ADLINK
TECHNOLOGY INC.

PCI-MPG24

4-CH MPEG4 Hardware

Video Compression Card

User's Manual

Manual Rev. 2.01

Revision Date: March 21, 2005

Part No: 50-15035-100

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 Introduction

The PCI-MPG24 is a MPEG4 hardware video compression card that provides four channels of real-time full D1 size MEGP4 video encoding and decoding with a preview function for digital video surveillance applications.

This 32-bit, 33MHz PCI bus frame grabber simultaneously captures and encodes four video analog streams in real time. It accepts standard composite color (PAL, NTSC) or monochrome video formats (CCIR, EIA) cameras inputs.

Each PCI-MPG24 card has a unique hardware ID number. System integrators can design protections to lock their system product. System integrators will benefit from a watchdog timer (for fault-tolerant applications) and easy-to-use standard connectors.

1.1 Features

Real-time MPEG4 Hardware Video Encoding

Supports real-time full D1* size, quarter or downscale video size encoding.

* D1 size video format:

- NTSC (640 x 480) at 30fps per channel
- PAL (768 x 576) at 25fps per channel
- Encoding Speed

NTSC (640 x 480)	1 Camera	2 Cameras	3 Cameras	4 Cameras
Frames	30	60	90	120
PAL (768 x 576)	1 Camera	2 Cameras	3 Cameras	4 Cameras
Frames	25	50	75	100

Table 1-1: Number of Frames

Adjustable Video Quality

Bit and frame rates are adjustable to fit variable bandwidths, as seen in remote Internet applications.

I, IP, IBP, and IBBP GOP structures are programmable for enhanced video quality.

Real-time Raw Data Preview

- Single channel: real-time preview and display by VGA resolution.

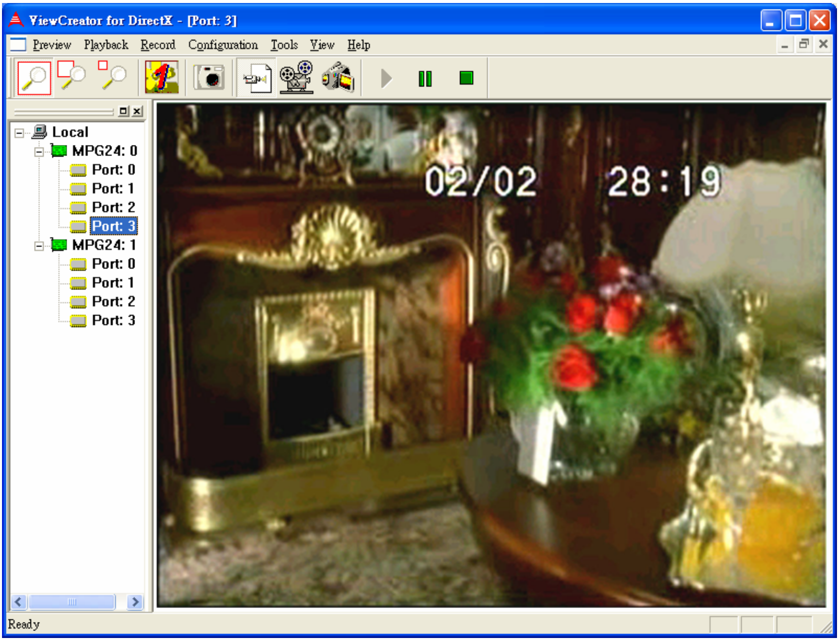


Figure 1-1: Real-time Raw Data Preview - single channel

- Four channels: real-time preview and display by quad format simultaneously.

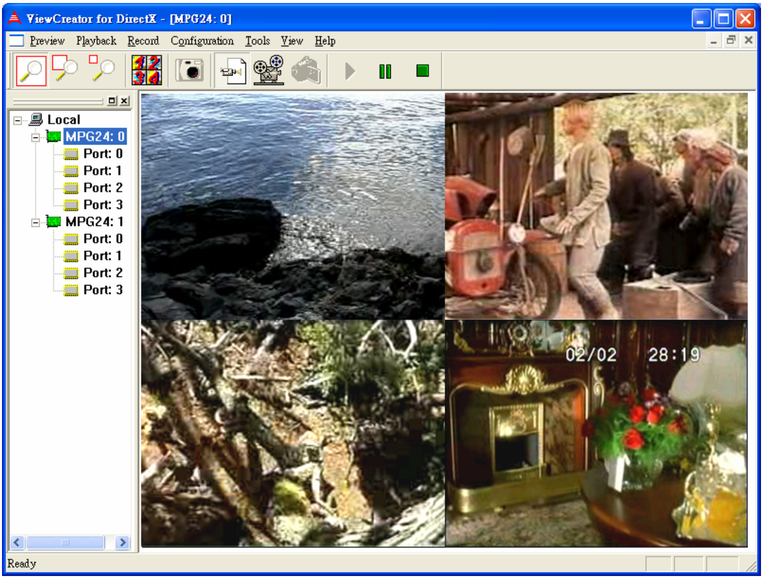


Figure 1-2: Real-time Raw Data Preview - four channels

Video Decoding

Smart software video decoding for playback or remote client monitoring and NO need to plug-in PCI-MPG24 card.

Save File

The video can be saved to AVI video file format. Users can playback AVI file by Microsoft Media Player.

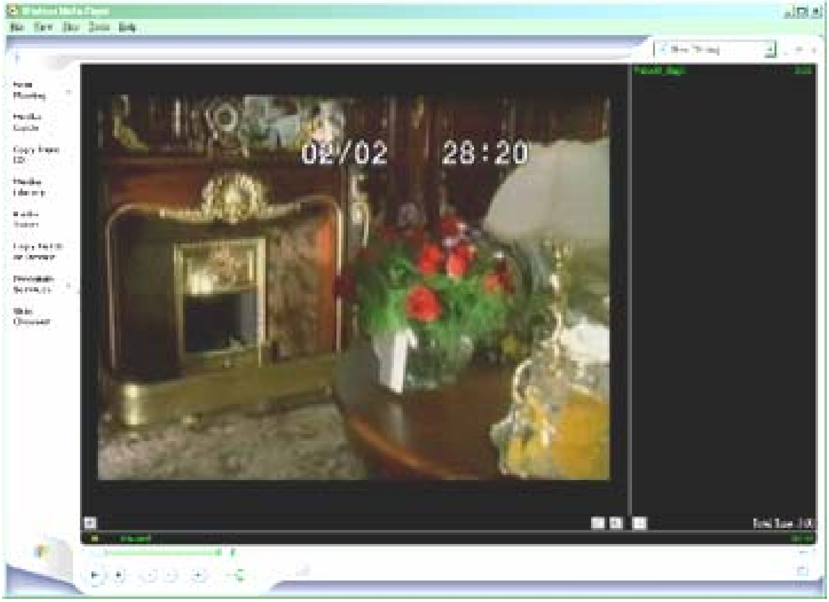


Figure 1-3: AVI video file format

I/O Lines

The PCI-MPG24 is fitted with TTL compatible I/O lines protected against overloads and electrostatic discharges. Each line may be configured as an input or output. They can be used to trigger acquisition or report alarm signals.

Watchdog Timer

A hardware watchdog is available on the PCI-MPG24 that is able to monitor PC application operation and will automatically reset the PC after a programmable inactivity time-out. This ensures reliable operation of remote systems.

Supported software

- Support Microsoft DirectX
- Support Visual Studio .net, VC++, and C++ Builder programming language
- Support Windows 2000 and Windows XP.
- Sample programs
- ‘ViewCreator for DirectX’ utility for assistance in the initial test and functional evaluation.

1.2 Applications

- PC Based Surveillance System
- Digital Video Recorder (DVR)
- Intelligent Traffic Monitoring System
- Factory Monitoring System
- Machine Vision Inspection System
- Scientific Research Instrumentation
- Medical Research Instrumentation

1.3 System Requirement

The PCI-MPG24 minimum system requirement as below:

- Platform: Pentium III, 850MHz CPU, and 512MB SDRAM or above.
- VGA display: AGP 4X above (Not recommended VIA or SiS VGA chipset solution).
- Display setting: 800 x 600 above resolution, 16-bit above color format.
- OS: Windows 2000 Professional with SP4 or Windows XP Professional with SP2
- Software requirement:
  - For end users: Microsoft DirectX 9.0 End-User Runtime
  - For developers: Microsoft DirectX 9.0 SDK
  - DivX Video Decoder (Optional)
- As software decoding consumes system resources, a system platform upgrade must be made for system decoding

requests of more than two channels, full D1 size real-time decode to specifications below:

  - ▷ Pentium 4, 2.4GHz CPU, 256MB DDR RAM above.

- Please refer to 1.4 PCI-MPG24 Benchmark for performance limitations.

1.4 PCI-MPG24 Benchmark

PCI-33 Platform

SBC:	ADLINK NuPRO-780
CPU:	Intel Pentium III, 850Hz
Memory:	512MB SDRAM
PCI Bus:	32-bit, 33MHz
VGA:	AGP 4X
OS:	Windows XP/SP1
HDD:	Maxtor 6E040LO 7200RPM

Configuration 1 – NTSC, CIF (320 x 240), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play-back
1	30	4M	22	11829K	O	20	13747K	O
		2M	19	8318K	O	19	8113K	O
		1M	20	4436K	O	24	4342K	O
2	30	4M	31	14356K	O	29	14740K	O
		2M	34	8531K	O	28	8098K	O
		1M	28	4497K	O	30	4371K	O
3	30	4M	36	130492K	O	36	15347K	O
		2M	39	8757K	O	36	8180K	O
		1M	37	4537K	O	34	4318K	O
4	30	4M	44	14265K	O	39	14628K	O
		2M	39	8676K	O	40	8200K	O
		1M	38	4550K	O	36	4333K	O
8	30	4M	58	10546K	O	64	13748K	O
		2M	63	8482K	O	56	6801K	O
		1M	66	4965K	O	71	5547K	O
12	30	4M	87	13497K	O	87	13689K	O
		2M	91	9225K	O	100	-	-
		1M	93	4887K	O	100	-	-

Table 1-2: Configuration 1 – NTSC, CIF (320 x 240)

Configuration 2 – NTSC, Full (640 x 480), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play-back
1	30	4M	22	14953K	O	22	14956K	O
		2M	22	7632K	O	21	7642K	O
		1M	23	4000K	O	22	3962K	O
2	30	4M	29	14963K	O	28	14966K	O
		2M	26	7638K	O	26	7669K	O
		1M	28	4015K	O	26	3971K	O
3	30	4M	35	14955K	O	35	14960K	O
		2M	33	7643K	O	34	7647K	O
		1M	34	4021K	O	32	3987K	O
4	30	4M	36	14812K	O	40	14610K	O
		2M	33	7649K	O	39	7657K	O
		1M	34	4044K	O	37	3999K	O
8	30	4M	71	14571K	O	70	14609K	O
		2M	67	7654K	O	64	7651K	O
		1M	55	4007K	O	57	3992K	O
12	30	4M	87	14029K	O	89	13976K	O
		2M	89	7673K	O	88	7687K	O
		1M	78	4018K	O	82	4103K	O

Table 1-3: Configuration 2 – NTSC, Full (640 x 480)

Configuration 3 – PAL, CIF (352 x 288), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play-back
1	25	4M	35	12226K	O	36	12868K	O
		2M	36	8485K	O	38	8205K	O
		1M	38	4308K	O	43	4224K	O
2	25	4M	40	12727K	O	40	14577K	O
		2M	40	8342K	O	47	8264K	O
		1M	42	4453K	O	50	4287K	O
3	25	4M	45	13444K	O	45	14690K	O
		2M	50	8452K	O	51	8216K	O
		1M	57	4447K	O	53	4255K	O
4	25	4M	45	14429K	O	49	14405K	O
		2M	51	8477K	O	58	8207K	O
		1M	60	4478K	O	65	4280K	O
8	25	4M	63	11618K	O	67	12335K	O
		2M	78	8463K	O	76	8417K	O
		1M	80	4457K	O	79	4208K	O
12	25	4M	84	13438K	O	92	13635K	O
		2M	86	8406K	O	88	8291K	O
		1M	90	4384K	O	91	4264K	O

Table 1-4: Configuration 3 – PAL, CIF (352 x 288)

Configuration 4 – PAL, Full (768 x 576), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play-back
1	25	4M	32	14290K	O	35	14904K	O
		2M	40	7604K	O	38	7596K	O
		1M	39	3914K	O	42	3934K	O
2	25	4M	38	14919K	O	41	14932K	O
		2M	50	7604K	O	51	7605K	O
		1M	47	3942K	O	50	3937K	O
3	25	4M	44	14912K	O	51	14904K	O
		2M	52	7596K	O	52	7594K	O
		1M	49	3939K	O	52	3954K	O
4	25	4M	46	14444K	O	47	14450K	O
		2M	57	7613K	O	54	7610K	O
		1M	57	3961K	O	58	3956K	O
8	25	4M	72	14370K	O	68	14505K	O
		2M	59	7612K	O	58	7651K	O
		1M	66	3992K	O	53	3968K	O
12	25	4M	92	13571K	O	89	13589K	O
		2M	87	7615K	O	83	7647K	O
		1M	79	3991K	O	79	4498K	O

Table 1-5: Configuration 4 – PAL, Full (768 x 576)

PCI-X Platform

SBC:	ADLINK NuPRO850
CPU:	Intel Pentium 4, 3.2GHz Hyper Threading Disable
Memory:	DDR266 1GB
PCI-X Bus:	32-bit, 66MHz
VGA:	AGP 8X
OS:	Windows 2000/SP4
HDD:	HITACHI ST340014A 7200RPM

Configuration 1 – NTSC, CIF (320 x 240), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play-back
1	30	4M	7	11827K	O	7	13382K	O
		2M	5	8156K	O	5	7936K	O
		1M	7	4303K	O	8	4248K	O
2	30	4M	8	13562K	O	8	14713K	O
		2M	7	8146K	O	7	8026K	O
		1M	8	4341K	O	8	4227K	O
3	30	4M	11	13566K	O	11	15008K	O
		2M	11	8360K	O	8	8036K	O
		1M	10	4342K	O	8	4249K	O
4	30	4M	13	14049K	O	11	14840K	O
		2M	10	8276K	O	10	7998K	O
		1M	10	4368K	O	10	4226K	O
8	30	4M	18	14705K	O	15	14883K	O
		2M	14	8652K	O	13	8492K	O
		1M	16	4598K	O	15	4477K	O
12	30	4M	22	14765K	O	18	14730K	O
		2M	16	8712K	O	13	8359K	O
		1M	16	4481K	O	15	4438K	O
16	30	4M	19	14880K	O	22	14952K	O
		2M	22	8798K	O	21	8389K	O
		1M	19	4418K	O	19	4386K	O

Table 1-6: Configuration 1 – NTSC, CIF (320 x 240)

Configuration 2 – NTSC, Full (640 x 480), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load (%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play-back
1	30	4M	7	14240K	O	8	14952K	O
		2M	7	7673K	O	7	7630K	O
		1M	7	3962K	O	7	3976K	O
2	30	4M	10	14718K	O	10	14988K	O
		2M	10	7642K	O	10	7661K	O
		1M	7	3980K	O	8	3986K	O
3	30	4M	8	14521K	O	10	14967K	O
		2M	11	7657K	O	10	7662K	O
		1M	10	3985K	O	10	4023K	O
4	30	4M	10	14930K	O	11	14934K	O
		2M	10	7674K	O	13	7665K	O
		1M	8	3993K	O	15	3992K	O
8	30	4M	13	14950K	O	19	14924K	O
		2M	16	7637K	O	18	7648K	O
		1M	13	4048K	O	11	4290K	O
12	30	4M	18	14912K	O	18	14940K	O
		2M	18	7650K	O	21	7680K	O
		1M	18	4036K	O	19	4597K	O
16	30	4M	25	14877K	O	25	14952K	O
		2M	21	7662K	O	24	7678K	O
		1M	19	4029K	O	18	4328K	O

Table 1-7: Configuration 2 – NTSC, Full (640 x 480)

Configuration 3 – PAL, CIF (352 x 288), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play-back	CPU Load(%)	Saving File Size	Play back
1	25	4M	5	12051K	O	5	14389K	O
		2M	5	8413K	O	4	8307K	O
		1M	5	4359K	O	5	4346K	O
2	25	4M	5	12549K	O	7	14812K	O
		2M	5	8364K	O	7	8330K	O
		1M	5	4337K	O	8	4326K	O
3	25	4M	7	12381K	O	7	14935K	O
		2M	8	8315K	O	8	8313K	O
		1M	5	4362K	O	8	4254K	O
4	25	4M	8	12374K	O	7	14860K	O
		2M	7	8413K	O	8	8339K	O
		1M	7	4336K	O	7	4300K	O
8	25	4M	13	14430K	O	10	14756K	O
		2M	11	8429K	O	11	8465K	O
		1M	13	4324K	O	11	4396K	O
12	25	4M	18	14775K	O	18	14645K	O
		2M	16	8368K	O	16	8437K	O
		1M	18	4339K	O	13	4398K	O
16	25	4M	18	14564K	O	19	14875K	O
		2M	22	8511	O	21	8416K	O
		1M	18	4453K	O	16	4364K	O

Table 1-8: Configuration 3 – PAL, CIF (352 x 288)

Configuration 4 – PAL, Full (768 x 576), 30 sec. Continuous Recording

Port	Frame Rate	BitRate	Codec: DivX			Codec: Microsoft		
			CPU Load(%)	Saving File Size	Play back	CPU Load(%)	Saving File Size	Play-back
1	25	4M	7	13755K	O	7	13720K	O
		2M	7	7625K	O	7	7614K	O
		1M	4	3942K	O	5	3935K	O
2	25	4M	7	14055K	O	8	24301K	O
		2M	8	7592K	O	7	7608K	O
		1M	8	3939K	O	5	3944K	O
3	25	4M	10	13707K	O	11	13971K	O
		2M	11	7675K	O	10	7615K	O
		1M	11	3994K	O	8	3954K	O
4	25	4M	11	14072K	O	10	14151K	O
		2M	11	7600K	O	11	7687K	O
		1M	8	3938K	O	10	4005K	O
8	25	4M	15	14707K	O	13	14890K	O
		2M	11	7611K	O	13	7613K	O
		1M	10	3952K	O	11	3962K	O
12	25	4M	16	14752K	O	15	14831K	O
		2M	13	7611K	O	18	7606K	O
		1M	15	3956K	O	15	4540K	O
16	25	4M	21	14767K	O	19	14908K	O
		2M	19	7620K	O	22	7599K	O
		1M	19	3957K	O	18	4692K	O

Table 1-9: Configuration 4 – PAL, Full (768 x 576)

2 Hardware Reference

2.1 PCI-MPG24 Specification

Video Input

- Four composite video color digitizers.
- Video input interface: DB15 pin female connector
- Support PAL/NTSC/CCIR/EIA standard cameras.

Video compress

- Four channels full D1 real time MPEG-4 video compress
- Advanced MPEG-4 bit-rate control (CBR/VBR) from 1Kbps to 40Mbps

Video preview

- Single channel full D1 size real-time preview
- Four channels quarter size real time preview

General Purpose I/O Lines

- The I/O lines are TTL compatible and support four inputs, four GPIO interfaces
- DB15 high density male connectors onboard
- The I/O lines are internally pulled up

Voltage	MIN	MAX
Input high voltage (5µA)	2.0V	5.25V
Input low voltage (-5µA)	0.0V	0.80V
Output high voltage (-1.0mA)	5.0V	-
Output low voltage (100.0mA)	-	0.5V

Table 2-1: Voltage Range

Watch Dog Timer

- For monitoring the PC's application operation and will reset the PC after a programmable inactivity time-out.
- Interface: 2-pin header

User EEPROM

- Support 1K bit EEPROM for user defined purposes

Form Factor

- 32bit/ 33MHz PCI bus half size board.

Power Consumption

- 3.3V @ 2.8A max
- 5V @ 0.8A max
- +12V @ 0.1A max
- -12V @ 0.1A max

PCI-MPG24 Appearance

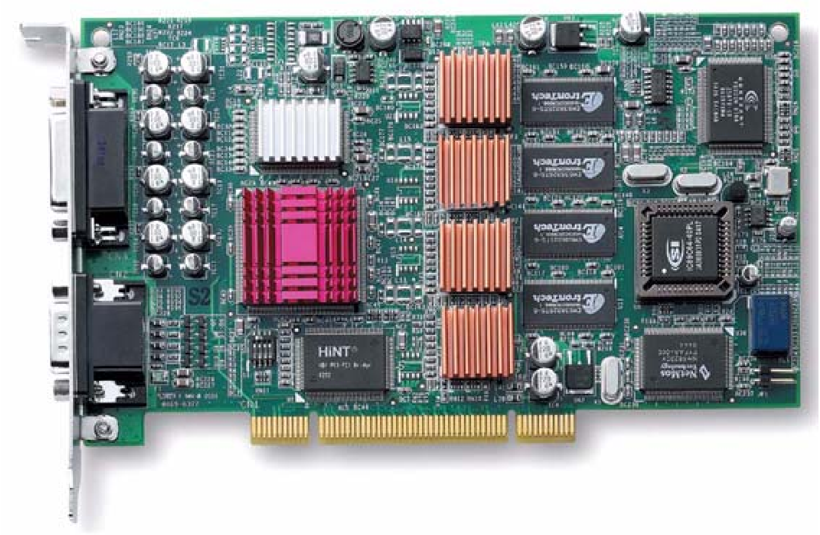


Figure 2-1: PCI-MPG24

PCI-MPG24 Standard accessories

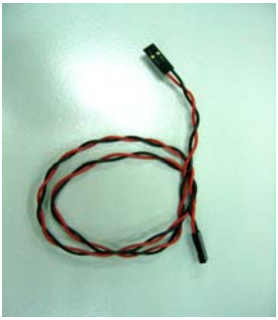


Figure 2-2: Watchdog reset cable

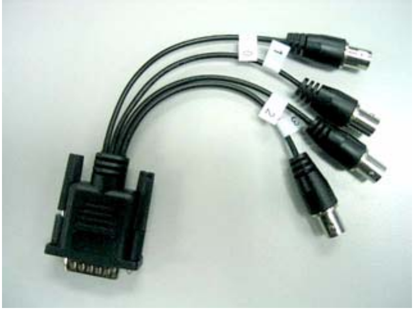


Figure 2-3: BNG Interface cable

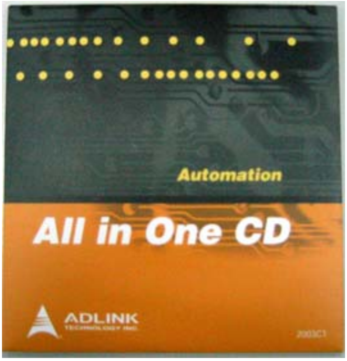


Figure 2-4: All in One CD

PCI-MPG24 Interface

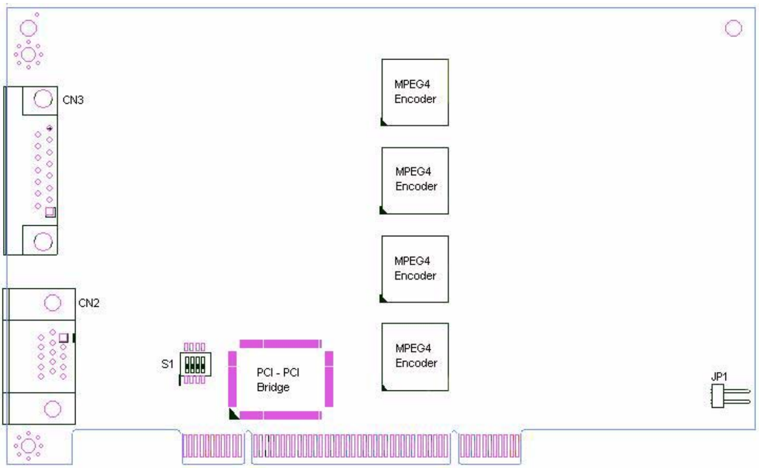
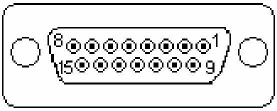


Figure 2-5: Outline Drawing

Connectors & Pin Definitions

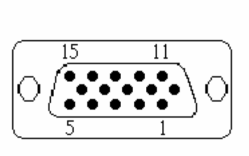
Video Inputs: CN3



Pin	Type	Function	Pin	Type	Function
1	IN	Video Port 0	9	--	GND
2	IN	Video Port 1	10	--	GND
3	IN	Video Port 2	11	--	GND
4	IN	Video Port 3	12	--	GND
5	--	--	13	--	GND
6	--	--	14	--	--
7	--	--	15	--	--
8	--	--			

Table 2-2: Video Inputs - CN3

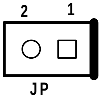
GPIO: CN2



Pin	Type	Function
1	IN	GPIO IN 1
2	IN	GPIO IN 2
3	IN	GPIO IN 3
4	IN	GPIO IN 4
5	G	GND
6	OUT	GPIO OUT 1
7	OUT	GPIO OUT 2
8	OUT	GPIO OUT 3
9	OUT	GPIO OUT 4
10	G	GND
11	G	GND
12	G	GND
13	G	GND
14	G	GND
15	P	+5V power output

Table 2-3: GPIO - CN2

Watchdog Timer Reset

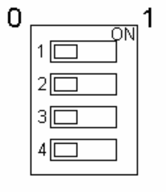


Pin	Function
1	System reset
2	GND

Table 2-4: Watchdog Timer Reset

DIP switch & Setting

S1: Card ID setting & NTSC/PAL mode setting



S1 Pin	Function	ON	OFF (Default)
1	Card ID BIT 0	1	0
2	Card ID BIT 1	1	0
3	Card ID BIT 2	1	0
4	NTSC / PAL	PAL	NTSC

Table 2-5: S1 Card ID setting & NTSC/PAL mode setting

Maximum support for 8 PCI-MPG24 cards in a single system

Card ID	S1 Pin3	S1 Pin2	S1 Pin1
0	OFF	OFF	OFF
1	OFF	OFF	ON
2	OFF	ON	OFF
3	OFF	ON	ON
4	ON	OFF	OFF
5	ON	OFF	ON
6	ON	ON	OFF
7	ON	ON	ON

Table 2-6: Pin setting for 8 PCI-MPG24 cards

3 Installation Guide

3.1 Software Driver Installation

Software Environment

1. Operating Systems Supported

- ▶ Windows 2000 Professional with SP4
- ▶ Windows XP Professional with SP2

2. Other necessary software packages

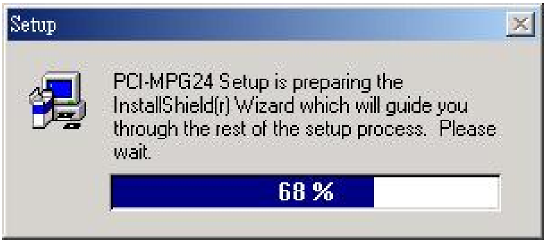
- ▶ Microsoft DirectX 9.0

Note:

- ▶ Install DirectX SDK before installing the PCI-MPG24 driver onto your system
- ▶ Do not plug the hardware before installing the software driver

Driver Installation

1. Double click **SETUP.exe** in the PCI-MPG24 setup disk. The driver will begin installing.

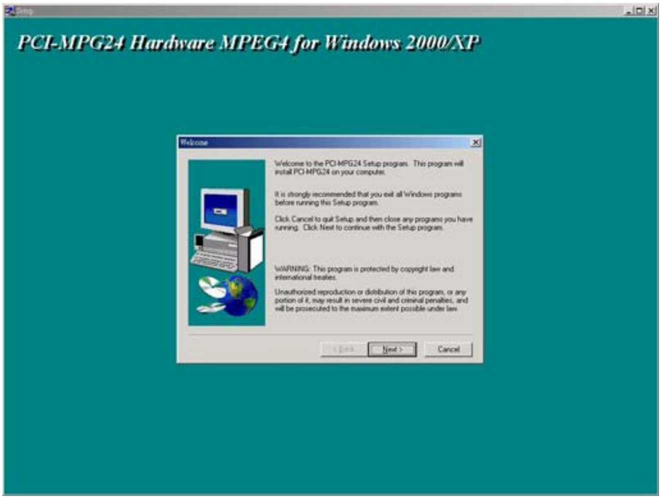


Setup

PCI-MPG24 Setup is preparing the InstallShield(r) Wizard which will guide you through the rest of the setup process. Please wait.

68 %

2. Click **Next** until the driver is installed completely.



PCI-MPG24 Hardware MPEG4 for Windows 2000/XP

Welcome

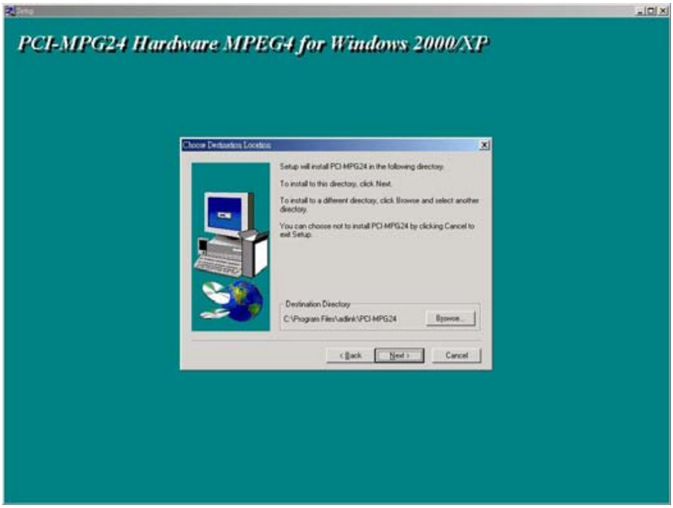
Welcome to the PCI-MPG24 Setup program. This program will install PCI-MPG24 on your computer.

It is strongly recommended that you exit all Windows programs before running this Setup program.

Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.



PCI-MPG24 Hardware MPEG4 for Windows 2000/XP

Choose Destination Location

Setup will install PCI-MPG24 in the following directory.

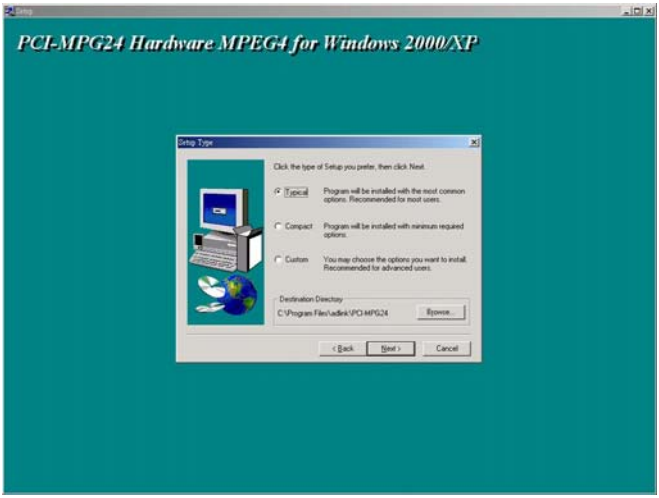
To install to this directory, click Next.

To install to a different directory, click Browse and select another directory.

You can choose not to install PCI-MPG24 by clicking Cancel to exit Setup.

Destination Directory

C:\Program Files\adlink\PCI-MPG24



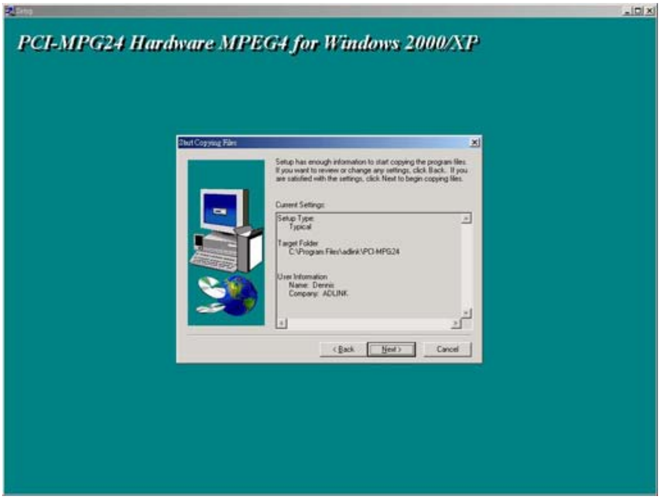
Click the type of Setup you prefer, then click Next.

● Typical — Program will be installed with the most common options. Recommended for most users.

○ Compact — Program will be installed with minimum required options.

○ Custom — You may choose the options you want to install. Recommended for advanced users.

Destination Directory: C:\Program Files\adlink\PCI-MPG24

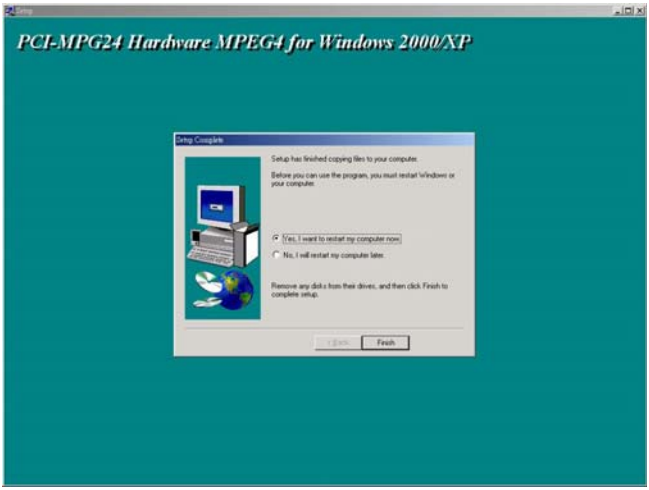


Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.

Current Settings:

```
Setup Type:
    Typical

Target Folder
    C:\Program Files\adlink\PCI-MPG24


User Information
    Name:  Dennis
    Company:  ADLINK
```

3. Click **Finish** to restart the system.



Setup has finished copying files to your computer.

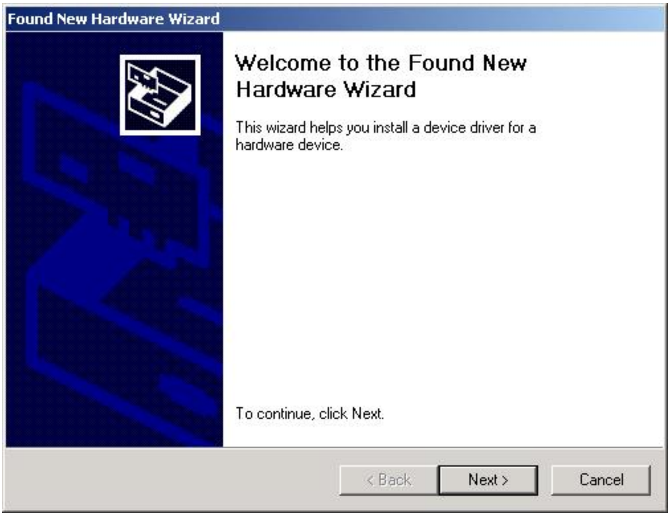
Before you can use the program, you must restart Windows or your computer.

● Yes, I want to restart my computer now.

○ No, I will restart my computer later.

Remove any disks from their drives, and then click Finish to complete setup.

4. After restarting the system, power off the system and insert the PCI-MPG24 card into your system. Power on the computer.
5. Windows should detect the new card and start **Found New Hardware** wizard. Click **Next** to start installation.



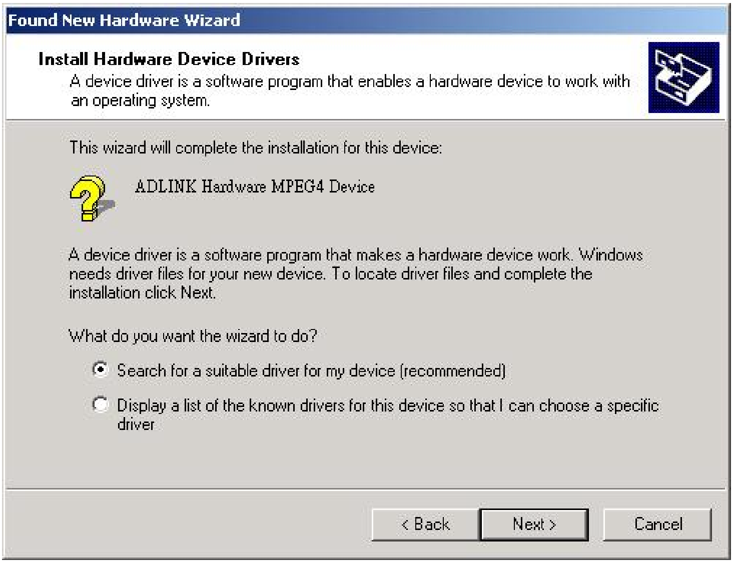
Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

6. At the **Install Hardware Device Drivers** dialog box, select **Search for a suitable driver for my device (recommended)**, click **Next** to continue.



Found New Hardware Wizard

Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

ADLINK Hardware MPEG4 Device

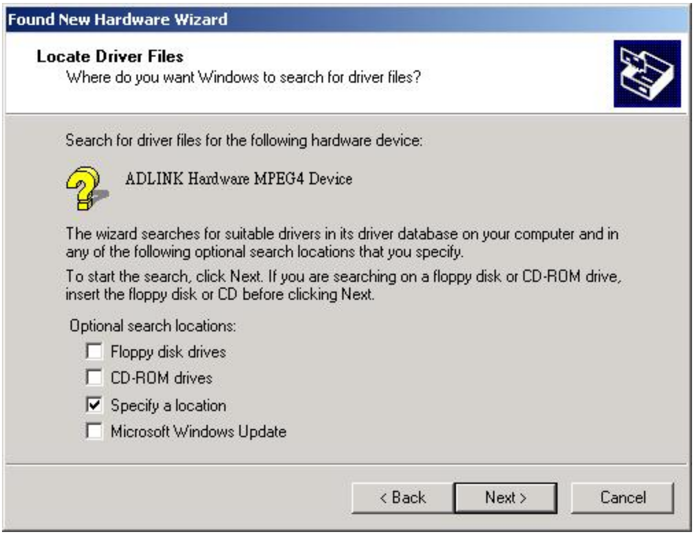
A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

● Search for a suitable driver for my device (recommended)

○ Display a list of the known drivers for this device so that I can choose a specific driver

7. In the **Locate Driver Files** dialog box, select **Specify a location** and click **Next** to continue.



Found New Hardware Wizard

Locate Driver Files

Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

ADLINK Hardware MPEG4 Device

The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

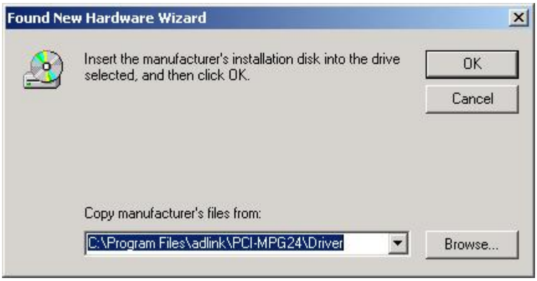
□ Floppy disk drives

□ CD-ROM drives

☑ Specify a location

□ Microsoft Windows Update

8. At the next dialog box, select the location of the PCI-MPG24 driver files and click **OK** to continue. The default installation folder for the PCI-MPG24 Driver is **C:\Program Files\ADLINK\PCI-MPG24\ Driver**.

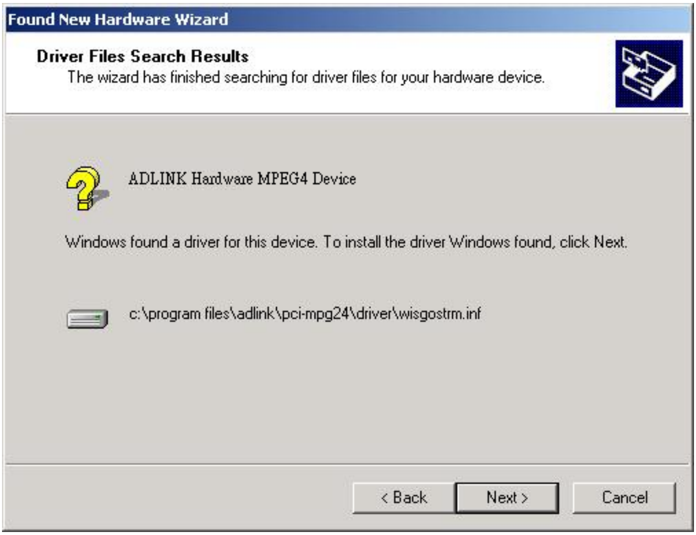


Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from:

C:\Program Files\adlink\PCI-MPG24\Driver

9. At the **Driver Files Search Results** dialog box, click **Next** to continue.



Driver Files Search Results

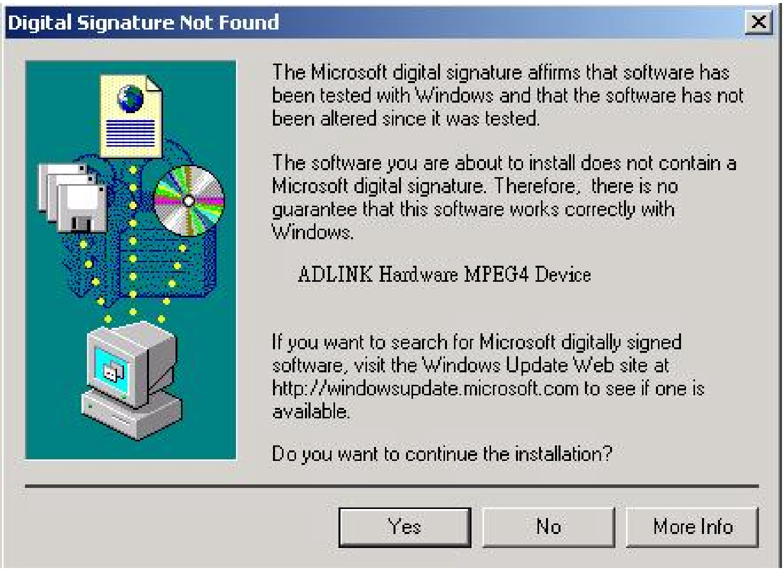
The wizard has finished searching for driver files for your hardware device.

ADLINK Hardware MPEG4 Device

Windows found a driver for this device. To install the driver Windows found, click Next.

c:\program files\adlink\pci-mpg24\driver\wisgostrm.inf

10. In the **Digital Signature Not Found** dialog box, click **Yes** to continue.



Digital Signature Not Found

The Microsoft digital signature affirms that software has been tested with Windows and that the software has not been altered since it was tested.

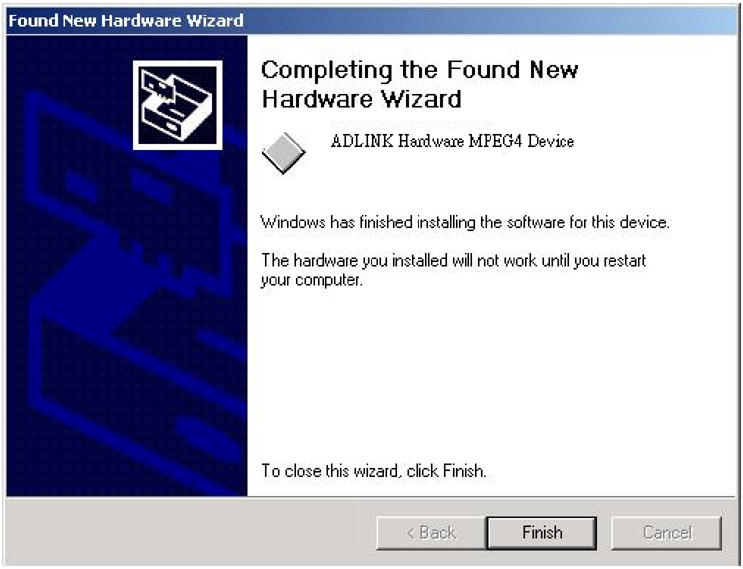
The software you are about to install does not contain a Microsoft digital signature. Therefore, there is no guarantee that this software works correctly with Windows.

ADLINK Hardware MPEG4 Device

If you want to search for Microsoft digitally signed software, visit the Windows Update Web site at http://windowsupdate.microsoft.com to see if one is available.

Do you want to continue the installation?

11. Click **Finish** in the **Found New Hardware Wizard**.



Found New Hardware Wizard

Completing the Found New Hardware Wizard

ADLINK Hardware MPEG4 Device

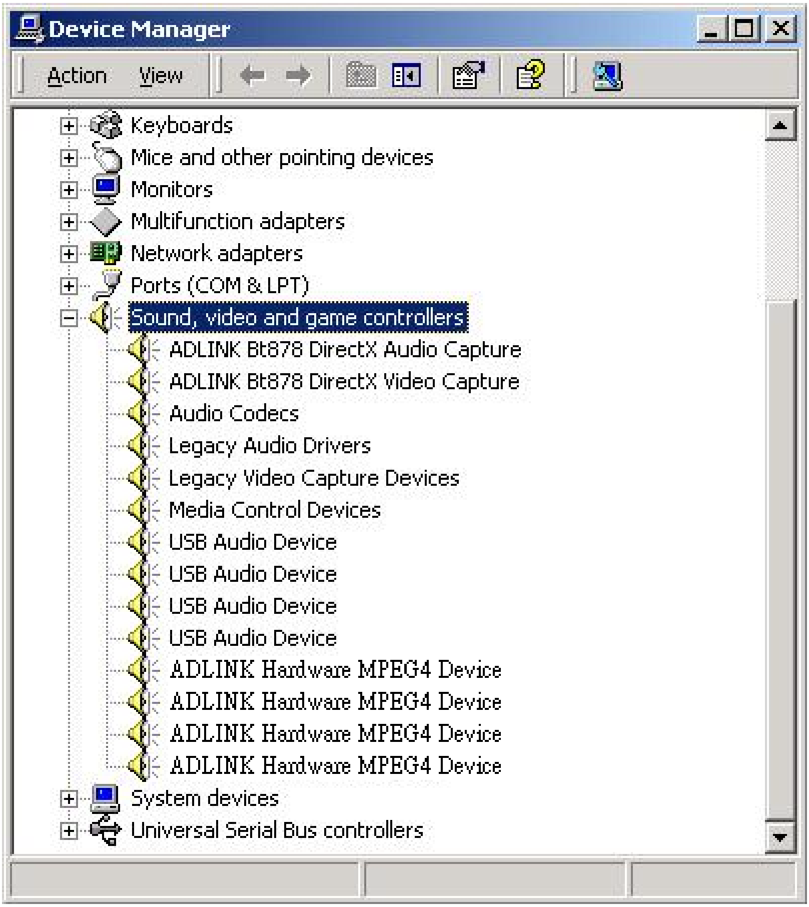
Windows has finished installing the software for this device.

The hardware you installed will not work until you restart your computer.

To close this wizard, click Finish.

12. For other types of new device, follow steps 2 to 11.

13. After installing all the devices, go to system control panel and select multimedia devices. There should be one **ADLINK Bt878 DirectX Audio Capture**, one **ADLINK Bt878 DirectX Video Capture**, four **USB Audio Devices**, four **ADLINK Hardware MPEG4 Devices**, and one **NetMos PCI Serial Port** as shown below.



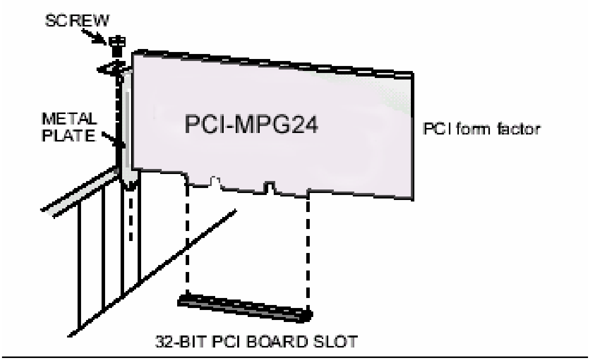
Device Manager

- Keyboards
- Mice and other pointing devices
- Monitors
- Multifunction adapters
- Network adapters
- Ports (COM & LPT)
- Sound, video and game controllers
  - ADLINK Bt878 DirectX Audio Capture
  - ADLINK Bt878 DirectX Video Capture
  - Audio Codecs
  - Legacy Audio Drivers
  - Legacy Video Capture Devices
  - Media Control Devices
  - USB Audio Device
  - USB Audio Device
  - USB Audio Device
  - USB Audio Device
  - ADLINK Hardware MPEG4 Device
  - ADLINK Hardware MPEG4 Device
  - ADLINK Hardware MPEG4 Device
  - ADLINK Hardware MPEG4 Device
- System devices
- Universal Serial Bus controllers

3.2 Hardware Installation

To install the PCI-MPG24 board onto the PCI bus:

1. Remove the computer cover using instructions from the computer manual.
2. Check that there is an empty PCI (32-bit) slot to accommodate the board. If there are no empty slots, remove a PCI board from the computer to make room for the PCI-MPG24 board and note down the chosen slot number.
3. Remove the blank metal plate located at the back of the selected slot (if any). Keep the removed screw to fasten the PCI-MPG24 board after installation.
4. Carefully position the PCI-MPG24 in the selected PCI slot as illustrated below. If using a tower computer, orient the board to accomodate the board slots.



5. Once aligned with an empty slot, press the board firmly but carefully into the connector.
6. Anchor the board by replacing the screw.
7. Connect your video sources for image acquisition tests. For details, refer to the ‘ViewCreator” utility.

4 ViewCreator Utility

Once hardware installation is complete, ensure that they are configured correctly before running the ViewCreator utility. This chapter outlines how to establish a vision system and how to manually control PCI_MPG24 cards to verify correct operation. ViewCreator provides a simple yet powerful means to setup, configure, test, and debug the vision system.

Note: ViewCreator is only available for Windows 2000/XP with a recommended screen resolution higher than 800x600 and 24-bit above color format. It also needs Microsoft Direct X runtime.

4.1 Overview

ViewCreator offers the following features:

1. 32-bit operation under Windows 2000/XP
2. PCI-MPG24 cards access
3. Video format adjustments
4. Video recording
5. Video file playback
6. Still image file saving (BMP)
7. Direct access to general purpose I/Os
8. Direct access to EEPROM
9. FULL, CIF, or QCIF Image size, 2x2 or 4x4 display

4.2 Component Description

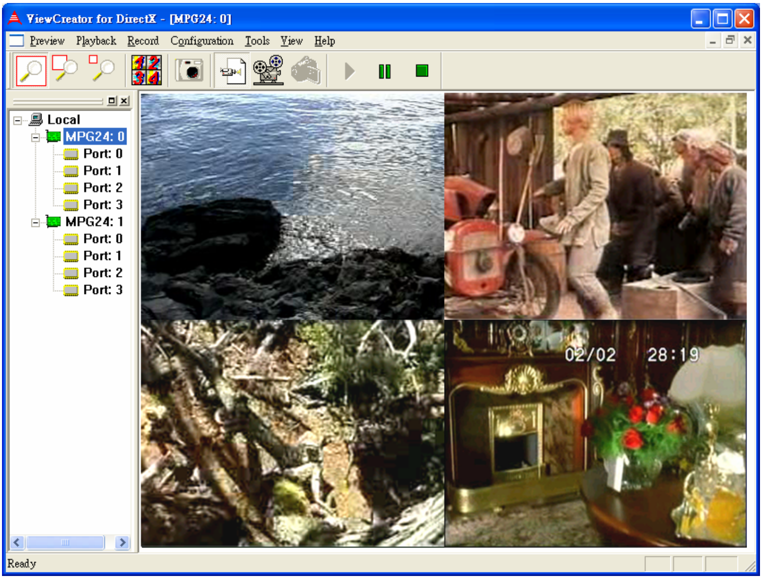


Figure 4-1: Component Description

Tree Browser

The Tree Browser window lists the PCI-MPG24 cards and video ports available at the local computer.

Image View

The Image View window displays Full, CIF, and QCIF size images and image effects. Playback is displayed in an individual window.

Toolbar

The toolbar simplify user’s operation. Full functions can be found on the menu.

4.3 Operation Theory

ViewCreator provides many functions for the PCI-MPG24 card as described below:

Preview

Single channel display

Click on the **video Port** icon in the Tree Browser window. A video frame will appear in the Image View window.

2x2 channels (Quad mode)

Click on the **card icon** in the Tree Browser window. All video ports in that card will appear in the Image View window.

You also can click **Single/Quad Image** on the toolbar to toggle between single channel display and 2x2 channels display.

All channels

Click on the **Local** icon in the Tree Browser window. All video ports in the system will appear in the Image View window.

Save still image

Click on **Capture Still Image** on the toolbar. This command saves the image into a bitmap format file. The path of the file can be set in the Set Still Image command of menu Preview.

This command can only be used in single channel display mode or 2x2 channels mode (select a video Port icon in the Tree Browser window).

Playback

Click **Play file** in the menu Playback. This command will open an **Open file** dialog. Select a media file to play.

This command can only be used in single channel display mode or 2x2 channels mode (select a video Port icon in the Tree Browser window).

Record

Select a video Port icon in the Tree Browser window. Click **Record Mode** on the toolbar then **Play** on the toolbar to capture video stream to a file. The path of the file can be set in the Set Recording command of menu Record.

Configurations

Record

Execute the Record Filter command of menu Configuration to open a setting menu. Click **OK** or **Apply** to apply these settings.

This command can only be used in the single channel display mode (select a video Port icon in the Tree Browser window).

Preview

Execute the Preview Filter command of menu Configuration to open a setting menu. There are two tabs allowing the user to select video standard and adjust the video amplifier. The changes will apply to the Image View window immediately.

Supported video standards are NTSC and PAL. Supported adjust video amplifiers are: brightness, contrast, hue, saturation, sharpness, gamma, white balance, black light compensation, and color enable.

This command can only be used in a single channel display mode or 2x2 channels mode (select a video Port icon in the Tree Browser window).

Tools

GPIO

Execute the GPIO command in the menu Tools to bring up the GPIO dialog box. Select the port to access and select the digital output value. Click the write or read button to write/read to/ from the digital I/O ports.

This command can only be used in the single channel display mode or in 2x2 channels mode (select a video Port icon in the Tree Browser window).

EEPROM

Execute EEPROM command in the menu Tools to bring up the EEPROM dialog box. Enter the offset and output values, and then click the Write button to write the value into the EEPROM. Enter the offset value and click the Read button to read the value from the EEPROM.

Valid offset values are between 0-127. Valid output values are in the range of 0 and 255. The value in the EEPROM will not be erased when the system is powered off.

This command can only be used in the single channel display mode or 2x2 channels mode (select a video Port icon in the Tree Browser window).

OSD

On Screen Display is supported by the ADLINK Hardware MPEG4 Device. It is used to overlay and OSD string with video, which will generate a corresponding OSD effect.

Execute an OSD command in the menu Tools to bring up the OSD dialog box. Enter the values of X and Y as the macro-block coordinate (X,Y) for the first font of display text, and enter display text in the Text textbox.

The maximum display text is limited to 94 characters. The size of a character is 16x16.

This command can only be used in single channel display mode and the duration of file recoding.

5 Programming Guide

5.1 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 5.2, along with a detailed description for each filter and its pins. Example filter graphs for previewing/capturing MPEG4 streams are illustrated in section 5.3. Section 5.4 provides examples of two ways of controlling the device driver.

5.2 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 5-1: 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 Reference Manual chapter 6.2: 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 Reference Manual chapter 6.2: Media Types)
Merit	MERIT_DO_NOT_USE

Table 5-2: 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 5-3: 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 5-4: 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 5-5: 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 5-6: CrossBar Filter

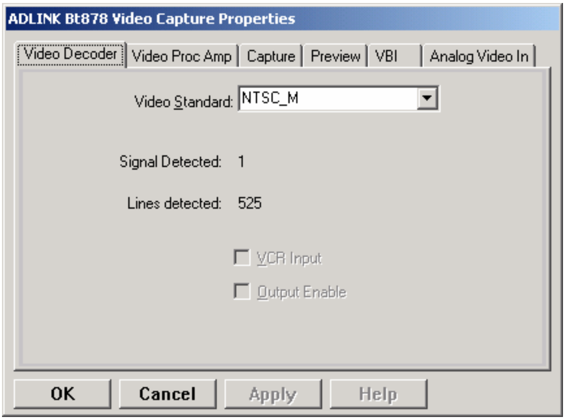
5.3 Example Graphs

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 Standard: NTSC_M

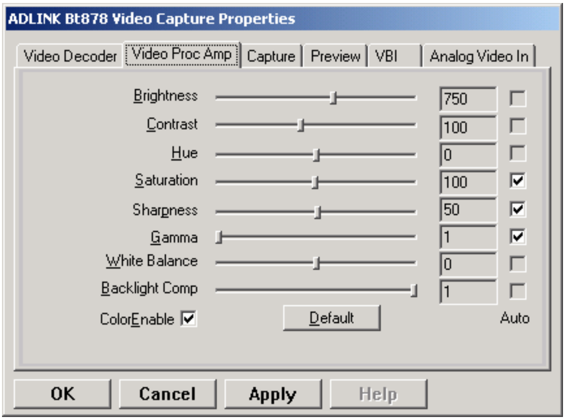
Signal Detected: 1

Lines detected: 525

□ VCR Input

□ Output Enable

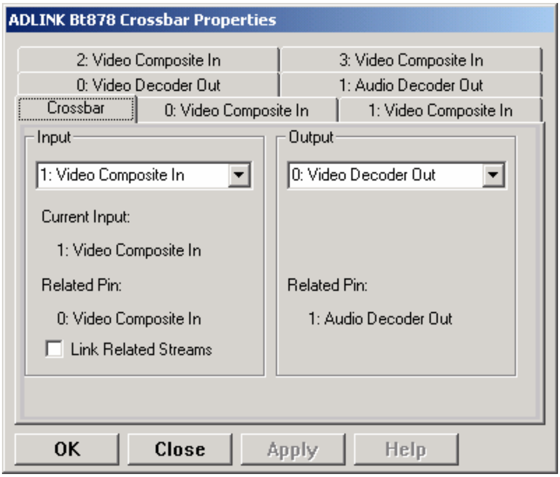
Video Proc Amp:



		Auto
Brightness	750	□
Contrast	100	□
Hue	0	□
Saturation	100	☑
Sharpness	50	☑
Gamma	1	☑
White Balance	0	□
Backlight Comp	1	□

ColorEnable ☑

ADLink Bt878 Crossbar filter


ADLINK Bt878 Crossbar Properties

Input: 1: Video Composite In

Current Input:

1: Video Composite In

Related Pin:

0: Video Composite In

□ Link Related Streams

Output: 0: Video Decoder Out

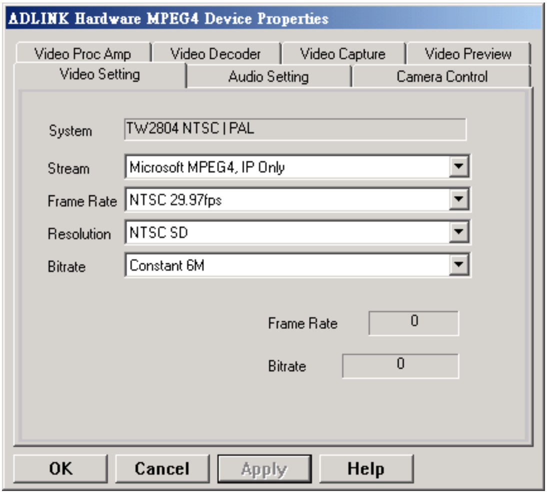
Related Pin:

1: Audio Decoder Out

Select video input before or during video previewing.

ADLINK Hardware MPEG4 Device

Video Setting:


ADLINK Hardware MPEG4 Device Properties

System: TW2804 NTSC | PAL

Stream: Microsoft MPEG4, IP Only

Frame Rate: NTSC 29.97fps

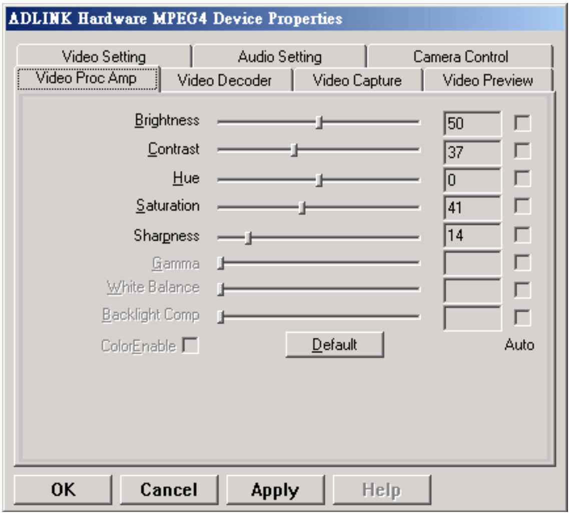
Resolution: NTSC SD

Bitrate: Constant 6M

Frame Rate: 0

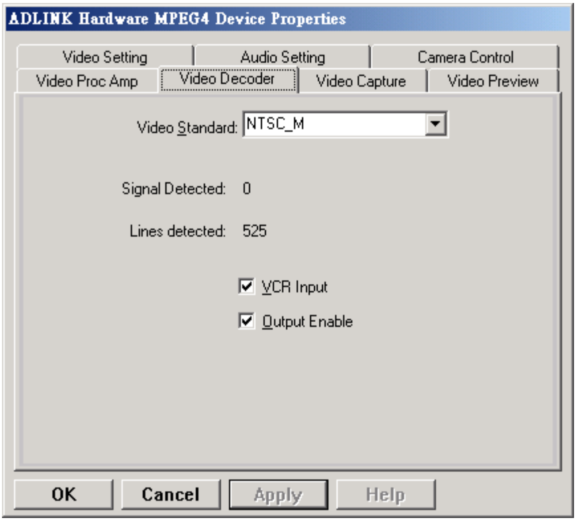
Bitrate: 0

Video Proc Amp:



Brightness 50 □
Contrast 37 □
Hue 0 □
Saturation 41 □
Sharpness 14 □
Gamma □
White Balance □
Backlight Comp □
ColorEnable □

Video Decoder



Video Standard: NTSC_M

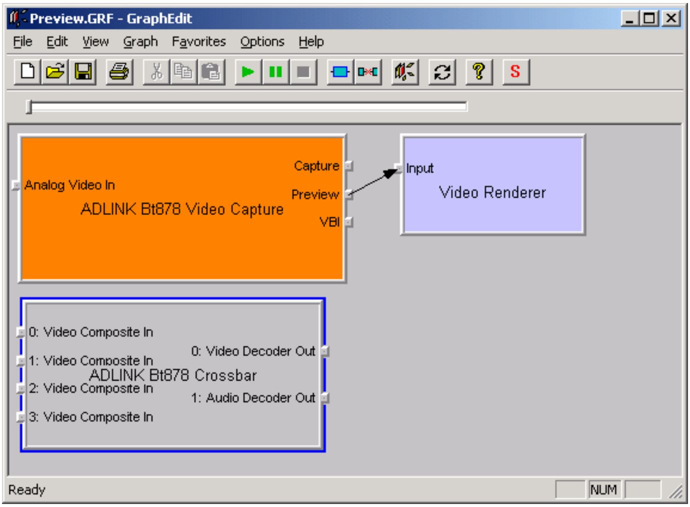
Signal Detected: 0

Lines detected: 525

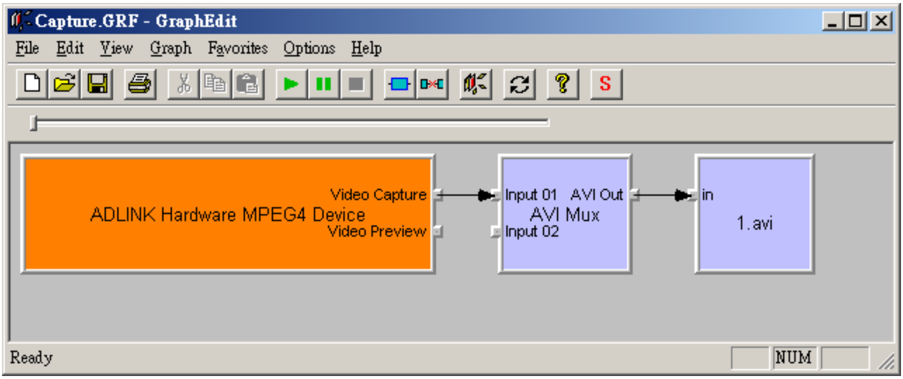
☑ VCR Input

☑ Output Enable

Preview



Capture



5.4 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:

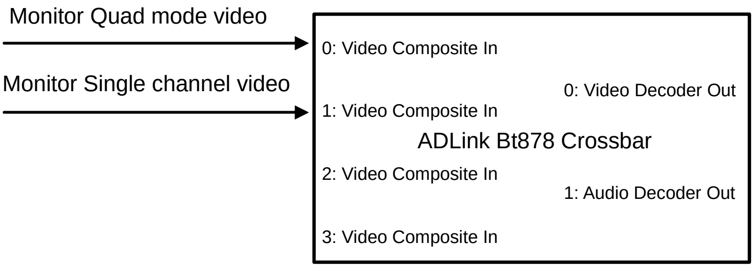


Figure 5-1: ADLink Bt878 Crossbar

For single video port selection please refer to the Bt878 GPIO pin definition in chapter 6.1.

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 Reference Manual.

6 Reference Manual

6.1 Preview

GPIO Access

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

Sample:

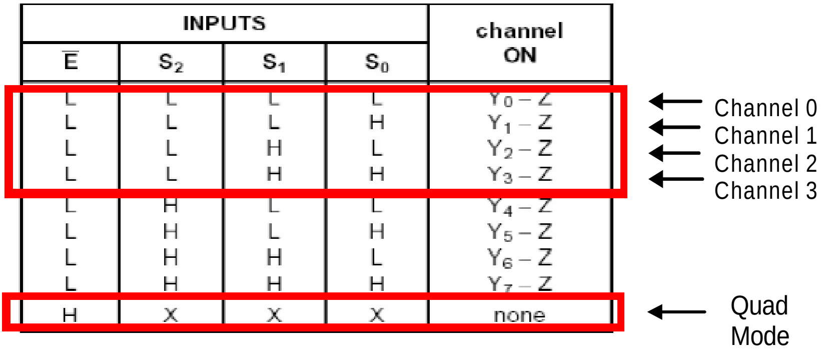
```
#define INSTANCE_DATA_OF_PROPERTY_PTR(x) (
  (PKSPROPERTY ((x)) ) + 1 )
#define INSTANCE_DATA_OF_PROPERTY_SIZE(x) (
  sizeof((x)) – sizeof (KSPROPERTY) )


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 6-1: 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

Channel 0
Channel 1
Channel 2
Channel 3
Quad
Mode

Table 6-2: 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];
}
```

6.2 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 6-3: 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 6-4: 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 6-5: MICROSOFT_MPEG4

MPEG2

Major type	MEDIATYPE_Video
Subtype	MEDIASUBTYPE_MPEG2_VIDEO
Format Type	FORMAT_MPEG2Video

Table 6-6: MPEG2

MPEG1

Major type	MEDIATYPE_Video
Subtype	MEDIASUBTYPE_MPEG1Payload
Format Type	FORMAT_MPEGVideo

Table 6-7: 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 6-8: 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 6-9: 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
▶ XVID_MPEG4:	3
▶ RCC_MPEG4:	4

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 indicating 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 resolution 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,
XVID_MPEG4= 3,
RCC_MPEG4= 4
};
```

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 GetStreamCaps.

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 TVIDEOCFGEX 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^{2}C register.

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

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^{2}C register.

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

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^{2}C 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

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^{2}C 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 6-10: 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 6-11: 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

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.

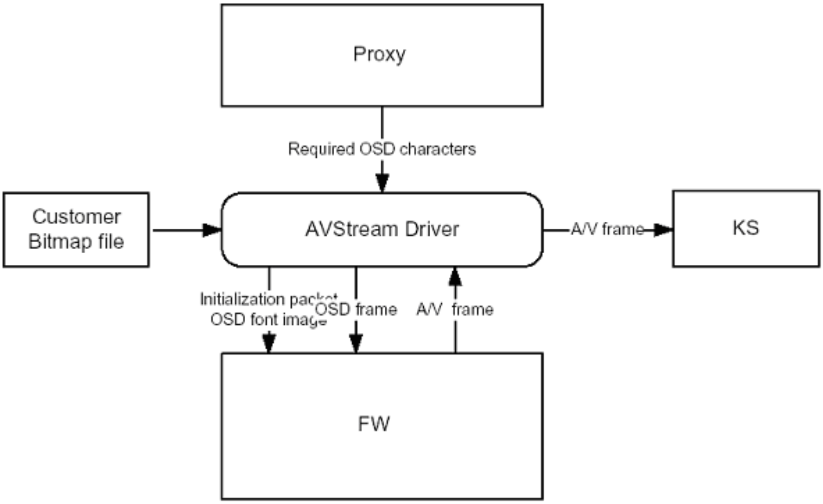
Alternatively, please visit http://msdn.microsoft.com/library/ and from the left panel navigation, select Graphics and Multimedia -> DirectX -> SDK Documentation -> DirectX 9.0 (C++) -> DirectShow -> DirectShow Reference -> Interfaces for a complete list of standard DirectShow filter interfaces references.

6.3 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

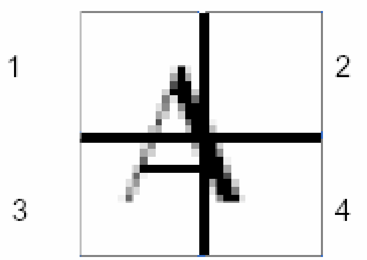
$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

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;
```

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 two-year guarantee:

   - The warranty period starts from the product's shipment date 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 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 two-year 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 technicians.
   - Products with altered and/or damaged serial numbers are not entitled to our service.
   - 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.