0000) – success;

ERR_MEMORY (0x0002) – no enough memory to create a new instance;

ERR_DEVICE (0x0001) – device file can not be opened.

Notes:

The argument, pp, is checked to ensure that it is not NULL. If yes, ERR_MEMORY will be returned.

Release

Synopsis:

VOID Release ()

Description:

This function releases the instance of CVideoCapture interface. And it also frees its allocated memory.

Arguments:

None

Returns:

None

DeviceExist

Synopsis:

SINT32 DeviceExist ()

Description:

This function specifies if a device is present.

Arguments:

None

Returns:

SINT32 – Status of whether the device exists.

0 – device not exists

1 – device exists

GetBoardInfo

Synopsis:

void GetBoardInfo (REVISION_INFO *p_bi)

Description:

This function gets the encoder information.

Arguments:

REVISION_INFO *p_bi – [out] A pointer to board information, caller is responsible to allocate memory for it.

Returns:

None

Video Operation Methods

SetVideoConfig

Synopsis:

void SetVideoConfig (TCFGVIDEO *pcfgvideo)

Description:

This function sets the video configuration but not take effect immediately until next call of StartCapturing ().

Arguments:

TCFGVIDEO *pcfgvideo – [in] A pointer to an instance of structure TCFGVIDEO. The caller should allocate memory for it.

Returns:

None

Notes:

This setting will not take effect until next call of StartCapturing.

StartCapturing

Synopsis:

SINT32 StartCapturing ()

Description:

This function starts capturing video from chip using the pre-set stream and bit rate setting.

Arguments:

None

Returns:

SINT32 – status of whether video capture starts successfully

SUCCESS (0) – success

ERR_DEVICE (0x0001) – device initialization failed.

Notes:

The function configures audio and video with pre-set value. The device will be put into DS_RUNNING status if start successfully.

StopCapturing

Synopsis:

SINT32 StopCapturing ()

Description:

This function stops capturing.

Arguments:

None

Returns:

SINT32 – status of if capture stops successfully

SUCCESS (0) – device stops successfully

ERR_DEVICE (0x0001) – chip is not in running state

GetOneFrame

Synopsis:

SINT32 GetOneFrame (UINT8 *pBuf, SINT32 BufLen, TFrameInfo *pFI)

Description:

This function gets one video frame from chip and stores into pBuf.

Arguments:

UINT8 *pBuf – [out] Buffer for receiving the video stream, caller is responsible for allocating a buffer big enough.

SINT32 BufLen – [in] Buffer length, caller is responsible for setting a valid buffer length.

TFrameInfo *pFI – [out] A pointer to TFrameInfo structure, it will save the returned frame information; caller is responsible for allocating memory for it.

Returns:

SINT32 – status of if get one frame successfully

SUCCESS (0) – get one frame from chip successfully

ERR_DEVICE (0x0001) –device not open or chip is not in running state ERR_NOMOREFRAME (0x1000) – no more frame is available in chip.

Notes:

This function works in block mode, it will not return until a frame is unavailable or StopCapturing () is called. If return ERR_NOMOREFRAME, StopCapturing () is called.

SetVideoSource

Synopsis:

SINT32 SetVideoSource (SINT32 source)

Description:

This function set the input video source.

Arguments:

SINT32 source – [in] input video source: 1 for Composite and 0 for S-Video

Returns:

SINT32 – status of video source setting:

0 – success

-1 – error input source

Notes:

This function will check the validity of input source. The input source is one of SVideo and Composite; if invalid it does nothing.

GetVideoSource

Synopsis:

SINT32 GetVideoSource (void)

Description:

This function gets the current video input source.

Arguments:

None

Returns:

SINT32 – the input video source retrieved from board module.

1 – Composite

0 – SVideo

-3 – Unknown

SetTVStandard

Synopsis:

SINT32 SetTVStandard (UINT32 mask)

Description:

This function specifies TV standard.

Arguments:

UINT32 mask – [in] the TV standard: PAL and NTSC.

Returns:

SINT32 – status of whether TV standard set successfully

0 – set TV standard successfully

-1 – no match configuration or argument error

-3 – error of unknown register

ChangePFrameRate

Synopsis:

SINT32 ChangePFrameRate (UINT16 rate)

Description:

This function changes P frame rate.

Arguments:

UINT16 rate – [in] the frame rate value.

Returns:

SINT32 –status of this function.

0 – P Frame rate set successfully

1 – write interrupt of ON CHIP Mode is timeout

100 – MAXUSBPOLLING, write interrupt error

SetFPS

Synopsis:

SINT32 SetFPS (SINT32 fps)

Description:

This function specifies frame rate for output video.

Arguments:

SINT32 fps – [in] frame rate, the range of this value depend on the TV standard: NTSC: 30, 15, 10, 7, 6, 5, 4, 3, 2, 1;

PAL: 25, 12.5, 6.25, 5, 4, 3, 2, 1

Returns:

SINT32 – return the frame rate value just setting.

Notes:

This function changes frame rate in effect by quickly reseting and restarting the encoder.

BitrateControl

Synopsis:

SINT32 BitrateControl (SINT32 change_direction,

SINT32 new_i_qscale,

SINT32 new_p_qscale,

SINT32 new_target_bitrate,

SINT32 new_peak_rate,

SINT32 new_vbv_buffer,

SINT32 new_converge_speed,

SINT32 new_lambda)

Description:

This function controls bit rate.

Arguments:

SINT32 change_direction – [in] the data direction when switching encode mode. This value is one of:

0: switching from VBR to VBR;

1: switching from CBR to VBR;

2: switching from VBR to CBR;

3: switching from CBR to CBR.

SINT32 new_i_qscale – [in] The quantized scale for I-frames. The range of this value is from 2 to 31.

SINT32 new_p_qscale – [in] The quantized scale for P-frames. The range of this value is from 2 to 31.

SINT32 new_target_bitrate – [in] The target bite rate of the encoded stream. The range of this value is from 0 to 0x1000000.

SINT32 new_peak_rate – [in] The peak bite rate in the encoded stream. The range of this value is from 0 to 0x4000000.

SINT32 new_vbv_buffer – [in] The vbv buffer size. The range of this value is from 0 to 112*16*1024(1835008)

SINT32 new_converge_speed – [in] The converge speed for CBR. The range of this value is from 0 to 100.

SINT32 new_lambda – [in] The lambda for CBR. The range of this value is from 0 to 100.

Returns:

SINT32 – status of if bit rate set successfully.

0 – change bit rate successfully and update the TCFGBRCTRL structure.

1 – write interrupt time out

-1 – change_direction parameter error

-2 – new_i_qscale parameter out of range

-3 – new_p_qscale parameter out of range

-4 – new_target_bitrate parameter out of range

-5 – new_peak_rate parameter out of range

-6 – new_converge_speed parameter out of range

-7 – new_lambda parameter out of range

-8 – new_vbv_buffer parameter out of range

Notes:

new_target_bitrate parameter is the desired average target bit rate of the encoded stream, in bits per second (bps):

0: If Q>0, apply variable bit rate control (VBR) using the value of Q. If Q=0, no bit rate control algorithm is applied. Bit rate will be determined by values of IQ, PQ, BQ provided by user.

>0: Apply constant bit rate control (CBR) using the value of target_bitrate.

new_peak_rate parameter is the highest bit rate allowed in the encoded stream, in bits per second (bps). This parameter is only valid when applying constant bit rate control. The larger this value is, the higher the peak bit rate is.

new_vbv_buffer parameter is a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

For new_converge_speed parameter, the larger value means faster converging speed.

new_lambda parameter is the factor determining stream quality. The larger the value is, the smoother the stream will be, but the quality of each frame will decrease. The smaller the value is, the better each picture quality in the stream will be, but the entire video stream will look jumpy due to frame drop.

SetResolution

Synopsis:

void SetResolution (TCFGVIDEO *pcfgvideo)

Description:

This function specifies the output video stream resolution mode.

Arguments:

TCFGVIDEO *pcfgvideo – [in] A pointer to structure TCFGVIDEO presenting one of the following mode: D1, D.5, 4SIF, 2SIF, SIF, CIF-N, CIF-P, QCIF.

Caller is responsible to allocate memory for it.

Returns:

None

Notes:

This function changes resolution parameters by quickly reseting and restarting the encoder.

ChangeIFrameQuantizer

Synopsis:

SINT32 ChangeIFrameQuantizer (UINT16 iqscale)

Description:

This function changes quantized scale for I frame.

Arguments:

UINT16 iqscale – [in] the quantized scale value, the range of value is from 2 to 30.

Returns:

SINT32 –status of this function:

0 – I Frame quantizer set successfully or board information can not be identified

1 – write interrupt of ON CHIP Mode is timeout

100 – MAXUSBPOLLING, write interrupt error

Notes:

Note that higher the iqscale value, lower the image quality and lower the bit rate.

ChangePFrameQuantizer

Synopsis:

SINT32 ChangePFrameQuantizer (UINT16 pqscale)

Description:

This function changes quantized scale for P frame.

Arguments:

UINT16 iqscale – [in] the quantized scale value, the range of value is from 2 to 30.

Returns:

SINT32 –status of this function:

0 – P Frame quantizer set successfully or board information can not be identified

1 – write interrupt of ON CHIP Mode is timeout

100 – MAXUSBPOLLING, write interrupt error

Notes:

Note that higher the iqscale value, lower the image quality and lower the bit rate.

ChangeBrightness

Synopsis:

SINT32 ChangeBrightness (SINT32 brightness_value)

Description:

This function configures video brightness.

Arguments:

SINT32 brightness_value – [in] brightness value. The range of this value is from 0 to 100. Greater value means brighter.

Returns:

SINT32 – return the input brightness value.

Notes:

This function validates input value and if it is invalid, function does nothing.

ChangeContrast

Synopsis:

SINT32 ChangeContrast (SINT32 contrast_value)

Description:

This function configures video contrast.

Arguments:

SINT32 contrast_value – [in] contrast value, the range of this value is from 0 to 100. Greater value means higher the contrast.

Returns:

SINT32 – return the input contrast value.

Notes:

This function validates input and if it is invalid, it does nothing.

ChangeHue

Synopsis:

SINT32 ChangeHue (SINT32 hue_value)

Description:

This function configures video hue.

Arguments:

SINT32 hue_value – [in] hue value, the range of this value is from -50 to 50. Greater value means higher hue.

Returns:

SINT32 – return the input hue value.

Notes:

This function validates input and if it is invalid, function does nothing.

ChangeSaturation

Synopsis:

SINT32 ChangeSaturation (SINT32 saturation_value)

Description:

This function configures video saturation.

Arguments:

SITN32 saturation_value – [in] saturation value. The range of this value is from 0 to 100. Greater value means higher saturation.

Returns:

SINT32 – return the input saturation value.

Notes:

This function validalidates input and if it is invalid, function does nothing.

ForceIFrame

Synopsis:

SINT32 ForceIFrame (void)

Description:

This function inserts an Iframe into the encoded video stream.

Arguments:

None.

Returns:

SINT32 – status of this function:

0 – force Iframe successfully or board information can not be identified

1 – write interrupt of ON CHIP Mode time out

100 – MAXUSBPOLLING, write interrupt error

InsertNewSequenceHeader

Synopsis:

SINT32 InsertNewSequenceHeader (void)

Description:

This function inserts a new sequence header at next GOP.

Arguments:

None

Returns:

SINT32 –status of this function:

0 – force a new sequence head successfully or board information can not be identified

1 – write interrupt of ON CHIP Mode is timeout

100 – MAXUSBPOLLING, write interrupt error

sigDetect

Synopsis:

SINT32 sigDetect (void)

Description:

This function checks if the input signal is lost.

Arguments:

None

Returns:

SINT32 – status of the input signal

0 – signal lost is detected

1 – no signal lost is detected

-3 – I2C read fails

SetMDRegions

Synopsis:

SINT32 SetMDRegions (UINT8 *arru8MotionCoordinates,

UINT32 u32MaxXCoord,

UINT32 u32MaxYCoord)

Description:

This function set the region numbers for every macro block to watch for motion.

Arguments:

UINT8 *arru8MotionCoordinates – [in] Two dimensional array of region numbers where each [x][y] location contains the region number for that macro block. This argument MUST be exactly the same number of macro blocks as the current picture. If there is a size mismatch the results are undetermined.

UINT32 u32MaxXCoord – [in] The maximum X coordinate for the picture size. This is really just used for error checking.

UINT32 u32MaxYCoord – [in] The maximum Y coordinate for the picture size. This is really just used for error checking.

Returns:

SINT32 – whether the macro block map successfully

0 – SUCCESS, the macro block map has been set in the device.

1 – ERR_DEVICE, the macro block map had invalid regions or there was a mismatch in the picture size.

Notes:

Be careful that this array is the same size as the picture (in macro blocks) and that the caller has initialized every location in this array to ensure that the map is correctly set.

SetMDThresholdsAndSensitivities

Synopsis:

SINT32 SetMDThresholdsAndSensitivities (UINT16 arrs16MotionThresholds [MAX_REGIONS_OF_INTEREST][NUM_MOTION_TYPES],

UINT8 *arru8MotionSensitivity)

Description:

Set the motion vector and SAD (Sum of Absolute Differences) thresholds for each region. Also set the sensitivity thresholds for each region.

Arguments:

UINT16 arrs16MotionThresholds – [in] two dimensional array of thresholds of thresholds for the device.

UINT8 *arru8MotionSensitivity – [in] one dimensional array of sensitivities.

Returns:

SINT32 – status of if thresholds have been set successfully:

0 – SUCCESS, the macro block thresholds have been set in the device. 1 – ERR_DEVICE, the macro block thresholds are invalid.

Notes:

This function validates threshold input and returns ERR_DEVICE for invalid.

ResetMotionDetection

Synopsis:

SINT32 ResetMotionDetection (void)

Description:

This function reset the state array of motion detection.

Arguments:

None

Returns:

SINT32 – return 0 always

InitMotionDetection

Synopsis:

SINT32 InitMotionDetection (void)

Description:

This function is to initialize and enable the motion detector. It is to be called before the initial package is generated as it only affects the initial package.

Arguments:

None

Returns:

SINT32 – return 0 always.

Notes:

Call this before Pacgen. If you don't call this, you can't use motion detection.

OSD_show

Synopsis:

SINT32 osd_show(unsigned char x,unsigned char y, void *String)

Description:

This function shows osd information with the IO_OSD_DISPLAY IO command.

Arguments:

unsigned char x – [in] type of osd operation:

0 – clean OSD display with writing the OSD frame EOF to the start address of current frame buffer;

1 – prepare OSD data from user to OSD buffer with OSD string format.

2 – write the prepared OSD data from the OSD buffer to firmware and display on screen.

Unsigned char y – [in] the string number of OSD data; User can input multi string to OSD buffer with the total length of OSD_STRING_LEN_MAX-3.

Void *string – [in] in order to show osd correctly, user should pass a pointer to osd_string_t struct, define in Include/osd.h.

x: the x coordinate of the location to show osd; in the unit of MB_SIZE (16 pixels)

y: the y coordinate of the location to show osd in the unit of MB_SIZE (16 pixels);

osd_string[]: the content of string text. The max lenght is OSD_STRING_LEN_MAX -3

Returns:

SINT32 – return 0 always.

Debugging Operation Methods

ReadCBusReg

Synopsis:

int ReadCBusReg (SINT32 RegNum, UINT16 Addr[], UINT16 Data[ ])

Description:

This function reads several encoder chip registers through firmware for debugging.

Arguments:

SINT32 RegNum – [in] Register number to identify the buffer length for register address and register data.

UINT16 Addr[ ] – [in] Array of address where read registers, caller is responsible for allocating buffer big enough.

UINT16 Data[ ] – [out] Array of data read from register, caller is responsible for allocating buffer big enough.

Returns:

int – status of if read register successfully.

0 – success

1 – download buffer error

2 – check interrupt pipe error

Notes:

Address and data are all in 16 bits.

WriteCBusReg

Synopsis:

int WriteCBusReg (SINT32 RegNum, UINT16 AddrData[])

Description:

This function writes data to a register on the C-bus for debugging.

Arguments:

SINT32 RegNum – [in] Register number to identify buffer length for register address.

UINT16 AddrData[ ] – [in] Array of address and data pair, caller is responsible for allocating buffer that is big enough.

Returns:

int – status of if write register successfully.

0 – write data to cbus register successfully

1 – write data operation time out

100 – write data interrupt error

Notes:

Address and data are all in 16 bits.

ReadCBus

Synopsis:

SINT32 ReadCBus (UINT16 addr, UINT16* data)

Description:

This function reads a register on the C-bus.

Arguments:

UINT16 addr – [in] register address of read from

UINT16 *data – [out] data read from register

Returns:

SINT32 – status of if read register successfully.

0 – success

1 – download buffer error

2 – check interrupt pipe error

WriteCBus

Synopsis:

SINT32 WriteCBus (UINT16 addr, UINT16 data)

Description:

This function writes data to c-bus.

Arguments:

UINT16 addr – [in] register address to write to

UINT16 data – [in] data to write to a register

Returns:

SINT32 – status of if write to C-bus register

0 – write data to cbus successfully

1 – write data operation time out

100 – write data interrupt error

I2C_WriteRegister

Synopsis:

int I2C_WriteRegister (UINT16 Addr, UINT8 Data)

Description:

This function writes I2C registers for programming/debugging purpose.

Arguments:

UINT16 Addr – [in] I2c register address

UINT8 Data – [in] data to write to I2C register

Returns:

int – status of I2C write register operation.

0 – write I2C register successfully

1 – write I2C register interrupt time out or read CBus error

2 – write I2C control register error

3 – write I2C loaddr register error

4 – write I2C data register error

5 – write i2c_devaddr_upaddr_reg register error

100 – write I2C register interrupt error

GPIO_Read

Synopsis:

int GPIO_Read(SINT32 Index, SINT32 *pValue)

Description:

This function read a signal on a GPIO pins.

Arguments:

SINT32 Index – [in] Index of GPIO pins, should be between 0 and 4. Pin0 and Pin1 are used for channel ID; Pin2, Pin3, and Pin4 are used for Card ID.

SINT32 *pValue – [out] The signal of GPIO pin.

Returns:

int – status of GPIO read operation.

0 – read GPIO successfully

ERR_DEVICE (0x0001) – device operation failed.

ERR_PARAMETER(0x0007) – bad parameter of Index.

5.4 WIS-LIVE Interface

WIS-LIVE is an interface to convert the encoded A/V stream to the LIVE streaming libraries which is an OpenSource library.

WIS-LIVE provides the following two libraries:

Wis-live is a library containint the APIs for go-server application.

Wis-live_stand is a library containing the APIs for live-server application.

WIS-LIVE APIs

PlayMPEG4AudioVideoStream

Synopsis:

```
void PlayMPEG4AudioVideoStream(StreamBufQueue_t *
     pABuf,
StreamBufQueue_t* pVBuf,
char *pIpAddr,
int Aport,
int Vport,
int RTSPport,
int audioIsPCM)
```

Description:

This function can be used to play MPEG4 video and mp2 audio with Live.com library.

Arguments:

StreamBufQueue_t *pABuf – [in] Pointer to audio buffer queue

StreamBufQueue_t *pVBuf – [in] Pointer to video buffer queue

char *pIpAddr – [in] destination ip address, NULL means use Unicast

int Aport – [in] audio destination port

int Vport – [in] video destination port

int RTSPport – [in] rtsp listen port

int audioIsPCM – [in] whether the input audio stream is raw PCM rather than MPEG audio

Returns:

None;

PlayMPEG1or2AudioVideoStream

Synopsis:

```
void
     PlayMPEG1or2AudioVideoStream(StreamBufQueue
     _t * pABuf,
StreamBufQueue_t * pVBuf,
char *pIpAddr,
int Aport,
int Vport,
int RTSPport,
int audioIsPCM)
```

Description:

This function is used to play MPEG1 and MPEG2 AV with Live.com library.

Arguments:

StreamBufQueue_t *pABuf – [in] Pointer to audio buffer queue

StreamBufQueue_t *pVBuf – [in] Pointer to video buffer queue

char *pIpAddr – [in] destination ip address, NULL means use Unicast

int Aport – [in] audio destination port

int Vport – [in] video destination port

int RTSPport – [in] rtsp listen port

int audioIsPCM – [in] whether the input audio stream is raw PCM rather than MPEG audio

Returns:

None;

WIS-LIVE-STAND APIs

PlayAudioVideoStream

Synopsis:

```
void PlayAudioVideoStream( FrameReadFunc* aRead,
FrameReadFunc* vRead,
char *pIpAddr,
int Aport,
int Vport,
int RTSPport,
char* RTSPaddr,
int audioNumChannels,
int audioSamplingRate,
int audioOutputMode,
int videoOutputMode,
char const* allowedUsername,
char const* allowedPassword)
```

Description:

This function is used to play MPEG4 video and mp2 audio with Live.com library.

Arguments:

ReadFrameFunc *aRead – [in] functions that read audio/video frames

ReadFrameFunc *vRead – [in] functions that read audio/video frames

char *pIpAddr – [in] destination ip address, NULL means use Unicast

int Aport – [in] audio destination port

int Vport – [in] video destination port

int RTSPport – [in] rtsp listen port

char* RTSPaddr – [in] IP address to use for RTSP server (NULL for default)

int audioNumChannels – [in] number of channels of PCM audio

int audioSamplingRate – [in] sampling frequency of audio (if PCM only)

int audioOutputMode – [in] the mode of audio output.

0: => 16-bit PCM;

1: => 8-bit u-law;

any other value (x) => x kbps MPEG-1 audio

int videoOutputMode – [in] the mode of video output

0: => MPEG-4;

1: => H.263

char const* allowedUsername – [in] user name to authorize access to the server;

NULL to allow anyone access

char const* allowedPassword – [in] password to authorize access to the server;

NULL to allow anyone access

Returns:

None;

WIS-LIVE Structure

StreamBufQueue_t

```
typedef struct StreamBufQueue_s
{
struct stream_buf_s stream_buf[MAX_NUM_BUFFER];/
     /stream buffer queue
int head;// queue header, value is between 0 -
     MAX_NUM_BUFFER
int tail;// queue tail, value is between 0 -
     MAX_NUM_BUFFER
int type;// 1: audio stream buffer queue, 2:
     video stream buffer queue
pthread_mutex_t mutex;//pthread mutex
} StreamBufQueue_t ;
```

Stream_buf_s

```
struct stream_buf_s
{
char *p; //buffer pointer
char *cur;//Current position
int len;  //the current string length
struct timeval creationTime; // when the data was
     created
};

#define MAX_NUM_BUFFER 128
```

5.5 OSD APIs

Purpose

This section describes how to use OSD APIs.

API Description

init_osd

Synopsis:

osd_error_t init_osd (void *pdxc, long video_frame_width, long video_frame_height)

Description:

This function initializes the encoder chip to handle OSD command.

Arguments:

void *pdxc – [in] pointer to DEVICE_EXTENSION_COMMON struct. User is responsible for its allocation and initialization.

long video_frame_widht – [in] width of input video frame. Unit is number of pixels.

long video_rame_height – [in] height of input video frame. Unit is number of pixels.

Returns:

osd_error_t: return the status of OSD initialization. osd_error_t is a enumeration type defined in src/drv/osd.h

OSD_ERROR_OUTOFMEMORY – system has not enough memory for osd.

OSD_ERROR_OK – initialization succeed.

Notes:

This function is called when initializing device.

get_osd_frame

Synopsis:

osd_error_t get_osd_frame (unsigned char num, void *string_ptr)

Description:

This function actually transfers to be displayed string from user program to the OSD internal buffer.

Arguments:

unsigned char num – [in] number of osd string. User shall specify a set of X, Y coordinates and data strings in the following structure passed by *string_ptr. This is number specifies number of valid strings in the structure.

void *string_ptr – [in] in order to show osd correctly, user should pass a pointer to osd_string_t struct defined in Include/osd.h:

The struct of osd_string_t contains the following:

unsigned char x: the x coordinate of the location to show osd; unit is number of macro blocks (a macro block is 16*16 pixels)

unsigned char y: the y coordinate of the location to show osd; unit is number of macro blocks (a macro block is 16*16 pixels).

Example: Preview window is 720*480, max value of x is 720/16-1=44, max value of y is 480/16-1 = 29.

char osd_string[]: the content of string text. The max lenght is OSD_STRING_LEN_MAX -3

Returns:

osd_error_t – return OSD_ERROR_OK, get OSD frame successfully.

Notes:

The function name should be named as send_osd_data(void *string_ptr).

show_osd_frame

Synopsis:

osd_error_t show_osd_frame (osd_display_mode_t mode)

Description:

This function writes the prepared OSD data to OSD font index buffer for chip to display.

Arguments:

osd_display_mode_t mode – [in] the mode to update current buffer of write to a new buffer.

OSD_NORMAL: 0

OSD_REFRESH: 1

Returns:

osd_error_t – return the status of OSD show.

OSD_ERROR_BUS_WRITE_FAIL – write CBus failed.

OSD_ERROR_OK – succeed.

Notes:

clean_osd

Synopsis:

osd_error_t clean_osd ()

Description:

When user want to clear the strings displayed by osd, just call this function.

Arguments:

None

Returns:

osd_error_t – return the status of OSD clean.

OSD_ERROR_OK – clean OSD successfully.

Notes:

Example:

Here is an example to show how to use OSD APIs.

1. initialize OSD

```
In the function initializeDe-
     vice(PDEVICE_EXTENSION_COMMON context),
     call function init_osd() to initialize OSD.
init_osd(context, video_config->rescfg.width,
     video_config->rescfg.height);
```

2. send string to OSD module

```
sd_string_t osd_str[2] = {
{1, 6, "How are you today?"},
{4, 9, " Wish you a good day"}
};
get_osd_frame(2, &osd_str[0]);
```

3. show OSD

```
show_osd_frame(0);
```

It will show the string “How are you today?” at (1,6) macro block.

“Wish you a good day” at (4,9) macro block.

4. clear OSD

```
clean_osd();
```

5.6 FFMPEG APIs

FFMPEG is a very fast video and audio converter. It can also grab from a live audio/video source. The command line interface is designed to be intuitive, in the sense that FFMPEG tries to figure out all the parameters, when possible. You have usually to give only the target bit rate you want.

FFMPEG can also convert from any sample rate to any other, and resize video on the fly with a high quality poly-phase filter.

FFMPEG provides the following two libraries:

Libavcodec is the library containing the codecs (both encoding and decoding). See `libavcodec/apiexample.c' to see how to use it.

Libavformat is the library containing the file formats handling (mux and demux code for several formats). See `ffplay.c' to use it in a player. See `output_example.c' to use it to generate audio or video streams.

You can integrate all the source code of the libraries to link them statically to avoid any version problem. All you need is to provide a 'config.mak' and a 'config.h' in the parent directory. See the defines generated by ./configure to understand what is needed.

Some of FFMPEG APIs are called by methods of CServer. Main FFMPEG APIs will be described in this section. User can customize its own CServer for its own functionality using FFMPEG APIs.

AV CODEC APIs

register_avcodec

Synopsis:

void register_avcodec(AVCodec *format)

Description:

This function is used for registering CODEC; the specified CODEC will be linked to the CODEC list first_avcodec;

Arguments:

AVCodec *format – [in] Points to the specified CODEC to be registered;

Returns:

None;

avcodec_find_encoder

Synopsis:

AVCodec *avcodec_find_encoder(enum CodecID id)

Description:

This function is used for finding the specified encoder in the CODEC list first_avcodec of FFMPEG.

Arguments:

enum CodecID id – [in] The CODEC id of the encoder to be found;

Returns:

AVCodec * - null, not found; Or the pointer to the struct AVCode for the encoder found.

avcodec_find_decoder

Synopsis:

AVCodec *avcodec_find_decoder(enum CodecID id)

Description:

This function is used for finding the decoder according to the specified CODEC id in the CODEC list first_avcodec of FFMPEG.

Arguments:

enum CodecID – The CODEC id of the decoder to be found;

Returns:

AVCodec * - null, not found; Otherwise, the pointer to AVCodec of the CODEC found;

avcodec_open

Synopsis:

int avcodec_open(AVCodecContext *avctx, AVCodec *codec)

Description:

This function opens the specified CODEC in order to code or encode.

Arguments:

AVCodecContext *avctx – Points to the context of the CODEC.

AVCodec *codec – Points to the CODEC used for coding or encoding.

Returns:

int – status of open:

-1 – the codec has been open

0 – open successfully

12 – not big enough memory to allocate

<0 – the codec initialized failed

avcodec_close

Synopsis:

int avcodec_close(AVCodecContext *avctx)

Description:

This function is used for closing the specified CODEC not to be used for coding or encoding any more;

Arguments:

AVCodecContext *avctx – Points to the context of the CODEC;

Returns:

int – 0;

avcodec_encode_audio

Synopsis:

int avcodec_encode_audio(AVCodecContext *avctx,

uint8_t *buf,

int buf_size,

const short *samples)

Description:

This function is used for encoding an audio stream.

Arguments:

AVCodecContext *avctx – [in] Points to the CODEC context used for encoding audio stream.

uint8_t *buf – [out] Points to the buffer for storing the returned encoded audio stream.

int buf_size – [in] The buffer size of the buffer pointed to by buf.

const short *samples – [in] Points to the buffer storing the audio stream.

Returns:

int – status of audio encoding:

-1, error;

Others, the number of bytes of returned encoded audio stream;

avcodec_decode_audio

Synopsis:

int avcodec_decode_audio(AVCodecContext *avctx,

int16_t *samples,

int *frame_size_ptr,

uint8_t *buf,

int buf_size)

Description:

This function is used for decoding an audio frame.

Arguments:

AVCodecContext *avctx – [in] Points to the context of the CODEC used for decoding audio stream;

int16_t * samples – [out] Points to the buffer for storing the decompressed audio stream;

int *frame_size_ptr -- [out] Points to the variable used for storing the decompressed size in BYTES.

uint8_t *buf – [in] Points to the buffer storing the audio stream to be decompressed;

int buf_size -- [in] The size of the audio stream to be decompressed;

Returns:

int - 1, error;Others, the number of bytes have been used by decompressing

avcodec_encode_video

Synopsis:

int avcodec_encode_video(AVCodecContext *avctx,

uint8_t *buf,

int buf_size,

const AVFrame *pict)

Description:

This function is used for encoding video stream.

Arguments:

AVCodecContext *avctx -[in] Points to the CODEC context used for encoding video stream;

uint8_t *buf – [out] Points to the buffer for storing the returned encoded video stream;

int buf_size – [in] The buffer size of the buffer pointed to by buf;

const AVFrame *pict – [in] Points to the buffer storing the video stream to be encoded.

Returns:

int – The size of the encoded output stream in bytes;

avcodec_decode_video

Synopsis:

int avcodec_decode_video(AVCodecContext *avctx,

AVFrame *picture,

int *got_picture_ptr,

uint8_t *buf,

int buf_size)

Description:

This function is used for decoding a frame;

Arguments:

AVCodecContext *avctx – [in] Points to the context of the CODEC used for decoding the video;

AVFrame *picture – [out] Points to an AVFrame

int *got_picture_ptr – [out] 0, No frame could be decompressed; otherwise, it is nonzero;

uint8_t *buf – [in] Points to the begin of bit stream buffer, must be FF_INPUT_BUFFER_PADDING_SIZE larger then the

actual read bytes because some optimized bit stream readers read 32 or 64 bit at once and could read over the end;

int buf_size – [in] The size in bytes of the buffer pointed by buf;

Returns:

int -1, error; Others, the number of bytes have been used for decompressing;

avpicture_get_size

Synopsis:

int avpicture_get_size(int pix_fmt, int width, int height)

Description:

This function used for getting the necessary buffer size for the raw picture to be encoded;

Arguments:

int pix_fmt – Pixel format of the raw picture to be encoded;

int width – Width in pixel for the raw picture to be encoded;

int height – Height in pixel for the raw picture to be encoded;

Returns:

int – the size in bytes for the raw picture to be encoded;

avpicture_fill

Synopsis:

int avpicture_fill(AVPicture *picture, uint8_t *ptr, int pix_fmt, int width, int height)

Description:

This function is used for setting the data[i] and linesize[i] (the range of i is from 0 to 2) fields through computation.

Arguments:

AVPicture *picture – [in] Points to the AVPicture whose data and linesize field is to be filled;

uint8_t *ptr – [in] data[0] field is filled with ptr;

data[1] is the start address for Y data;data[1] is the start address for U data;

data[2] is the start address for V data;

int pix_fmt – [in] Pixel format of the raw picture;

int width – [in] Width in pixel of the raw picture;

int height – [in] Height in pixel of the raw picture;

Returns:

int - 1, the input pixel format cannot be recognized; Others, the size in bytes for the raw picture to be encoded

av_mallocz

Synopsis:

void *av_mallocz(unsigned int size)

Description:

This function is used for allocating memory for libavcodec; the allocated memory is aligned at 16-byte boundry.

Arguments:

unsigned int size – The specified size of memory to be allocated;

Returns:

void * - null, no memory can be allocated; Others, points to the allocated memory;

avcodec_alloc_context

Synopsis:

AVCodecContext *avcodec_alloc_context(void)

Description:

Allocates an AVCodecContext and set it to defaults.

Arguments:

None

Returns:

AVCodecContext * - null, no memory can be allocated;

The pointers to the allocated AVCodecContext;

Notes:

This can be deallocated by simply calling free();

avcodec_alloc_frame

Synopsis:

AVFrame *avcodec_alloc_frame(void)

Description:

This function is used for allocating an AVFrame and set it to defaults.

Arguments:

None;

Returns:

AVFrame * - null, no AVFrame is allocated;

Others, the pointer to the allocated AVFrame;

Notes:

This can be deallocated by simply calling free().

This function uses av_mallocz to allocate memory; so the allocated memory is 16-byte alignment

avcodec_get_context_defaults

Synopsis:

void avcodec_get_context_defaults(AVCodecContext *s)

Description:

This function is used for setting CODEC context with default values;

Arguments:

AVCodecContext *s – [in] Pointer to the CODEC context to be set with default values;

Returns:

None.

AV Format APIs

register_protocol

Synopsis:

int register_protocol(URLProtocol *protocol)

Description:

This function is used for registering file I/O operations.

Arguments:

URLProtocol *protocol – Points to the URLProtocol to be registered;

Returns:

int – 0;

mpegps_init

Synopsis:

int mpegps_init(void)

Description:

This function is used for registering the MPEGI and MPEG II formats;

Arguments:

None;

Returns:

int 0;

raw_init

Synopsis:

int raw_init(void)

Description:

This function is used for registering with the following raw formats:

Input format: ac3; h263; MPEG4; H264; MPEG video; MJPEG video; raw video;

For output format, null raw video is added besides the above formats;

Arguments:

None;

Returns:

int – 0;

mp3_init

Synopsis:

int mp3_init(void)

Description:

This function is used for registering with the following formats,

Input: mp3;

Output: mp2, mp3;

Arguments:

None;

Returns:

int – 0;

wav_init

Synopsis:

int wav_init(void)

Description:

This function register wave format;

Arguments:

None;

Returns:

int – 0;

guess_format

Synopsis:

AVOutputFormat *guess_format(const char *short_name,

const char *filename,

const char *mime_type)

Description:

This function is used for finding the matched output file format. It will match according the following parameters: short_name, mime_type, filename;

Arguments:

const char *short_name – [in] File name;

const char *filename –[in] File extension;

const char *mime_type –[in] Mime type;

Returns:

AVOutputFormat * - Null, no matching file format exist; Others, the pointer to the matched file format;

url_fopen

Synopsis:

int url_fopen(ByteIOContext *s, const char *filename, int flags)

Description:

This function opens a file with a registered URLProtocol.

Arguments:

ByteIOContext *s – [out] A pointer to ByteIOContext, its fields will be set according to the specified filename and flags;

const char *filename – [in] The filename of the file to be opened;

int flags -- #define URL_RDONLY 0

#define URL_WRONLY 1

#define URL_RDWR 2

Returns:

int – status of file open:

0, ok;

-ENOMEM, no memory;

-EIO, IO error;

Notes:

When opened as read/write, the buffers are only used for reading.

url_fclose

Synopsis:

int url_fclose(ByteIOContext *s)

Description:

This function is used for freeing the buffer allocated to the specified byte IO context;

Arguments:

ByteIOContext *s – [in] Pointer to the ByteIOContext, the buffer of which will be freed.

Returns:

0, ok;

Others, IO error;

Decided by the registered URLProtocol;

av_write_frame

Synopsis:

int av_write_frame(AVFormatContext *s,

int stream_index,

const uint8_t *buf,

int size)

Description:

Write a packet to an output media file. The packet shall contain one audio or video frame.

Arguments:

AVFormatContext *s – [in] Media file handle;

int stream_index – [in] Stream index;

const uint8_t *buf – [in] Points to buffer containing the frame data;

int size – [in] Size of buffer in bytes;

Returns:

int – status of frame write

< 0, error,

= 0, ok,

=1, end of stream wanted.

av_set_parameters

Synopsis:

int av_set_parameters(AVFormatContext *s, AVFormatParameters *ap)

Description:

This function currently is only used for setting pixel format if not YUV420P

Arguments:

AVFormatContext *s – [in] Media file handle;

AVFormatParameters *ap – [in] Pointer to the audio/video parameters structure

Returns:

int – status of parameters setting

0, OK;

<0, Error;

Others, no use;

av_write_header

Synopsis:

int av_write_header(AVFormatContext *s)

Description:

Allocate the stream private data and write the stream header to an output media file

Arguments:

AVFormatContext *s - media file handle

Returns:

int – 0, ok; AVERROR_xxx, error.

av_probe_input_format

Synopsis:

AVInputFormat *av_probe_input_format(AVProbeData *pd, int is_opened)

Description:

This function is used for detecting file or stream format.

Arguments:

AVProbeData *pd – The pointer to the AVProbeData containing the data to be checked. The following fields of filename: buf, buf_size of this should be set;

int is_opened – 1, file is opened;

0, file is not opened;

Returns:

AVInputFormat * - null, not found;

Others, points to the AVInputFormat matching the specified stream of file;

av_write_trailer

Synopsis:

int av_write_trailer(AVFormatContext *s)

Description:

This function is used for writing the stream trailer to an output media file and free the file private data.

Arguments:

AVFormatContext *s – [in] media file handle

Returns:

int – 0, OK;AVERROR_xxx, error.

av_gettime

Synopsis:

int64_t av_gettime(void)

Description:

This function is used for getting system time in microseconds;

Arguments:

None

Returns:

int64_t – System time in microseconds;

av_new_stream

Synopsis:

AVStream *av_new_stream(AVFormatContext *s, int id)

Description:

This function is used for adding a new stream to a media file.

Arguments:

AVFormatContext *s - Media file handle;

int id - File format dependent stream id;

Returns:

AVStream * - Null, no stream can be allocated; Others, the pointer to the allocated AVStream;

Notes:

It can only be called in the read_header function.

If the flag AVFMTCTX_NOHEADER is in the format context, then new streams can be added in read_packet too.

dump_format

Synopsis:

void dump_format(AVFormatContext *ic,

int index,

const char *url,

int is_output)

Description:

This function dumps the audio / video format to standard output.

Arguments:

AVFormatContext * – [in] Points to the AVFormatContext to be dumped;

int index – not used;

const char *url – [in] Pointer to the sting for URL;

int is_output – [in] 1 on output; 0 on input;

Returns:

None;

av_write_header

Synopsis:

int av_write_header(AVFormatContext *s)

Description:

Allocate the stream private data and write the stream header to an output media file

Arguments:

AVFormatContext *s –[in] Media file handle;

Returns:

int – 0, OK;AVERROR_xxx, error;

Structures and Constants

Pease refer to libavcodec\avcodec.h for the following structures and constants:

```
AVCodec
AVCodecContext
AVFrame
AVPicture
enum CodecID
```

Please refer to libavformat\Avformat.h for the following structures and constants:

```
AVFormatContext
AVProbeData
AVStream
AVFormatParameters
```

Please refer to libavformat\avio.h for the following structures and constants:

```
ByteIOContext
URLProtocol
```

Please refer to libavcodec\common.h for the following types:

```
int16_t
uint8_t
int64_t
```

5.7 CServer Interface

Purpose

CServer is an application level class developed based on CVideoCapture. This class includes a series of methods through calling Linux thread APIs to implement its own functions. These methods presenting the go-server features are implemented in this class.

Method Summary

Methods	Description
Start	Start server.
Stop	Stop server.
init_channel	Initialize rtp or sockets for multicast packets.
do_work_thread	Get video frame from chip to its own stream buffer.
do_audio_work_thread	This function gets audio sample from driver
do_send_thread	Multicast video and audio frame from its own buffer to go-client
do_send_video_thread	Multicast only video frame from its own buffer to go-client
do_md_thread	Enable motion detection thread
do_sigdet_thread	Detect if signal lost
send_one_frame	Send one video and audio frame packet by packet to go-client
do_mux_thread_simple	Mux video and audio stream into a media file
IsPFrame	Determine whether the frame is a P frame.
GetVideoSource	Gets the current video input source.
SetVideoSource	Set video input source on CServer level.
SetTVStandard	Set TV standard on CServer level.

Method Description

Start

Synopsis:

int Start (TCFGVIDEO *videoconfig,

AUDIO_CONFIG *audioconfig,

char *multicast_addr,

char *save_filename,

int use_rtp,

int rtp_port)

Description:

This function starts the following threads according to different configuration:

do_work_thread, do_audio_work_thread, do_send_thread, do_send_video_thread, do_md_thread, do_sigdet_thread, do_mux_thread_simple

Arguments:

TCFGVIDEO *videoconfig – [in] Pointer to TCFGVIDEO, a completed video setting, caller is responsible to allocate memory for it.

AUDIO_CONFIG *audioconfig – [in] Pointer to AUDIO_CONFIG, a audio setting, caller is responsible to allocate memory for it.

char *multicast_addr – [in] Pointer to multicast address, caller is responsible to ensure the validity of this address.

char *save_filename – [in] save file name.

int use_rtp – [in] A flag of if do rtp transport.

int rtp_port – [in] The rtp port value.

Returns:

int – status of if server start successfully.

0 – success.

1 – the device status error, not in stop status and cannot be start again

2 – the initialization of transmission channel for rtp or multicast failed

3 – start capturing video from chip failed

-1 – file I/O operation failed

Stop

Synopsis:

int Stop ()

Description:

This function stops all actions running in server and change device to stop status. Also it deletes the buffers of server.

Arguments:

None

Returns:

int – status of if stop successfully.

0 – success

1 – device error and device is not running

init_channel

Synopsis:

int init_channel (char *addr)

Description:

This function initializes RTP packets or socket for multicasting packets.

Arguments:

char *addr – [in] Pointer to ip address for RTP transport or multicast. Caller is responsible for ensuring the validity of this value.

Returns:

int – status of if channel initialization successfully.

0 – success

-1 – RTP initialization failed

1 – Socket initialization error

Notes:

If need multicast, this function will be called.

do_work_thread

Synopsis:

void do_work_thread ()

Description:

This function gets one video frame from the chip to its own stream buffer.

Arguments:

None

Returns:

None

Notes:

This function is called by the work_thread. It will always run and get video frame to its own stream buffer until the Stop () function is called by server.

do_audio_work_thread

Synopsis:

void do_audio_work_thread ()

Description:

This function gets audio sample from driver and write them to test.wav.

Arguments:

None

Returns:

None

Notes:

This function is called by the audio_work_thread. It will always run and get audio sample to its own stream buffer until the Stop () function is called by server.

do_send_thread

Synopsis:

void do_send_thread ()

Description:

This function multicast video frames and audio samples from send buffer to go-client.

Arguments:

None

Returns:

None

do_send_video_thread

Synopsis:

void do_send_video_thread ()

Description:

This function only multicasts video from its own buffer to the go-client.

Arguments:

None

Returns:

None

do_md_thread

Synopsis:

void do_md_thread ()

Description:

This function enables motion detection.

Arguments:

None

Returns:

None

do_sigdet_thread

Synopsis:

void do_sigdet_thread ()

Description:

This function enables signal detection.

Arguments:

None

Returns:

None

send_one_frame

Synopsis:

int send_one_frame (char *pframe, int frame_len, unsigned char SubId)

Description:

This function sends one video or audio frame packet by packet through RTP transport or multicast.

Arguments:

char *pframe – [in] Pointer to buffer of video or audio which is to be sent. Caller is responsible to allocate buffer big enough.

int frame_len – [in] buffer length, caller is responsible to ensure its validity.

unsigned char SubId – [in] type of stream, 0 is video stream and 1 is audio stream

Returns:

int – flag of if send one frame successfully.

0 – success

-1 – send error

do_mux_thread_simple

Synopsis:

void do_mux_thread_simple ()

Description:

This function muxes audio and video stream into a file.

Arguments:

None.

Returns:

None.

IsPFrame

Synopsis:

static bool IsPFrame (unsigned char* pdata, unsigned long len)

Description:

This function specifies whether the frame is a P frame.

Arguments:

Unsigned char *pdata – [in] buffer to a frame, caller is responsible to allocate buffer big enough.

Unsigned long len – [in] frame length

Returns:

bool – whether is a P frame

true – is a P frame

false – is not a P frame

GetVideoSource

Synopsis:

SINT32 GetVideoSource (void)

Description:

This function gets the current video in source.

Arguments:

None.

Returns:

SINT32 – the input video source retrieved from board module.

0 – Composite

1 – SVideo

2 – TV tuner

-3 – Unknown

SetVideoSource

Synopsis:

SINT32 SetVideoSource (SINT32 source)

Description:

This function set video input source on CServer level.

Arguments:

SINT32 source – [in] video input source, user is responsible to ensure the validity of input source. The valid value range is within the enumeration:

VIDEO_SOURCE, defined in Include/struct.h.

Returns:

SINT32 – status of if set video input source successfully.

Notes:

This function will check the validity of input source. The input source is one of value of VIDEO_SOURCE; if invalid it does nothing.

SetTVStandard

Synopsis:

SINT32 SetTVStandard (UINT32 mask)

Description:

This function set TV standard on CServer level.

Arguments:

UINT32 mask – [in] TV standard mask, user is responsible to ensure the validity of TV standard value. The valid value range is within the enumeration:

TV_STANDARD, defined in Include/struct.h.

Returns:

SINT32 – status of whether TV standard set successfully

0 – set TV standard successfully

-1 – no match configuration or argument error

-3 – error of unknown register

5.8 Motion Detector Interface

Purpose

This provides a motion detection API at CServer level.

Method Summary

Methods	Description
MotionDetector	Create MotionDetector
MDSetEnable	Set MD status.
SetPicSize	Set macro block size for MD.

Method Description

MotionDetector

Synopsis:

MotionDetector::MotionDetector (IVideoCapture** vc, int width, int height)

Description:

This function create a motion detector.

Arguments:

IVideoCapture** vc – [in] pointer of pointer to IVideoCapture struct.

int width – [in] the input source piture width, in pixel unit.

int height – [in] the input source picture height, in pixel unit.

Returns:

None

MDSetEnable

Synopsis:

Int MotionDetector::MDSetEnable(int bEnable)

Description:

This function set motion detection status.

Arguments:

Int bEnable – [in] motion detection status.

0: MD_STATE_DISABLE_NO_THRESHOLD ,disabled and no threshold defined;

1: MD_STATE_DISABLED , disabled with previous set threshold

2: MD_STATE_ENABLED , enabled

Returns:

Int – status of md setting.

SET_SUCCESS – 0, set MD success.

SET_DISABLE_FAILURE – 1, set MD threasholds failure.

DISABLE_ALREADY – 2, disable MD already.

ENABLED_ALREADY – 3, enable MD already.

NO_THRESHOLD_DEFINED – 4, enable MD but no threasholds for it.

SET_ENABLE_FAILURE – 5, enable MD failure for the macro block map with an invalid region number; or set MD threasholds error.

SetPicSize

Synopsis:

Void MotionDetector::SetPicSize(int width,int height)

Description:

This function set macro block size for picture, in pixel unit.

Arguments:

int width – [in] input picture width, in pixel unit.

int height– [in] input picture height, in pixel unit.

Returns:

None.

5.9 Structures and Enumerations

This section describes the important structure for developing SDK APIs and calling APIs. Note that this section is only for reference and will change without any notice. For detailed information, please refer to file: Include\struct.h

REVISION_INFO Structure

The REVISION_INFO structure describes the board reversion information.

Include

Include\struct.h

Syntax

```
typedef struct {
     int        DriverMajor;
     int        DriverMinor;
     int        BoardRevision;
     char  BoardName[MAX_NAME];
     int        BoardCapability;
     int        MaxBandWidth;
     int        SourceWidth;
     int        SourceHeight;
} REVISION_INFO;
```

Members

driverMajor

Major ID of device

driverMinor

Minor ID of device

boardRevision

Board revision ID

boardName

Board name, use a string less than MAX_NAME (64) characters included in quotes.

boardCapability

An integer with each bit representing one kind of capability of board, using values in Enumeration: BOARD_CAP.

maxBandWidth

Reserved for future use.

sourceWidth

Width of source video

sourceHeight

Height of source video

TCFGVIDEO Structure

The TCFGVIDEO structure describes a complete video setting, including miscellaneous setting, stream setting, resolution setting, frame rate setting and bitrate control setting.

Include

Include\struct.h

Syntax

```
typedef struct{
     TCFGMISC   misccfg;
     TCFGSTREAM strcfg;
     TCFGRESOLUTIONrescfg;
     TCFGFRAMERATEfpscfg;
     TCFGBRCTRL ctlcfg;
} TCFGVIDEO;
```

Members

misccfg

A TCFGMISC structure for miscellaneous settings.

strcfg

A TCFGSTREAM structure for stream settings.

rescfg

A TCFGRESOLUTION structure for resolution settings.

fpscfg

A TCFGFRAMERATE structure for frame rate settings.

ctlcfg

A TCFGBRCTRL structure for bitrate control settings.

AUDIO_CONFIG Structure

The AUDIO_CONFIG structure describes the audio setting.

Include

Include\struct.h

Syntax

```
typedef struct _AUDIO_CONFIG{
     unsigned longFormat;
     unsigned longSampleRate;
     unsigned longChannels;
     unsigned longSampleBits;
     unsigned shortBlockAlign;
     unsigned longAvgBytesPerSec;
     unsigned shortSamplesPerBlock;
     unsigned shortExtSize;
} AUDIO_CONFIG;
```

Members

Format

Audio format, refer to Enumeration: AUDIO_FORMAT.

SampleRate

Audio sample rate, in byte. Possible values are 44100, 48000, etc.

Channels

Audio channels. Possible values are 1 for Mono and 2 for Stereo.

SampleBits

Audio sample bits. Possible values are 8 bits and 16 bits.

TCFG_HEADER Structure

The TCFG_HEADER structure is used in structures: TCFGSYSTEM, TCFGSTREAM, TCFGFRAMERATE, TCFGRESOLUTION, TCFGBRCTRL and TCFGMISC. It provides general information about the structure it resides in.

Include

Include\struct.h

Syntax

```
typedef struct{
     char            name[MAX_NAME];
     char            desc[MAX_DESC];
     unsigned longflags;
     unsigned longsize;
} TCFG_HEADER;
```

Members

Name

The name of the configuration. Use a string less than MAX_NAME (64) characters included in quotes.

Desc

The description of the configuration. Use a string less than MAX_DESC (256) characters included in quotes.

Flags

The flags member provides information of what fields are provided in the structure where the TCFG_HEADER is located. Each bit in flags corresponds to one field.

Size

The size of the structure where the TCFGHEADER is located.

TCFGMISC Structure

The TCFGMISC structure describes miscellaneous settings of the output video stream of encoder chip.

Include

Include\struct.h

Syntax

```
typedef struct{
      TCFG_HEADERheader;
      unsigned char av_sync_enable;
      unsigned chariip_enable;
      unsigned charvbi_enable;
      unsigned charfour_channel_enable;
      FilterMode   h_filter_mode;
      FilterMode   v_filter_mode;
      char            filter_nAX;
      char            filter_nBX;
      char            filter_nCX;
      char            filter_nAY;
      char            filter_nBY;
      char            filter_nCY;
      long            reserved;
} TCFGMISC;
```

Members

header

Header information about the structure.

av_sync_enable

0: disable Audio/Video Synchronization algorithm

1: enable Audio/Video Synchronization algorithm

iip_enable

Specifies if enabling Input Image Processing. Only valid when sensor pixel format is RGB Bayer.

0: disable IIP

1: enable IIP

vbi_enable

0: disable VBI

1: enable VBI

four_channel_enable

If four_channel_enable is set, each frame of the encoded stream will be divided into four quadrants. Motion search will be confined in each quadrant and will not be performed in other quadrants.

0: disable four channel feature

1: enable four channel feature

h_filter_mode

The mode of pre-filtering in a horizontal direction.

0: No pre-filtering in the horizontal direction before encoding

1: Median filter applied in the horizontal direction before encoding

2: Linear filter applied in the horizontal direction before encoding

v_filter_mode

The mode of pre-filtering in a vertical direction.

0: No pre-filtering in the vertical direction before encoding

1: Median filter applied in the vertical direction before encoding

2: Linear filter applied in the vertical direction before encoding

filter_nAX

filter_nBX

filter_nCX

The coefficients of linear filter in horizontal direction. Valid only if h_filter_mode is 2. A typical requirement for the coefficients is:filter_nAX + filter_nBX + filter_nCX = 16.

filter_nAY

filter_nBY

filter_nCY

The coefficients of linear filter in vertical direction. Valid only if v_filter_mode is 2. A typical requirement for the coefficients is: filter_nAY + filter_nBY + filter_nCY = 16.

reserved

Reserved for future use.

TCFGSTREAM Structure

The TCFGSTREAM structure describes settings of the output video stream of encoder chip.

Include

Include\struct.h

Syntax

```
typedef struct{
      TCFG_HEADERheader;
      EVideoFormatcompress_mode;
      ESequenceModesequence;
      unsigned char   gop_mode;
      unsigned chargop_size;
      unsigned charmpeg4_mode;
      unsigned char   DVD_compliant;
      unsigned chardeinterlace_mode;
      unsigned charsearch_range;
      unsigned chargop_head_enable;
      unsigned charseq_head_enable;
      unsigned charaspect_ratio;
      long                reserved;
} TCFGSTREAM;
```

Members

header

Header information about the structure.

compress_mode

The stream's compression mode, refer to Enumeration: EVideoFormat.

MPEG1: 0x00

MPEG2: 0x01

H263: 0x03

MPEG4: 0x04

MOTIONJPEG: 0x08

sequence

The types of frame being used in a stream sequence. Refer to Enumeration:ESequenceMode for details.

GOP_MODE_OPEN: 0

IONLY: Only I-frames in the stream sequence (1)

IPONLY: Only I and P frames in the stream sequence (2)

IPB: All types of frames (I, P and B) in the stream sequence (3)

gop_mode

The GOP (Group of Picture) mode of the encoding stream sequence. EncodingDivX format MPEG4 stream requires closed GOP mode.

GOP_MODE_OPEN:0

GOP_MODE_CLOSE:1

gop_size

The Group of Pictures (GOP) size. This value is the key frame interval.

mpeg4_mode

MPEG4 stream mode. Valid only when compress_mode is 4.

WIS_MPEG4:0

DIVX_MPEG4: 1

MICROSOFT_MPEG4:2

DVD_compliant

Specify if the stream is desired to be DVD compliant. Valid only when the compression mode is MPEG2.

0: disable DVD_compliant

1: enable DVD_compliant deinterlace_mode

0: use one field only

1: use WIS deinterlace algorithm

2: interlace coding is used; no deinterlacing performed

search_range

The searching range for motion vectors. Typical values are 16, 32, 64 or 128. Microsoft format MPEG4 stream requires that search range must be 64. H.263 format stream requires that search range must be 32.

gop_head_enable

Only encoding Microsoft format MPEG4 stream requires disabling GOP head. ll other format streams require enabling GOP head.

0: isable GOP head

1: nable GOP head

seq_head_enable

Only encoding Microsoft format MPEG4 stream requires disabling sequence head. All other format streams require enabling sequence head.

0: isable sequence head

1: nable sequence head

aspect_ratio

The ratio between the width and the height of the picture.

1: 1:1

2: 4:3

3:16:9

reserved

Reserved for future use.

TCFGFRAMERATE Structure

The TCFGFRAMERATE structure describes frame rate related settings of the output video stream of encoder chip.

Include

Include\struct.h

Syntax

```
typedef struct{
      TCFG_HEADERheader;
      TV_STANDARDtv_standard;
      unsigned longframe_rate;
      unsigned long   drop_frame;
      unsigned charivtc_enable;
      long            reserved;
} TCFGFRAMERATE;
```

Members

hader

Header information about the structure.

tv_standard

Can be only set as TVStandard_NTSC_Mask or VStandard_PAL_Mask

frame_rate

output frame rate.

drop_frame

0:keep original frame rate. No frames dropped.

1:keep 1/2 original frame rate.

2:keep 1/3 original frame rate.

n:keep 1/(n+1) original frame rate.

ivtc_enable

IVTC (InVerse TeleCine) is a process where video editing tools reverse Telecine rocess. Basically IVTC brings back movie's original frame rate from NTSC's 9.97fps to 24fps.

0: disable IVTC

1: enable IVTC

reserved

Reserved for future use.

TCFGRESOLUTION Structure

The TCFGRESOLUTION structure describes the resolution of encoded stream.

Include

Include\struct.h

Syntax

```
typedef struct{
     TCFG_HEADER     header;
     TV_STANDARD     tv_standard;
     unsigned longwidth;
     unsigned longheight;
     unsigned charh_sub_window;
     unsigned charv_sub_window;
     unsigned longh_sub_offset;
     unsigned longv_sub_offset;
     unsigned charh_scale_enb;
     unsigned charv_scale_enb;
     unsigned charsub_sample;
     unsigned longmax_bitrate;
     unsigned longmin_bitrate;
     long                 reserved;
} TCFGRESOLUTION;
```

Members

hader

Header information about the structure.

tv_standard

Can only be set as TVStandard_NTSC_Mask or VStandard_PAL_Mask

width

The desired output stream resolution: horizontal size.

Height

The desired output stream resolution: vertical size.

h_sub_window

Specify if performing sub-window (cropping) in the horizontal direction.

0:disable sub-window

1:enable sub-window

v_sub_window

Specify if performing sub-window (cropping) in the vertical direction.

0:disable sub-window

1:enable sub-window

h_sub_offset

If h_sub_window is set, it specifies a relative offset between the leftmost pixel of he output stream and the leftmost pixel of the source stream, in pixels.

v_sub_offset

If v_sub_window is set, it specifies a relative offset between the topmost pixel of he output stream and the topmost pixel of the source stream, in pixels.

h_scale_enb

Specify if performing ½ scaling in the horizontal direction.

0:disable scaling

1:enable scaling

v_scale_enb

Specify if performing ½ scaling in the vertical direction.

0:disable scaling

1:enable scaling

sub_sample

Specify if performing sub sampling. Sub-sampling will perform ½ scaling down o the stream in both horizontal and vertical directions.

0:disable sub_sampling

1:enable sub_sampling

max_bitrate

The maximum bit rate allowed for this resolution.

min_bitrate

The minimum bit rate allowed for this resolution.

Reserved

Reserved for future use.

TCFGBRCTRL Structure

The TCFGBRCTRL structure describes the bitrate control setting of encoded stream.

Include

Include\struct.h

Syntax

```
typedef struct{
      TCFG_HEADERheader;
      unsigned longtarget_bitrate;
      unsigned longpeak_bitrate;
      unsigned longvbv_buffer;
      unsigned charconverge_speed;
      unsigned charlambda;
      unsigned longQ;
      unsigned charIQ;
      unsigned charPQ;
      unsigned charBQ;
      long                  reserved;
} TCFGBRCTRL;
```

Members

hader

Header information about the structure.

target_bitrate

The desired average target bit rate of encoded stream, in bits per second (bps).

0:If Q>0, apply variable bitrate control (VBR) using the value of Q. If Q=0, no bitrate control algorithm applied. Bitrate will be determined by values of IQ, PQ, BQ provided by user.

>0:Apply constant bitrate control (CBR) using the value of arget_bitrate.

peak_bitrate

The highest bit rate allowed in the encoded stream, in bps. This parameter is only valid when applying constant bitrate control (both Q and target_bitrate are greater than 0).

vbv_buffer

Specifies VBV (Video Buffering Verifier) buffer size.

converge_speed

Specifies the converging speed of bit rate control process, from 0 to 100.

lambda

The factor determining stream quality. Its value range is from 0 to 100.

Q

Initial quantizer. This value will be divided by 4.

0:If target_bitrate>0, apply constant bitrate control. Initial quantizer value will be calculated. If target_bitrate=0, no bitrate control algorithm applied. Use IQ, PQ and BQ provided by user to determine bitrate value.

>0:If target_bitrate is set to 0, apply VBR (variable bitrate) using the value of Q. If target_bitrate is greater than 0, apply CBR (constant bitrate) using the value of target_bitrate.

IQ

The fixed quantize scale for I-frames during the entire encoding session.

PQ

The fixed quantize scale for P-frames during the entire encoding session.

BQ

The fixed quantize scale for B-frames during the entire encoding session.

Reserved

Reserved for future use.

TFrameInfo Structure

The TFrameInfo structure provides the frame information.

Include

Include\VideoCapture.h

Syntax

```
typedef struct _TFrameInfo {
    SINT32 VideoLength;// video frame length
    SINT32 VBILength;
    SINT32AudioFingerprint;
    SINT32VIPNumber;
    SINT32AudioTimestamp; // time stamp for audio
    SINT32 FrameType;// I, P or B frame
    SINT32 MDRegionMap;  //  bitmap  region  for
     motion detection
    SINT32 MDMapFlag;
    UINT16 MDMacroblockMap[86];//
     MACROBLOCK_MAP_SIZE];
} TFrameInfo;
```

Members

BOARD_CAP Enumeration

Enumeration for encoder chip capability

Include

Include\struct.h

Syntax

```
typedef enum {
     BC_VIDEO= 0x00000001,
     BC_AUDIO= 0x00000002,
     BC_TVTUNER= 0x00000004,
     BC_XBAR    = 0x00000008,
     BC_VBI     = 0x00000010,
} BOARD_CAP;
```

AUDIO_FORMAT Enumeration

Audio format which encoder chip supports now

Include

Include\struct.h

Syntax

```
enum AUDIO_FORMAT
{
      AUDIO_FORMAT_PCM= 0x1,
      AUDIO_FORMAT_ADPCM_MS= 0x2,
      AUDIO_FORMAT_ADPCM_IMA= 0x6,
      AUDIO_FORMAT_ALAW= 0x7,
      AUDIO_FORMAT_ULAW= 0x11,
      AUDIO_FORMAT_MP3= 0x55
};
```

EVideoFormat Enumertion

Video format

Include

Include\Multimedia.h

Syntax

```
typedef enum {
     MPEG1      = 0x00,
     MPEG2      = 0x01,
     H261       = 0x02,
     H263       = 0x03,
     MPEG4      = 0x04,
     MPEG4XGO= 0x05,
     MPEG2X4    = 0x06,
     MOTIONJPEG= 0x08,
     DV              = 0x09,
     H26L       = 0x20,
     GO              = 0x40
} EVideoFormat;
```

ESeqenceMode Enumeration

Frame type in a video sequence

Include

Include\Multimedia.h

Syntax

```
typedef enum{
     IONLY     = 1, //  only  I  frame  in  this
     sequence
     IPONLY    = 2, //  only  I  and  P  frame  in
     this sequence
     IPB            = 3, // I, P and B frame in
     this sequence
     IPBDROP   = 4       //  I,  P  and  B  frame
     drop in this sequence
} ESequenceMode;
```

Typedef

The following data type is defined in “Typedef.h” file:

```
Typedef int SINT32;
Typedef char SINT8;
Typedef unsigned char UINT8;
Typedef unsigned short UINT16;
Typedef void VOID;
```

Appendix

Appendix A: Glossary

Brightness:

Attribute of a visual sensation according to which an area appears to exhibit more or less light

CCIR:

Committee Consulat International Radiotelegraphique. This is a standards committee of the International Telecommunications Union, which made the technical recommendation for European 625 line standard for video signals.

Composite Video:

Composite video (CVS/CVBS) signal carries video picture information for color, brightness, and synchronizing signals for both horizontal and vertical scans.

CIF:

CIF has 352(H) x 288(V) luminance pixels, and 176(H) x 144(V) chrominance pixels. QCIF is a similar picture format with one-quarter the size of CIF.

EIA:

Electronic Industry Association. An industry lobbying group; it collects statistics and establishes testing standards for many types of home electronics.

Field:

For interlaced video the total picture is divided into two fields, one even and one odd, each containing one half of the total vertical information. Each field takes one sixtieth of a second (one fiftieth for PAL) to complete. Two fields make a complete frame of video.

Frame:

One frame (two fields) of video contains the full vertical interlaced information content of the picture. For NTSC this consists of 525 lines and PAL a frame is consisted of 625 lines.

Gamma:

Cathode ray tubes (CRTs) do not have a linear relationship between brightness and the input voltage applied. To compensate for this non-linearity, a pre distortion or gamma correction is applied, generally at the camera source. A value of gamma equal to 2.2 is typical, but can vary for different CRT phosphors.

Hue:

Attribution of visual sensation according to which area appears to be similar to one, or proportions of two, of the perceived colors red, yellow, green, and blue.

NTSC:

Color TV standard developed in the U.S. in 1953 by National Television System Committee. NTSC is used in United States, Canada, Japan, in most of the American continent countries and in various Asian countries. The rest of the world uses either some variety of PAL or SECAM standards.

NTSC runs on 525 lines/frame and it's vertical frequency is 60Hz. NTSC's framerate is 29, 97 frames/sec.

PAL:

PAL (Phase Alternating Line) TV standard was introduced in the early 1960's in Europe. It has better resolution than in NTSC, having 625 lines/frame, but the frame rate is slightly lower - 25 frames/sec. PAL is used in most of the western European countries (except France, where SECAM is used instead), Australia, various countries in Africa and in South America and in some Asian countries. There are various versions of PAL, the most commonly used method is PAL B/G, but others include PAL I (used in the UK and in

Ireland) and PAL M (hybrid standard, having the same resolution as NTSC, but uses PAL transmission and color coding technology).

Saturation:

A characteristic describing color amplitude or intensity. A color of a given hue may consist of low or high saturation value, which relates to the vividness of color.

AGC

Abbreviation for automatic gain control. On a TV or VCR, AGC is a circuit that automatically adjusts the incoming signal to the proper levels for display or recording. On a video camera, AGC is a circuit that automatically adjusts the sensitivity of the pickup tube to render the most pleasing image.

Appendix B: Standard Compliance

Notice for USA

Compliance Information Statement (Declaration of Conformity Procedure) DoC FCC Part 15

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules.

These limits are designed to provide reasonable protection against harmful interference in a residential installation or when the equipment is operated in a commercial environment.

This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Notice for Europe

This product is in conformity with the Council Directive 89/336/EEC amended by 92/31/EEC and 93/68/EEC

This equipment has been tested and found to comply with EN55022/CISPR22 and EN55024/CISPR24. To meet EC requirements, shielded cables must be used to connect a peripheral to the card. This product has been tested in a typical class B compliant host system. It is assumed that this product will also achieve compliance in any class A compliant unit.

Warranty Policy

Thank you for choosing ADLINK. To understand your rights and enjoy all the after-sales services we offer, please read the following carefully.

1. Before using ADLINK's products please read the user manual and follow the instructions exactly. When sending in damaged products for repair, please attach an RMA application form which can be downloaded from: http://rma.adlinktech.com/policy/.
2. All ADLINK products come with a limited two-year warranty, one year for products bought in China:
   - The warranty period starts on the day the product is shipped from ADLINK's factory.
   - Peripherals and third-party products not manufactured by ADLINK will be covered by the original manufacturers' warranty.
   - For products containing storage devices (hard drives, flash cards, etc.), please back up your data before sending them for repair. ADLINK is not responsible for any loss of data.
   - Please ensure the use of properly licensed software with our systems. ADLINK does not condone the use of pirated software and will not service systems using such software. ADLINK will not be held legally responsible for products shipped with unlicensed software installed by the user.
   - For general repairs, please do not include peripheral accessories. If peripherals need to be included, be certain to specify which items you sent on the RMA Request & Confirmation Form. ADLINK is not responsible for items not listed on the RMA Request & Confirmation Form.

3. Our repair service is not covered by ADLINK's guarantee in the following situations:
   - Damage caused by not following instructions in the User's Manual.
   - Damage caused by carelessness on the user's part during product transportation.
   - Damage caused by fire, earthquakes, floods, lightening, pollution, other acts of God, and/or incorrect usage of voltage transformers.
   - Damage caused by unsuitable storage environments (i.e. high temperatures, high humidity, or volatile chemicals).
   - Damage caused by leakage of battery fluid during or after change of batteries by customer/user.
   - Damage from improper repair by unauthorized ADLINK technicians.
   - Products with altered and/or damaged serial numbers are not entitled to our service.
   - This warranty is not transferable or extendible.
   - Other categories not protected under our warranty.
4. Customers are responsible for shipping costs to transport damaged products to our company or sales office.
5. To ensure the speed and quality of product repair, please download an RMA application form from our company website: http://rma.adlinktech.com/policy. Damaged products with attached RMA forms receive priority.

If you have any further questions, please email our FAE staff: service@adlinktech.com.