National Semiconductor

August 2001
Revision 1.04

Geode™ SC1200/SC1210 IAOC Devices: Linux Middleware API Programmer's Reference

Introduction

This specification describes the National Semiconductor® Linux Middleware API (application programming interface) for platforms based on National's Geode™ SC1200 and SC1210 Information on Chip (IAOC) devices. In particular, the National API covers analog and digital video focusing on multimedia applications. The API works in conjunction with standard audio, graphics, and networking interfaces currently built into the Linux operating system.

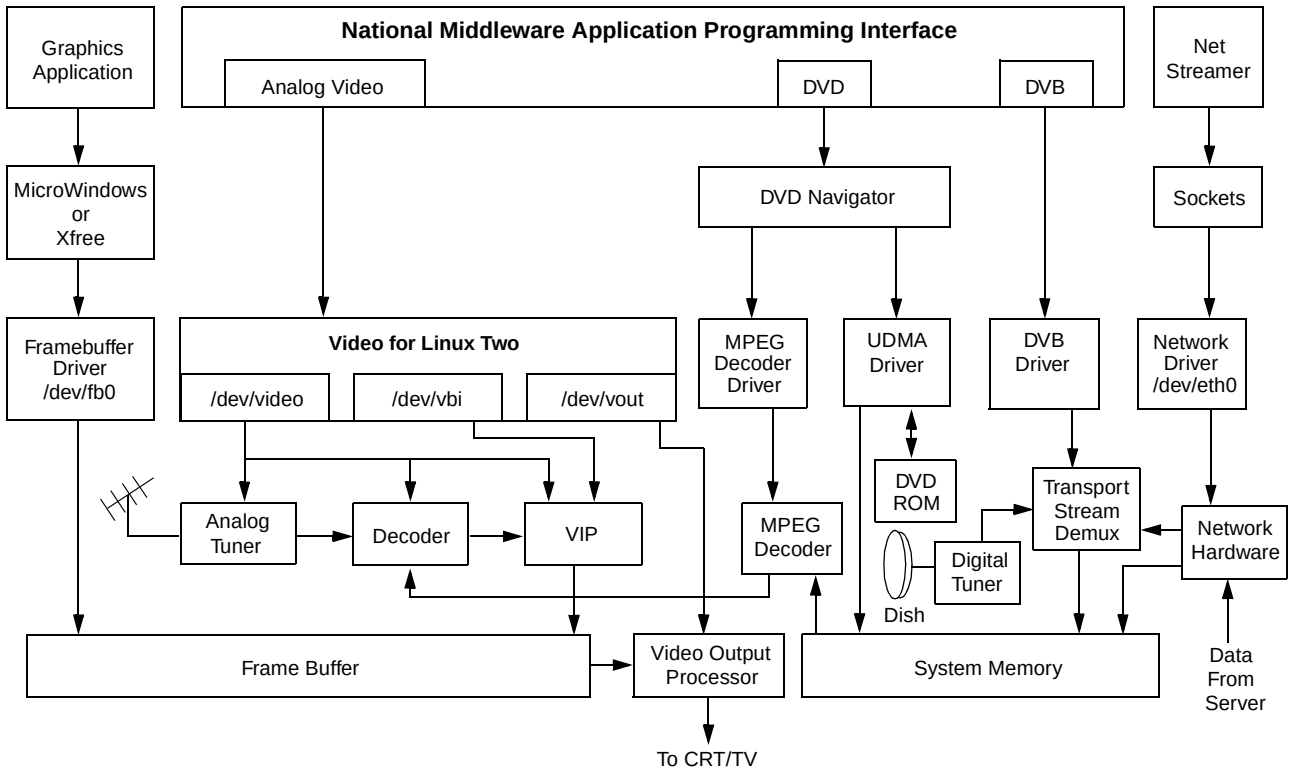
General Description

Linux has grown from a simple Internet community project to a widely used operating system for servers, desktop, and more recently in embedded applications. Linux is very similar to the standard UNIX distributions, and has been primarily used in server applications, thus leaving little support for high-end graphics, audio, or other multimedia applications.

In an effort to improve multimedia support under Linux, National Semiconductor built middleware layer. It leverages from much of the existing multimedia support built into standard Linux distributions and focuses on the missing or less mature areas of multimedia under Linux, such as video overlay, DVD, DVB, and additional analog TV support.

The figure illustrates the software architecture of the National middleware layer. The middleware layer primarily focuses on video applications and overlay support. For audio and networking, a programmer should use the support built into the Linux distributions such as, the Advanced Linux Sound Architecture (ALSA) for audio support, and the standard Berkeley Sockets interface available for network programming. For graphics, the application programmer can use the X-windows library, Microwindows or the Simple DirectMedia layer. The choice of graphics support is up to the application developer.

Middleware Layer Architecture



Software Overview

As previously stated, the middleware layer primarily focuses on video applications and overlay support. The video support can be grouped into two types, standard analog video and digital video.

Analog Video Subsystem

The Analog Video subsystem APIs provide a consistent interface to analog and digital broadcast services so that applications developed for one service can be extended easily to work with other services. The API also provides functionality that lets end users control interactive TV sets in familiar ways, such as changing the channels and adjusting volume.

Video capture APIs enable applications to capture single or multiple frames of live video in real-time and save them to a pixel map in graphics memory. Applications can then manipulate the captured images using the functionality of the graphics subsystem or other image-processing facilities.

Digital Video Subsystem

The Digital Video subsystem provides support for MPEG streams from interfaces such as DVDs, DVB, or MPEG streaming from a network. MPEG transport APIs provide traffic management of MPEG transport streams on digital set-tops. MPEG transport streams are multi-program data streams of interleaved digital video, audio, and data. These APIs receive MPEG transport streams containing encoded content from the network, and extract the desired program from them. They then separate the audio, video, and data components and route them to the audio decoder, video decoder, and CPU RAM respectively.

Overlay Subsystem

The Overlay subsystem API is used to do hardware overlay of video data on graphics data. This is mainly used to display video coming to the frame buffer through the Video Input Port (VIP) in the SC1200/SC1210. The input to the VIP can be from the TV tuner, or an MPEG decoder. Along with overlay, this subsystem can also do alpha blending of video data with graphics data.

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

1.0 Analog Video Subsystem

In the Analog Video subsystem, there are two data paths. One is the Streaming Play path where the data goes from the decoder (tuner, S-video, composite, or CCIR656) to the display directly. The other is the Capture path where the data is captured by the application in to the hard disk.

In the Streaming Play path, the data comes from the selected input into the decoder where it is captured frame-by-frame and passed on to the VIP (Video Input Port) which stores this data in to the frame buffer or graphics off-screen memory.

Based on the application's requirement, the frame buffer driver programs the VIP hardware to place the data at an appropriate address in off-screen memory. The application also makes calls to the frame buffer driver to program the video output processor to overlay this video data on top of graphics at the required screen coordinates and required size.

If the incoming data is smaller in size compared to the destination display size, then it is up-scaled by the video output processor. If the screen display size is smaller than the incoming data, then it is downscaled to the required size by programming the decoder through the analog video driver.

For video capture, the application does a read call to the video driver and the video driver reads a frame of data from the frame buffer and passes that to the application through a buffer. The application can also read VBI data in the same way. This data can then be stored by the application into a storage device such as files.

The implementation of the National Analog Video Middleware API is based on the Video for Linux Two (V4L2) specification. All API functions are implemented by using calls to the National V4L2 driver.

Figure 1-1 depicts the software architecture for the Analog Video Middleware API.

The V4L2 driver does not access hardware directly. All access to hardware from V4L2 goes through two libraries. Video Library is for all Geode hardware-specific access like overlay, alpha blend, and VIP. TV library is for non-Geode hardware access such as analog TV tuner, Video decoder, and VBI.

The Analog Video middleware API accesses the V4L2 drivers through a thin driver layer that is a part of the main API implementation layer. This layer abstracts the kernel drivers and the middleware API layer so that when an OEM wants to port the middleware API to their own hardware, they can modify this thin driver to work with their kernel drivers without touching the main API layer.

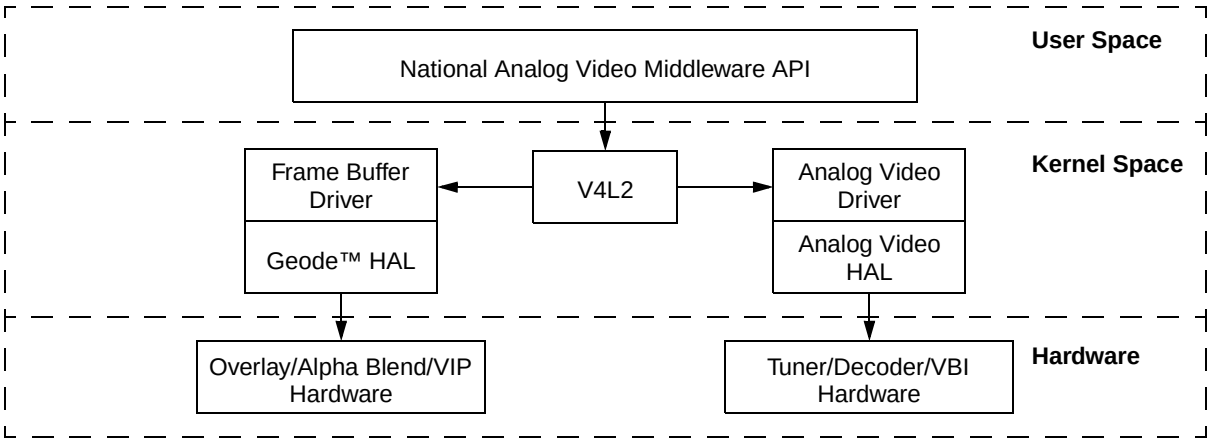


Figure 1-1. Analog Video Middleware API Implementation Block Diagram

Analog Video Subsystem (Continued)

1.1 ANALOG VIDEO SUBSYSTEM FUNCTIONS

1.1.1 VMW_AVS_VideoInit

Description:

Initialize the analog video subsystem. This function should be called before any other function in the API.

Syntax:

U32 VMW_AVS_VideoInit()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If video is already opened.
VMW_AVSE_INIT	If memory is not allocated.
VMW_AVSE_OPEN	Could not open any device.

Comments:

This function opens all the required device files and initializes them if necessary. A global device context is initialized.

Also see:

Section 1.1.13 "VMW_AVS_VideoClose" on page 13.

1.1.2 VMW_AVS_GetVideoCapabilities

Description:

Retrieve the capabilities of the video device.

Syntax:

U32 VMW_AVS_GetVideoCapabilities(VIDEO_CAPABILITY * pVidCap)

Parameters:

Parameter	Description
pVidCap	Pointer to video capability structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If video is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Analog Video Subsystem (Continued)

1.1.3 VMW_AVS_EnumFormat

Description:

Retrieve packing information about the supported capture formats. Frame buffer formats can be queried as well as pixel formats.

Syntax:

U32 VMW_AVS_EnumFormat(int type, VIDEO_FMTDESC * pFmtDesc)

Parameters:

Parameter	Description
type	ENUM_PIXEL_FMT or ENUM_FRBUFF_FMT.
pFmtDesc	Pointer to VIDEO_FMTDESC structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

An application can query the list of supported capture formats with this call. The application fills in the index field of a VIDEO_FMTDESC structure, and passes it to this function, which fills in the rest of the fields. The application should use index values from 0 onwards, until the call fails. The type field indicates whether pixel or frame buffer format information is required.

Also see:

None

1.1.4 VMW_AVS_Read

Description:

Read the video buffer. This function is supported only if the V4L2_FLAG_READ flag is set in the VIDEO_CAPABILITY structure. Each call to this function fills the buffer with a new frame. The driver may fail if the length of the buffer is less than the required buffer size. The required size is retrieved by calling the GetCaptureFormat function. (The sizeimage field in the VIDEO_FORMAT structure.)

Syntax:

U32 VMW_AVS_Read(char * pBuffer, U32 count)

Parameters:

Parameter	Description
pBuffer	Pointer to buffer of adequate size.
count	Length of buffer in bytes.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Also see:

None

Analog Video Subsystem (Continued)

1.1.5 VMW_AVS_EnumVideoInput

Description:

Retrieve the properties of a video input in to a VIDEO_INPUT structure.

Syntax:

U32 VMW_AVS_EnumVideoInput(VIDEO_INPUT * pVidInput)

Parameters:

Parameter	Description
pVidInput	Pointer to VIDEO_INPUT structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

Before calling this function, the caller should fill in the index field to indicate which input is being queried.

Also see:

Section 1.1.6 "VMW_AVS_GetCurrentVideoInput" on page 9 and Section 1.1.7 "VMW_AVS_SetCurrentVideoInput" on page 10.

1.1.6 VMW_AVS_GetCurrentVideoInput

Description:

Retrieve the index of the currently selected video input.

Syntax:

U32 VMW_AVS_GetCurrentVideoInput(U32 * pIndex)

Parameters:

Parameter	Description
pIndex	Pointer to index of currently selected input.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

The index which has been returned can be used to query the input using the VMW_AVS_EnumVideoInput function.

Also see:

Section 1.1.7 "VMW_AVS_SetCurrentVideoInput" on page 10.

Analog Video Subsystem (Continued)

1.1.7 VMW_AVS_SetCurrentVideoInput

Description:

Set the index of the video input to be used.

Syntax:

U32 VMW_AVS_SetCurrentVideoInput(U32 * pIndex)

Parameters:

Parameter	Description
pIndex	Pointer to index of input.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 1.1.6 "VMW_AVS_GetCurrentVideoInput" on page 9.

1.1.8 VMW_AVS_SetVideoWindow

Description:

Set the size of the video window.

Syntax:

U32 VMW_AVS_SetVideoWindow(int x, int y, int width, int height)

Parameters:

Parameter	Description
x	X coordinate of the window in pixels.
y	Y coordinate of the window in pixels.
Width	Width of the window in pixels.
Height	Height of the window in pixel.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

This function sets the size of the displayed or captured video. At the driver level, this function will program the decoder chip to scale the incoming video date as required.

Also see:

None

Analog Video Subsystem (Continued)

1.1.9 VMW_AVS_EnumVideoStandards

Description:

Query video standards. This function enables the programmer to query the different video standards supported by the video sub-system. The programmer must fill the index field of the ENUM_STANDARD structure (see Section 1.6.10 on page 27) and call the function. The ENUM_STANDARD structure will be filled up by the function. The index must start from zero, incrementing by one. The call will fail when there are no more standards to query.

Syntax:

U32 VMW_AVS_EnumVideoStandards(ENUM_STANDARD * pEnumStd)

Parameters:

Parameter	Description
pEnumStd	Pointer to ENUM_STANDARD structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 1.1.10 "VMW_AVS_GetVideoStandard" on page 11.
Section 1.1.11 "VMW_AVS_SetVideoStandard" on page 12.

1.1.10 VMW_AVS_GetVideoStandard

Description:

Retrieve current video standard.

Syntax:

U32 VMW_AVS_GetVideoStandard(VIDEO_STANDARD * pVidStd)

Parameters:

Parameter	Description
pVidStd	Pointer to VIDEO_STANDARD structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 1.6.11 "VIDEO_STANDARD" on page 27.

Analog Video Subsystem (Continued)

1.1.11 VMW_AVS_SetVideoStandard

Description:

Set current video standard. The user must fill in the correct standard in a *VIDEO_STANDARD* structure and call the function.

Syntax:

U32 VMW_AVS_SetVideoStandard(VIDEO_STANDARD * pVidStd)

Parameters:

Parameter	Description
pVidStd	Pointer to VIDEO_STANDARD structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

1.1.12 VMW_AVS_SetControl

Description:

Change the value of a setting.

Syntax:

U32 VMW_AVS_SetControl(U32 type, int value)

Parameters:

Parameter	Description
type	Control that needs to be changed: AVS_CONTROL_BRIGHT AVS_CONTROL_HUE AVS_CONTROL_SAT
value	The value of the control.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Analog Video Subsystem (Continued)

1.1.13 VMW_AVS_VideoClose

Description:

Shutdown the video subsystem. This function shuts down the video subsystem by closing all open device file descriptors.

Syntax:

Void VMW_AVS_VideoClose()

Parameters:

None

Returns:

None

Comments:

This call will not fail.

Also see:

Section 1.1.1 "VMW_AVS_VideoInit" on page 7.

1.1.14 VMW_AVS_EnableDVIP

Description:

Enable or disable direct video capability.

Syntax:

U32 VMW_AVS_EnableDVIP(int enable)

Parameters:

Parameter	Description
Enable	Flag to specify enable/disable option.

Returns:

Return	Description
Zero	On success.
Non-zero error code	On failure.

Comments:

None.

Also see:

None

Analog Video Subsystem (Continued)

1.2 ANALOG VIDEO CAPTURE (AVC) FUNCTIONS

1.2.1 VMW_AVC_GetCaptureFormat

Description:

Retrieve format of the captured video image.

Syntax:

U32 VMW_AVC_GetCaptureFormat(VIDEO_FMTDESC * pFmtDesc)

Parameters:

Parameter	Description
pFmtDesc	Pointer to VIDEO_FMTDESC structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 1.6.4 "VIDEO_FMTDESC" on page 22.

1.2.2 VMW_AVC_SetCaptureFormat

Description:

Set captured image format.

Syntax:

U32 VMW_AVC_SetCaptureFormat(VIDEO_FMTDESC * pFmtDesc)

Parameters:

Parameter	Description
pFmtDesc	Pointer to VIDEO_FMTDESC structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Also see:

None

Analog Video Subsystem (Continued)

1.3 ANALOG VIDEO TUNER (AVT) FUNCTIONS

1.3.1 VMW_AVT_GetTunerCapabilities

Description:

Retrieve capability of the tuner in the current input, if present.

Syntax:

U32 VMW_AVC_GetTunerCapabilities(VIDEO_TUNER* pVidTuner)

Parameters:

Parameter	Description
pVidTuner	Pointer to VIDEO_TUNER structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Description:

Set tuner capabilities.

Syntax:

U32 VMW_AVT_SetTunerCapabilities(VIDEO_TUNER * pVidTuner)

Parameters:

Parameter	Description
pVidTuner	Pointer to VIDEO_TUNER structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Also see:

None

Analog Video Subsystem (Continued)

1.3.2 VMW_AVT_GetTunerFrequency

Description:

Retrieve the frequency to which the tuner is currently tuned.

Syntax:

U32 VMW_AVT_GetTunerFrequency(U32 * pFrequency)

Parameters:

Parameter	Description
pFrequency	Pointer to U32.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

Here, the word “Frequency” is a misnomer. It actually specifies a channel number. The channel number to frequency translation and vice-versa is done by the driver.

Also see:

None

1.3.3 VMW_AVT_SetTunerFrequency

Description:

Set the frequency to which the tuner is currently tuned.

Syntax:

U32 VMW_AVT_SetTunerFrequency(U32 * pFrequency)

Parameters:

Parameter	Description
pFrequency	Pointer to U32 (new frequency value).

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

Here, the word “Frequency” is a misnomer. It actually specifies a channel number. The channel number to frequency translation and vice-versa is done by the driver.

Also see:

None

Analog Video Subsystem (Continued)

1.4 ANALOG VIDEO – AUDIO (AVA) FUNCTIONS

1.4.1 VMW_AVA_GetAudioCapabilities

Description:

Retrieve information about the audio portion of the video device if available.

Syntax:

U32 VMW_AVA_GetAudioCapabilities(VIDEO_AUDIO * pVidAudio)

Parameters:

Parameter	Description
pVidAudio	Pointer to VIDEO_AUDIO structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

Only the tuner source has this feature.

Also see:

None

1.4.2 VMW_AVA_SetAudioCapabilities

Description:

Set information about the audio portion of the video device if available.

Syntax:

U32 VMW_AVA_SetAudioCapabilities(VIDEO_AUDIO * pVidAudio)

Parameters:

Parameter	Description
pVidAudio	Pointer to VIDEO_AUDIO structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

Only the tuner source will have this feature.

Also see:

None

Analog Video Subsystem (Continued)

1.5 ANALOG VIDEO VBI (AVV) FUNCTIONS

1.5.1 VMW_AVV_GetVBICapabilities

Description:

Retrieve VBI capabilities of the current video input.

Syntax:

U32 VMW_AVV_GetVBICapabilities(VIDEO_CAPABILITY * pVidCap)

Parameters:

Parameter	Description
pVidCap	Pointer to VIDEO_CAPABILITY structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

1.5.2 VMW_AVV_GetVBIFormat

Description:

Retrieve VBI format of the current video input.

Syntax:

U32 VMW_AVV_GetVBIFormat(VIDEO_FORMAT * pVidFmt)

Parameters:

Parameter	Description
pVidFmt	Pointer to VIDEO_FORMAT structure

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Analog Video Subsystem (Continued)

1.5.3 VMW_AVV_SetVBIFormat

Description:

Set VBI format of the current video input.

Syntax:

U32 VMW_AVV_SetVBIFormat(VIDEO_FORMAT * pVidFmt)

Parameters:

Parameter	Description
pVidFmt	Pointer to VIDEO_FORMAT structure.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

This function is used to get raw VBI data or two bytes of closed-caption data.

Also see:

None

1.5.4 VMW_AVV_SetupVBICallback

Description:

Set up the callback function for VBI capture.

Syntax:

U32 VMW_AVV_SetupVBICallback(pVBICallback pCallBack, uchar * pBuffer,int length, void * pUserData)

Parameters:

Parameter	Description
callBack	Pointer to callback function. The callback function returns void and takes two parameters, the first is an integer and the second a void * pointer. The first parameter tells the function how many bytes are ready to be transferred and the second is a user definable pointer.
buffer	The buffer where the VBI data will be captured. This is allocated by the user.
length	The length of the allocated buffer.
userData	The user can set this to point to any user defined data. The callback function is called with this as the second parameter.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

Actual capture starts only after the user calls VMW_AVV_EnableVBICapture.

Also see:

Section 1.5.5 "VMW_AVV_EnableVBICapture" on page 20.

Analog Video Subsystem (Continued)

1.5.5 VMW_AVV_EnableVBICapture

Description:

Enable or disable VBI capture.

Syntax:

U32 VMW_AVV_EnableVBICapture(int enable)

Parameters:

Parameter	Description
enable	Non-zero value enables capture and zero disables capture.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

The VBI format and callback should be set.

Also see:

Section 1.5.4 "VMW_AVV_SetupVBICallback" on page 19.

1.6 ANALOG VIDEO STRUCTURES

The Video4Linux2 structures are named with the prefix “v4l2”. In the National API, these structures are type defined (typedef) for ease of using.

1.6.1 VIDEO_FORMAT

V4L2 Name: v4l2_format

```
typedef struct {
        U32 type;
        union
        {
                struct v4l2_pix_format          pix;    /* image format (pix_format)*/
                struct v4l2_vbi_format          vbi;    /* vbi format (vbi_format)  */
                u8 raw_data[200];                       /* User defined data */
        } fmt;
} VIDEO_FORMAT;
```

Structure Member	Description
type	To indicate the type of data carried by the structure (VBI or pixel).
fmt	Union of different types of format structures. Currently, only v4l2_pix_format and v4l2_vbi_format are supported. The union also contains 200 bytes of used definable data.

Analog Video Subsystem (Continued)

1.6.2 PIX_FORMAT

V4L2 Name: v4l2_pix_format

```
typedef struct {
        U32 width;
        U32 height;
        U32 depth;
        U32 pixelformat;
        U32 flags;
        U32 bytesperline;
        U32 sizeimage;
        U32 priv;
} PIX_FORMAT;
```

Structure Member	Description
width	Width of the picture.
height	Height of the structure.
depth	Depth of the image (bits per pixel).
pixelformat	The way in which a pixel is represented V4L2_PIX_FMT_RGB565 For the other formats, please refer to videodev.h, available in /usr/include/linux.
flags	Flag to indicate special attributes of the image. V4L2_FMT_FLAG_BYTESPERLINE - Bytes per line field is valid.
bytesperline	The number of bytes per scan-line. This is used only when there are pad bytes.
sizeimage	The size of the image in bytes.
priv	Some private data which may be required. This will depend on the pixelformat.

1.6.3 VBI_FORMAT

V4L2 Name: v4l2_vbi_format

```
typedef struct {
        U32 samplingrate;          /* in Hz */
        U32 offset;
        U32 samplesperline;
        U32 sampleformat;          /* VIDEO_PALETTE_RAW only (1 byte) */
        S32 start[2];              /* starting line for each frame */
        U32 count[2];              /* count of lines for each frame */
        U32 flags;
} VBI_FORMAT;
```

Structure Member	Description
samplingrate	Samples per second (i.e., unit 1 Hz).
offset	Horizontal offset of the VBI image, relative to the rising edge of the (first) horizontal sync pulse and counted in samples. The first sample in the VBI image will be located offset / sampling_rate seconds following the rising edge.
samplesperline	Number of samples that will be taken per scanline of the VBI image.
sampleformat	Currently only one sample format is specified, V4L2_VBI_SF_UBYTE; where each sample consists of eight bits (an unsigned number), with lower values oriented towards the sync level. Do not assume any other correlation of values with the signal level. For example, the MSB does not necessarily indicate if the signal is 'high' or 'low' because 128 may not be the mean value of the signal.

Analog Video Subsystem (Continued)

Structure Member	Description
start	This is the scanning system line number associated with the first line of the VBI image, of the first and the second field in time respectively. Valid numbers for 625 line scanning systems (usually PAL/SECAM) are 0 to 312 for the first, 313 to 624 for the second field, inclusive. Valid numbers for 525 line systems (usually NTSC) are 0 to 262 and 263 to 524, respectively. To clarify, in this scheme the PAL VPS signal is located on lines 15/327, the NTSC CC signal ("LITO") on lines 20/283. It is assumed here that the first field in time is the odd aka top field of the frame, hence one field line longer. This may not apply to all scanning systems.
count	The number of lines in the first and second field image, respectively. Typically these two values are equal. However, hardware restrictions may require different start and count values for the two fields. Drivers should provide as much flexibility as possible. For example, it may be possible to extend or move the VBI capture window down to the picture area, implementing a 'full field mode' to capture data service transmissions embedded in the picture, or to access data services in the border regions of the picture, like WSS (wide screen signalling). For non-interlaced scanning systems like Progressive PAL, when the term 'field' does not apply, set count[1] to zero. start[1] is not valid then. Otherwise a count value of zero, or a value exceeding the valid scanning line numbers given above, or an image covering lines of different fields, is invalid. To initialize the start and count fields, applications must first determine the current video standard selection.
flags	V4L2_VBI_UNSYNC - This flag indicates hardware which does not properly distinct between fields. Normally, the first field transmitted, the first in memory, is the odd a.k.a. TOPFIELD. If this flag is set, the fields still appear in correct order in time, however, it is not apparent if the odd or even field appears first. This is fatal only for certain data services which change their meaning depending on the field number. V4L2_VBI_INTERLACED - By default, the two field images are transmitted sequentially. This is, all lines of the first field followed by all lines of the second field. If this flag is set, the two fields appear interlaced, the first line of the first field followed by the first line of the second field, then the two second lines, and so on. Such a layout may be necessary if the hardware has been programmed to capture interlaced video images and can not separate the fields for VBI at the same time. Setting this flag implies the two count values are equal.

1.6.4 VIDEO_FMTDESC

V4L2 Name: v4l2_fmtdesc

```
typedef struct {
        U32 index;
        U8 description[32];
        U32 pixelFormat;
        U32 flags;
        U32 depth;
        U32 reserved[2];
} VIDEO_FMTDESC;
```

Structure Member	Description
index	Format number.
description	Description string.
pixelformat	The way in which a pixel is represented can take several values; one of which is: V4L2_PIX_FMT_RGB565 For the other formats, please refer to videodev.h, available in /usr/include/linux.

Analog Video Subsystem (Continued)

Structure Member	Description
flags	Format description flags: V4L2_FMT_FLAG_COMPRESSED - 0x0001; Compressed format. V4L2_FMT_FLAG_BYTESPERLINE - 0x0002; Bytesperline field valid. V4L2_FMT_FLAG_NOT_INTERLACED - 0x0000. V4L2_FMT_FLAG_INTERLACED - 0x0004; Image is interlaced. V4L2_FMT_FLAG_TOPFIELD - 0x0008; Is a top field only. V4L2_FMT_FLAG_BOTFIELD - 0x0010; Is a bottom field only. V4L2_FMT_FLAG_ODDFIELD - V4L2_FMT_FLAG_TOPFIELD. V4L2_FMT_FLAG_EVENFIELD - V4L2_FMT_FLAG_BOTFIELD. V4L2_FMT_FLAG_COMBINED - V4L2_FMT_FLAG_INTERLACED. V4L2_FMT_FLAG_FIELD_field - 0x001C. V4L2_FMT_CS_FIELD - 0xF000; Color space field mask. V4L2_FMT_CS_601YUV - 0x1000; ITU YCrCb color space. V4L2_FMT_FLAG_SWCONVERSION - 0x0800; Used only in format enum. SWCONVERSION indicates the format is not natively supported by the driver and the driver uses software conversion to support it.
depth	Bits per pixel.
reserved	Reserved for future capabilities.

1.6.5 VIDEO_AUDIO

V4L2 Name: v4l2_audio

```
typedef struct {
        U32 audio;
        char name[32];
        U32 capability;
        U32 mode;
        U32 reserved[2];
} VIDEO_AUDIO;
```

Structure Member	Description
audio	Indicates that this is an audio structure.
name	Description string.
capability	Flags for the capability field: V4L2_AUDCAP_EFFECTS - 0x0020 V4L2_AUDCAP_LOUDNESS - 0x0040 V4L2_AUDCAP_AVL - 0x0080
mode	Flags for the mode field: #define V4L2_AUDMODE_LOUDNESS - 0x00002 #define V4L2_AUDMODE_AVL - 0x00004 #define V4L2_AUDMODE_STEREO_field - 0x0FF00 #define V4L2_AUDMODE_STEREO_LINEAR - 0x00100 #define V4L2_AUDMODE_STEREO_PSEUDO - 0x00200 #define V4L2_AUDMODE_STEREO_SPATIAL30 - 0x00300 #define V4L2_AUDMODE_STEREO_SPATIAL50 - 0x00400
reserved	Reserved for future capabilities.

Analog Video Subsystem (Continued)

1.6.6 VIDEO_CAPABILITY

V4L2 Name: v4l2_capability

```
typedef struct {
        char name[32];
        U32 type;
        U32 inputs;
        U32 outputs;
        U32 audios;      /* Num audio devices */
        U32 maxwidth;    /* Supported width */
        U32 maxheight;   /* And height */
        U32 minwidth;    /* Supported width */
        U32 minheight;   /* And height */
} VIDEO_CAPABILITY;
```

Structure Member	Description
name	Device name.
type	Type of device: V4L2_TYPE_CAPTURE - Capture device. V4L2_TYPE_CODEC - Codec device (has draft specification). V4L2_TYPE_OUTPUT - Video output device (i.e., not a graphics display). V4L2_TYPE_FX - An effects or video filter device. V4L2_TYPE_VTR - Video tape recorder controller device. V4L2_TYPE_VBI - VBI device. V4L2_TYPE_RADIO - Radio device (audio and tuning IOCTLs).
inputs	Number of video inputs that can be selected.
outputs	Number of video outputs that can be selected.
audios	Number of audio inputs that can be selected.
maxwidth	Best case maximum image width in pixels.
maxheight	Best case maximum image height in pixels.
minwidth	Minimum width in pixels.
minheight	Minimum height in pixels.
flags	Device capability flags: V4L2_FLAG_READ - Capable of capturing frames or data via read(). V4L2_FLAG_WRITE - Capable of accepting frames or data via write(). V4L2_FLAG_STREAMING - Capable of transferring frames or data asynchronously via pre-allocated buffers. V4L2_FLAG_PREVIEW - Supports automatic video preview. V4L2_FLAG_SELECT - Supports the select() call. V4L2_FLAG_TUNER - Has a tuner of some form. V4L2_FLAG_MONOCHROME - Image capture is grey scale only. V4L2_FLAG_DATA_SERVICE - Does teletext.
reserved	Reserved for future capabilities.

Analog Video Subsystem (Continued)

1.6.7 VIDEO_WINDOW

V4L2 name: v4l2_window

```
typedef struct {
        U32   x,y;              /* Position of window */
        U32   width,height;     /* Its size */
        U32   chromakey;
        VIDEO_CLIP *clips;      /* Set only */
        U32     clipcount;
} VIDEO_WINDOW;
```

Structure Member	Description
x	X coordinate of video window position.
y	Y coordinate of video window position.
width	Width of video window.
height	Height of video window.
chromakey	Unused.
clips	Pointer to the next VIDEO_CLIP structure (used to create a singly linked list). This is not currently used in the National API.
clipcount	Not applicable in the National API.

1.6.8 VIDEO_INPUT

V4L2 name: v4l2_input

```
typedef struct  {
        U32 index;
        char name[32];
        U32 type;
        U32 capability;
        U32 assocaudio;
        U32 reserved[4]
} VIDEO_INPUT;
```

Structure Member	Description
index	The input to which these properties apply (set by the caller).
name	Friendly name of the input, preferably including the label on the input itself.
type	Type of input: V4L2_INPUT_TYPE_TUNER - Analog TV tuner input. V4L2_INPUT_TYPE_CAMERA - Analog base-band input. V4L2_INPUT_TYPE_MPEGDEC
capability	Capability flags of this input: V4L2_INPUT_CAP_AUDIO - The input has an associated audio channel.
assocaudio	Audio input is associated to this video input.
reserved	Reserved for future capabilities.

Analog Video Subsystem (Continued)

1.6.9 VIDEO_TUNER

V4L2 name: v4l2_tuner

```
typedef struct {
        U32 input;
        char name[32];
        VIDEO_STANDARD std;
        U32 capability;
        U32 rangelow;
        U32 rangehigh;
        U32 rxsubchans;
        U32 audmode;
        U32 signal;
        U32 afc;
        U32 reserved[4];
} VIDEO_TUNER;
```

Structure Member	Description
input	The video input for this tuner.
name	Canonical name for this tuner.
std	The video standard supported by this tuner.
capability	Tuner capability flags: V4L2_TUNER_CAP_LOW - Frequency is in a lower range. V4L2_TUNER_CAP_NORM - The norm for this tuner is settable. V4L2_TUNER_CAP_STEREO - Stereo reception is supported. V4L2_TUNER_CAP_LANG1 - First language (of two or more) reception is supported. V4L2_TUNER_CAP_LANG2 - Second language channel reception is supported. V4L2_TUNER_CAP_SAP - Second audio program reception is supported.
rangelow	Lowest tunable frequency.
rangehigh	Highest tunable frequency.
rxsubchans	Audio subprograms currently being received: V4L2_TUNER_SUB_MONO - Receiving a mono signal. V4L2_TUNER_SUB_STEREO - Receiving a stereo signal. V4L2_TUNER_SUB_LANG1 - Receiving first language signal. V4L2_TUNER_SUB_LANG2 - Receiving a second language signal. V4L2_TUNER_SUB_SAP - Receiving a second audio program signal.
audmode	Currently selected audio mode: V4L2_TUNER_MODE_MONO - Playing mono audio. V4L2_TUNER_MODE_STEREO - Playing stereo audio. V4L2_TUNER_MODE_LANG1 - Playing first language audio. V4L2_TUNER_MODE_LANG2 - Playing second language audio. V4L2_TUNER_MODE_SAP - Playing SAP audio.
signal	Signal strength if known.
afc	If negative, tune higher; if positive, tune lower.
reserved	Reserved for future capabilities.

Analog Video Subsystem (Continued)

1.6.10 ENUM_STANDARD

V4L2 name: v4l2_enumstd

```
typedef struct {
        int     index;
        struct  VIDEO_STANDARD std;
        u32     inputs;
        U32     outputs;
        U32     reserved[2];
} ENUM_STANDARD;
```

Structure Member	Description
index	Index to query the standard.
std	The actual video standard information (see Section 1.6.11 "VIDEO_STANDARD" on page 27).
inputs	Number of inputs that support this standard.
outputs	Number of outputs that support this standard.
reserved	Reserved for future capabilities.

1.6.11 VIDEO_STANDARD

V4L2 name: v4l2_standard

```
typedef struct {
        u8      name[24];
        struct {
                U32     numerator;
                U32     denominator;
        } framerate;
        U32 framelines;
        U32 reserved1;
        U32 colorstandard;
        union {
                struct {
                        U32 colorsubcarrier;  /* Hz */
                } pal ;
                struct {
                        U32 colorsubcarrier;  /* Hz */
                } ntsc;
                struct {
                        U32 f0b;                        /* Hz (blue) */
                        U32 f0r;            /* Hz (red) */
                } secam;
                u8 reserved[12];
        } colorstandard_data;
        U32 transmission;           /* Bit field (zero for non-modulators/demodulators) */
        U32 reserved2;                      /* set to zero */
} VIDEO_STANDARD;
```

Analog Video Subsystem (Continued)

Structure Member	Description
name	Name of the standard.
framerate	Structure containing the numerator and denominator of the frame rate.
framelines	Number of lines in a frame.
reserved	Reserved for future capabilities.
colorstandard	The color-standard. Possible values for this field: V4L2_COLOR_STD_PAL V4L2_COLOR_STD_NTSC V4L2_COLOR_STD_SECAM
colorstandard_data	Union of structures containing color standard information. For PAL and NTSC, this holds the color sub-carrier frequency in Hz. For SECAM, it holds the red and blue sub-carrier frequencies. Common values are: V4L2_COLOR_SUBC_PAL V4L2_COLOR_SUBC_PAL_M V4L2_COLOR_SUBC_PAL_N V4L2_COLOR_SUBC_NTSC V4L2_COLOR_SUBC_SECAMB V4L2_COLOR_SUBC_SECAMR
transmission	Bit field indicating the transmission type: V4L2_TRANSM_STD_B V4L2_TRANSM_STD_D V4L2_TRANSM_STD_G V4L2_TRANSM_STD_H V4L2_TRANSM_STD_I V4L2_TRANSM_STD_K V4L2_TRANSM_STD_K1 V4L2_TRANSM_STD_L V4L2_TRANSM_STD_M V4L2_TRANSM_STD_N

2.0 Digital Video Subsystem

Digital video (MPEG) data finds its way into the system via the X-Port of the video decoder (e.g., Philips SAA7114). This port provides the decoder with a digital input to the built-in scaler. The scaled video is sent to the core processor via the VIP port. The decoder serves as a switch, to select either analog data from the TV tuner, composite or S-video or digital data (CCIR656) from the MPEG decoder (e.g., Sigma Designs). Since digital video is usually of a higher frame size than normal video, it might have to be scaled by the decoder before being presented.

2.1 DIGITAL VIDEO DISC (DVD)

The LinuxTV project currently offers a complete DVD framework, defined APIs, and working DVD player software (assuming hardware decode). With regards to the variety of DVD authoring schemes available, LinuxTV is fairly robust, supporting DVD video discs, DVD image files on drive file systems (HD / NFS / etc.), and udf filesystems. Unfortunately, the API is in early development. The API presented here is similar in functionality to the proposed DVD framework by the LinuxTV project.

Figure 2-1 shows the framework for playback of DVD video under Linux. The left side shows the data source, like a DVD ROM drive or the content of DVD video in the normal file system. The right side shows the decoders and the display of the content. The SPU decoder is required if menus or subtitles are to be supported. The DVD navigator is responsible for control and for moving the data from the disc to the decoder.

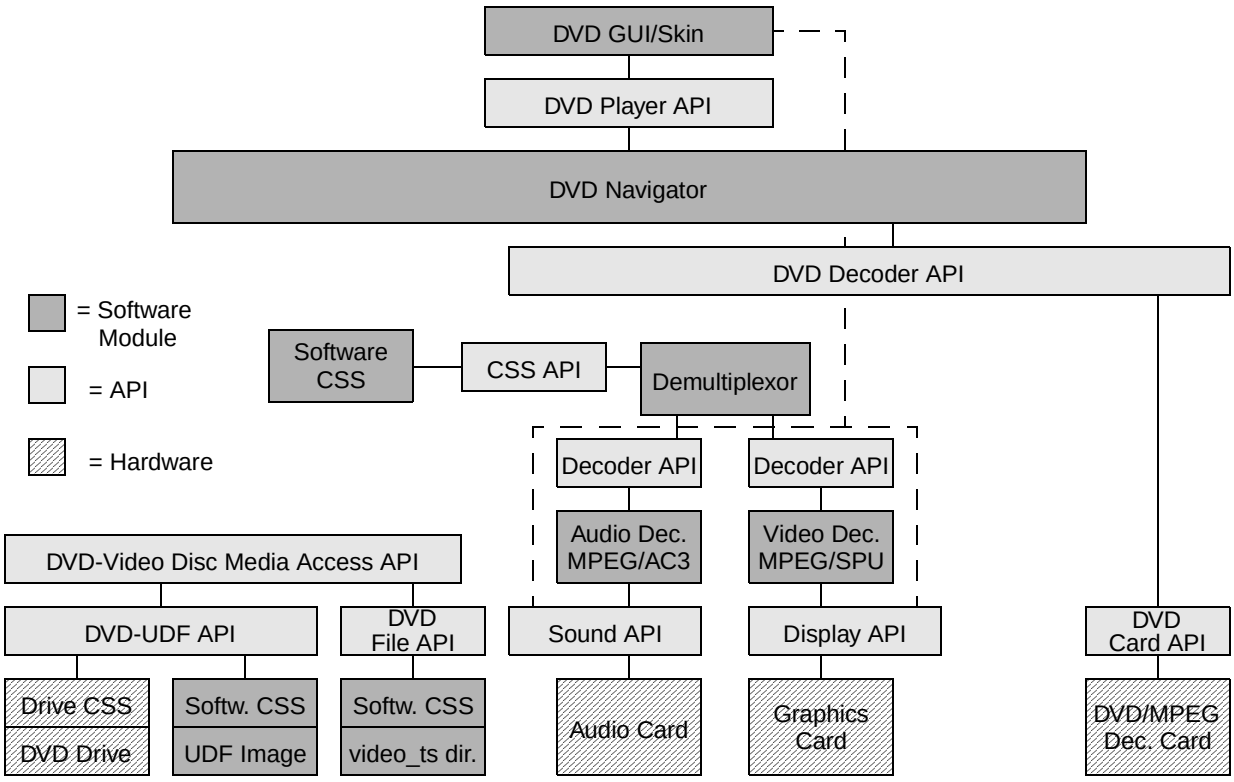


Figure 2-1. Linux DVD Framework

Digital Video Subsystem (Continued)

The navigator accesses the DVD data with the dvd_disc API. This uses either the dvd_udf API for block access or the dvd_file API for file access.

The data source can be (from left to right):

- A DVD video disc in a DVD ROM drive. The CSS is done by the drive in hardware.
- The image file of a DVD video disc, available as one huge file or many data fragments on the filesystem (HD / NFS / etc.) The CSS for encrypted images has to be done in software.
- The files of the VIDEO_TS directory from a DVD video disc. Either from a udf mounted disc or copies of these files on the filesystem. The CSS for encrypted images has to be done in software, and only when the keys were previously retrieved from the disc.

The decoder can be done either in software or hardware. The navigator uses a decoder API to access the demultiplexor with the adjoining decoders. For a hardware DVD decoder card, the card can do all the demultiplexing and decoding internally. Therefore, the decoder API passes its calls on to the IOCTL interface of the card, which handles the CSS, video, audio and SPU decoding and displays the video on a composite video output.

For all other decoders, like a software AC3 decoder or a hardware MPEG decoder card, demultiplexing of the DVD data must be done in software. The demultiplexor has to access an optional CSS decoder for encrypted movies. The CSS decoder module is accessed with a CSS API.

The media decoder (such as MPEG video, MPEG-, AC3-, LPCM-, SDDS-, DTS-audio or SPU bitmap overlay) are hooked into the demultiplexor with a plug-in API, that is currently under development. The video or audio decoder has to decode the stream data. Then the decoded audio or video data has to be displayed on the available hardware (e.g., a sound card or the VGA graphics card).

The dashed lines (in Figure 2-1) are optional controls for volume, brightness, contrast, window position, and other display settings.

Digital Video Subsystem (Continued)

2.2 DIGITAL VIDEO BROADCASTING (DVB)

DVB is a transmission scheme based on the MPEG-2 video compression/transmission scheme. DVB provides superior picture quality as compared with standard broadcast television, with the opportunity to view pictures in standard format or wide screen (16:9) format with mono, stereo or surround sound. It also allows a range of new features and services including subtitling, multiple audio tracks, interactive content, multimedia content - where, for instance, programs may be linked to World Wide Web material.

DVB transaction is based on a generic transport system that does not impose any restriction on the type of material being sent. Transmission techniques have been defined for video, audio, Electronic Program Guides (EPG) (multimedia magazine-style), Electronic Service Guide (ESG) teletext-style program guides which are processed and displayed using the receiver's menus), pay-per-view TV, data carousels (resembling traditional teletext) and Internet Protocol packets over the same system.

Note: DVB is currently being revision and will be updated in the next revision.

2.2.1 Program Guides

The purpose of a program guide is to assist the viewer in selection of the program to be viewed. The guide presents information about current programs being shown and upcoming programs. The program guide is not bound to the channel(s) being viewed, and it is feasible to see informaiton from channels that are not currently transmitting. The program guide carries all program information, including regional variations, which it may not be possible to receive outside a specific area, within a common MPEG-2 multiplex stream.

2.2.1.1 Event Schedule Guide (ESG)

The simplest type of guide resembles that provided by the analogue teletext system, Event Schedule Guide (ESG). The ESG may be constructed by the receiver based on the Service Information (SI) contained in the MPEG-2 multiplex stream. Transmitting the information in this way allows the receiver manufacturers and viewers to determine the presentation format - level of detail, preferred presentation style, channels of interest, type of program desired, etc.

2.2.1.2 Electronic Program Guide (EPG)

An EPG presents an additional service to the user in the form of a multimedia magazine-style channel guide. This type of guide requires support in the manufacture's set-top box for a consistent interface to allow the view of video stills, movie clips, buttons, layout, etc. It is expected that the look and feel for this type of guide will be dictated by the service provider. EPGs have some considerable advantage when considering other services which may be provided, such as, on-screen shopping or remote banking.

2.2.2 Video/Audio Data

Under DVB, the data to be transmitted is known as an Elementary Stream (ES). Elementary streams come in various forms including, Digital Control Data, Digital Audio (sampled and compressed), Digital Video (sampled and compressed) and Digital Data (synchronous, or asynchronous). A DVB transmission is transmitted in a MPEG-2 bit stream (which may consist of: MPEG-2 compressed video, compressed audio, control data and/or user data) and are combined into a single synchronous transmission bit stream. The process of combining the steams is known as multiplexing.

Each stream is input to an MPEG-2 processor (e.g., a video compressor or data formatted) which accumulates the data into a stream of Packetized Elementary Stream (PES) packets. A PES packet may be a fixed (or variable) sized block, with up to 65536 bytes per block and includes a 6-byte protocol header. For audio/video streams the header includes time stamps which may be used to synchronize a set of elementary streams and control the rate at which they are replayed by the receiver.

2.2.3 Multiplexing

The MPEG-2 standard allows two forms of multiplexing, MPEG Transport Stream and MPEG Program Stream. In a transport stream, each PES packet is broken into fixed-sized transport packets forming a general purpose way of combining one or more streams, possibly with independent time bases. This uses fixed length packets and is suited for transmission in which there may be potential packet loss or corruption by noise. A program stream is a group of tightly coupled PES packets referenced to the same time base. Such streams are suited for transmission in a relatively error-free environment and enable easy software processing of the received data. This form of multiplexing is used for video playback and network applications. DVB uses the MPEG-2 Transport Stream (TS) as the service multiplex.

Streams can be multiplexed on the same transmission and is known as MCPC (Multiple Channel Per Carrier). Given an average satellite transponder with a bandwidth of 27 MHz, typically, the highest symbol rate that can be used is SR 26MS/s. Obviously, with this large bandwidth, multiple video or audio channels can be transmitted via the transponder at the same time. MCPC uses a technique called Time Division Multiplex (TDM) to transmit the multiple programs at the same time.

Digital Video Subsystem (Continued)

As one can expect from the name, TDM sends data for one channel at a certain time and then data for another channel at another time. If the transmission uses only a part of the available transponder bandwidth for a single transmission, this is known as SCPC (Single Channel Per Carrier). In general, the satellite operator sends data as a MCPC transmission.

MPEG-2 transmits its data in packets of 188 bytes each. At the start of each packet is a package identifier (or PID) that tells the receiver what to do with the packet. Because the MPEG-2 data stream might be in MCPC mode, the receiver has to decide which packets are part of the current channel being watched and therefore pass them onto the video decoder for further processing. Those packets that are not part of the current channel are simply discarded.

There are typically four types of PIDs used by satellite receivers. The VPID is the PID for the video stream and the APID is the audio stream. Occasionally, a PCR PID (program clock reference) is used to synchronize the video and audio packets, however, most of the time, this data is embedded into the video stream. The forth data PID is used for data such as the program guide, information about other frequencies that make up the total package, etc. This data is called the System Information and uses a PID value of between 0000 and 0014 (hex).

The demultiplexor on the DVB receiver uses this PID to identify a particular program that has been selected for viewing. All other PIDs are typical discarded. Before the MPEG data is reassembled, it typically will go through a descrambler which uses decryption keys stored in the Conditional Access Module (CAM).

2.2.4 Conditional Access

In the DVB specification, there are only a few of the stream types that must be transmitted without scrambling. These non-scrambled streams include some of the systems information streams such as the Program Association Tables (points to more info about each channel) and the Network Information Table (points to the other transponders used by the service). These streams must be transmitted without scrambling so that any DVB compliant receiver can determine basic information about the DVB service.

Scrambling of the appropriate streams is performed at the uplink site. The MPEG-2 packets are encrypted based on a common key known to both the scrambling and decryption devices. When a scrambled packet arrives, before it is passed through to the demultiplexor, it is first sent through the CAM. The CAM is the descrambling engine and can be either built directly into the receiver or inserted into the receiver via a PC card (aka PCMCIA) connector. At the start of each MPEG-2 packet is a 2-bit field called the Transport Scrambling Flags (TSF). If the value is zero or one, the packet is passed through the CAM onto the demultiplexor for display since this value indicates an un-scrambled stream. If the TSF is set to either two or three, then the packet is passed through to the CAM. This takes the key obtained from the smartcard and uses it to turn the scrambled packet back into an MPEG-2/DVB transport packet that can then be processed by the rest of the system.

2.2.5 DVB Network driver

The DVB support is a video source as well as a data source. The DVB infrastructure is used to carry video MPEG and can be used for the reception of Internet Protocol (IP) data. A network driver is a different then a block or character driver because network driver receives data asynchronously without any specific application ready to receive the data. That is, packets can be received from a network without any particular application ready to receive the data. The data casting functionality must therefore tie into the Linux networking architecture. In the Microsoft Windows world, this would be similar to writing a NDIS driver.

IP packets that are encapsulated in an MPEG stream are passed through the Linux networking stack and can be received by applications listen the necessary sockets to receive the DVB information.

2.2.6 DVB Video Interface

The DVB API builds upon the original Video4Linux(V4L) API with plans to move to Video4Linux2 in the near future (discussed in Section 2.2 "Digital Video Broadcasting (DVB)" on page 31). Currently only the Siemens/Fujitsu DVB and Penta-Media PCI cards are supported by the DVB Linux driver. The driver hooks itself into the operating system as a video4linux device and is used similarly to a video capture card.

For the purpose of the National Geode SC1200 and SC1210 based hardware platforms, a driver for both the /dev/video, /dev/vbi and /dev/codec driver interfaces seem to be the minimal set of drivers which are required. There may be more than one video driver interface, one for each video source, composite video in, dvb video in, mpeg video in could correspond to /dev/video0, /dev/video1, and /dev/video2, respectively.

Digital Video Subsystem (Continued)

2.3 DIGITAL VIDEO - MPEG DECODER (DVM) FUNCTIONS

2.3.1 VMW_DVM_Open

Description:

Open the MPEG decoder driver.

Syntax:

U32 VMW_DVM_Open(U32 flags, U32 size, U32 count, (void *) pCallback)

Parameters:

Parameter	Description
flags	Specifies initial demultiplexor and stream playback mode: MPEG_TRANSPORT - Transport stream demultiplexor. MPEG_SYSTEM - System/program stream demultiplexor. MPEG_VIDEO - Video stream. MPEG_AUDIO - MPEG audio stream. MPEG_DVD - DVD demultiplexor. MEPG_SVCD - SVCD/VCD demultiplexor.
size	Size of buffers.
count	Number of buffers to receive stream notification.
pCallback	Callback function to receive stream notifications.

Returns:

Return	Description
VMW_OK	On success.
Non-zero error code	On failure. The return values on failure can be VMW_DVME_OPEN or VMW_DVME_MPEGOPEN.

Comments:

None

Also see:

Section 2.3.7 "VMW_DVM_Close" on page 36.

Digital Video Subsystem (Continued)

2.3.2 VMW_DVM_GetBufferPointer

Description:

Get a buffer for MPEG decoding. Before starting a MPEG decode operation, the application gets a pointer to a buffer where the data to be decoded is going to be placed.

Syntax:

U32 VMW_DVM_GetBufferPointer(DECODE_BUFFER_INFO *pDecodeBufferInfo)

Parameters:

Parameter	Description
pDecodeBufferInfo	typedef struct { U16 *pBuffer; /*Pointer to the buffer*/ U32 bufferSize; /*size of the buffer */ U32 dataSize; /*size of valid data in buffer*/ U32 flags; /*Flags*/ U32 reserved[8]; /*reserved for more parameters*/ } DECODE_BUFFER_INFO**;**

Returns:

Return	Description
VMW_OK	On success.
VMW_DVME_NOMEMORY	On failure.

Comments:

The data can be from any source (DVD, DVB, or hard disk file) once the buffer is allocated and application gets the pointer to fill the stream to this buffer for decoding.

Also see:

None

2.3.3 VMW_DVM_StartMPEGDecode

Description:

Start the MPEG decoding process. The application must provide a pointer to the buffer that contains the data to be decoded.

Syntax:

U32 VMW_DVM_StartMPEGDecode(DECODE_BUFFER_INFO *pDecodeBuffer)

Parameters:

Parameter	Description
pDecodeBuffer->buffer	Pointer to the buffer that contains the data to be decoded.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVME_PUSH	On failure.

Comments:

None

Also see:

Section 2.3.2 "VMW_DVM_GetBufferPointer" on page 34.

Digital Video Subsystem (Continued)

2.3.4 VMW_DVM_Play

Description:

Begin playback of the decoded video and audio streams. Basically this functions sets up the VIP port to receive the video data from the MPEG decoder.

Syntax:

U32 VMW_DVM_Play()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVME_PLAY	On failure.

Comments:

None

Also see:

Section 2.3.5 "VMW_DVM_Pause" on page 35.
Section 2.3.6 "VMW_DVM_Stop" on page 36.

2.3.5 VMW_DVM_Pause

Description:

Pause the video and audio decoders or continue playing from the previously paused state.

Syntax:

U32 VMW_DVM_Pause(U32 pauseOnOff)

Parameters:

Parameter	Description
U32 pauseOnOff	If '1' pause audio and video playing, if '0' resume play.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVME_PAUSE	On failure.

Comments:

None

Also see:

Section 2.3.4 "VMW_DVM_Play" on page 35.
Section 2.3.6 "VMW_DVM_Stop" on page 36.

Digital Video Subsystem (Continued)

2.3.6 VMW_DVM_Stop

Description:

Stop playback of video and audio streams.

Syntax:

U32 VMW_DVM_Stop()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVME_STOP	On failure.

Comments:

None

Also see:

Section 2.3.4 "VMW_DVM_Play" on page 35.
Section 2.3.5 "VMW_DVM_Pause" on page 35.

2.3.7 VMW_DVM_Close

Description:

Close the MPEG decoder driver.

Syntax:

U32 VMW_DVM_Close()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVME_CLOSE	On failure.

Comments:

None

Also see:

Section 2.3.1 "VMW_DVM_Open" on page 33.

Digital Video Subsystem (Continued)

2.4 DIGITAL VIDEO DISC (DVD) FUNCTIONS

2.4.1 VMW_DVD_Open

Description:

Opens the device driver and perform any necessary initialization of the driver or the device.

Syntax:

U32 VMW_DVD_Open()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_OPEN	On failure.

Comments:

None

Also see:

Section 2.4.2 "VMW_DVD_Close" on page 37.

2.4.2 VMW_DVD_Close

Description:

Close the device driver and perform any necessary clean up of the driver or the device.

Syntax:

U32 VMW_DVD_Close()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_CLOSE	On failure.

Comments:

None

Also see:

Section 2.4.1 "VMW_DVD_Open" on page 37.

Digital Video Subsystem (Continued)

2.4.3 VMW_DVD_TitlePlay

Description:

Start playing the DVD from the specified title.

Syntax:

U32 VMW_DVD_TitlePlay(U32 title)

Parameters:

Parameter	Description
title	Specified Title of the DVD to be played.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_TITLENUMBER	On failure.

Comments:

This function is complementary to the VMW_DVD_Play function.

Also see:

Section 2.4.4 "VMW_DVD_ChapterPlay" on page 38.
Section 2.4.5 "VMW_DVD_TimePlay" on page 39

2.4.4 VMW_DVD_ChapterPlay

Description:

Start playing the requested chapter in the specified title.

Syntax:

U32 VMW_DVD_ChapterPlay(U32 title,U32 chapter)

Parameters:

Parameter	Description
title	Specifies title in which the chapter belongs to.
chapter	Specifies chapter number of specified title.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_CHAPTER	On failure.

Comments:

This function is complementary to the VMW_DVD_Play function.

Also see:

Section 2.4.3 "VMW_DVD_TitlePlay" on page 38.

Digital Video Subsystem (Continued)

2.4.5 VMW_DVD_TimePlay

Description:

Start playing from the specified time under the given title.

Syntax:

U32 VMW_DVD_TimePlay(U32 title,DVD_TIME_CODE time)

Parameters:

Parameter	Description
title	Specified title of the DVD to be played.
time	Time structure specifies time format to play. typedef struct { U32 hour1; /* Hours */ U32 hour10; /* Tens of hours */ U32 minutes1; /* Minutes */ U32 minutes10; /* Tens of minutes */ U32 seconds1; /* Seconds */ U32 frames; /* Frames */ U32 frames10; /* Tens of frames */ } DVD_TIME_CODE;

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STOP	On failure.

Comments:

The TIME_CODE structure is followed from DirectShow by Microsoft.

Also see:

Section 2.4.3 "VMW_DVD_TitlePlay" on page 38.
Section 2.4.4 "VMW_DVD_ChapterPlay" on page 38.

2.4.6 VMW_DVD_PauseOn

Description:

Pause a playing DVD stream.

Syntax:

U32 VMW_DVD_PauseOn()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_PAUSE	On failure.

Comments:

None

Also see:

Section 2.4.7 "VMW_DVD_PauseOff" on page 40.
Section 2.4.8 "VMW_DVD_Stop" on page 40.

Digital Video Subsystem (Continued)

2.4.7 VMW_DVD_PauseOff

Description:

Resume playing the DVD stream from where it was paused.

Syntax:

U32 VMW_DVD_PauseOff()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_PAUSE	On failure.

Comments:

None

Also see:

Section 2.4.6 "VMW_DVD_PauseOn" on page 39.
Section 2.4.8 "VMW_DVD_Stop" on page 40.

2.4.8 VMW_DVD_Stop

Description:

Stop playing the DVD.

Syntax:

U32 VMW_DVD_Stop()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STOP	On failure.

Comments:

None

Also see:

Section 2.4.7 "VMW_DVD_PauseOff" on page 40.
Section 2.4.3 "VMW_DVD_TitlePlay" on page 38.

Digital Video Subsystem (Continued)

2.4.9 VMW_DVD_Search

Description:

Put the DVD navigator into a search mode.

Syntax:

U32 VMW_DVD_Search(U32 searchMode, U32 *pSearchParam)

Parameters:

Parameter	Description
searchMode	SearchMode flag: DVD_CHAPTER_SRCH DVD_TIME_SRCH DVD_GOUP_SRCH DVD_TOP_SRCH DVD_NEXT_SRCH DVD_PREV_SRCH
pSearchParam	Meaningful only if ChapterSearch or TimeSearch mode.The pointer is dereferenced based on the TimeSearch or ChapterSearch.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_INSEARCH	On failure.

Comments:

This function can be split into an individual search to make it more search-specific/meaningful. For example, the user need not pass any argument to search and it will go to the next program in the program chain but the ChapterSearch, ChapterNo must be specified. DirectShow implements this as individual APIs.

Also see:

None

2.4.10 VMW_DVD_ForwardScan

Description:

Search forward through the current disc at the specified speed.

Syntax:

U32 VMW_DVD_ForwardScan(float speed)

Parameters:

Parameter	Description
speed	The value that specifies how quickly the function will search forward, through the media file.1.0 is the authored speed. 0.0 < speed < 1.0 – Slow forward motion. speed > 1.0 – Fast forward play.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_SCAN	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.4.11 VMW_DVD_BackwardScan

Description:

Scan the DVD by playing forward at non-normal speeds.

Syntax:

U32 VMW_DVD_BackwardScan(float speed)

Parameters:

Parameter	Description
speed	The value that specifies how quickly the function will search Backward, through the media file. 1.0 is the authored speed. 0.0 < speed < 1.0 – Slow Backward motion. speed > 1.0 – Fast Backward play.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_SCAN	On failure.

Comments:

None

Also see:

Section 2.4.10 "VMW_DVD_ForwardScan" on page 41.

2.4.12 VMW_DVD_ActivateMenu

Description:

Activate the specified menu.

Syntax:

U32 VMW_DVD_ActivateMenu(U32 menuId)

Parameters:

Parameter	Description
menuId	Menu ID to activate: DVD_MENU_ID_TITLE DVD_MENU_ID_ROOT DVD_MENU_ID_ANGLE DVD_MENU_ID_AUDIO DVD_MENU_ID_SUBPICTURE DVD_MENU_ID_PART DVD_MENU_ID_RESUME

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_MENU	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.4.13 VMW_DVD_ActivateButton

Description:

Select and activate a given button (or an equivalent mouse-click position).

Syntax:

U32 VMW_DVD_ActivateButton(U32 param)

Parameters:

Parameter	Description
param	Represents the button to activate.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

Before calling this function, buttons must be selected.

Also see:

None

2.4.14 VMW_DVD_LanguageSelect

Description:

Set the specified ISO 639 language for the menus.

Syntax:

U32 VMW_DVD_LanguageSelect(char *pLanguageCode)

Parameters:

Parameter	Description
pLanguageCode	ISO-639 2-character language code.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STREAM	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.4.15 VMW_DVD_AudioStreamSelect

Description:

Select the specified audio stream for playback.

Syntax:

U32 VMW_DVD_AudioStreamSelect(U32 streamId)

Parameters:

Parameter	Description
streamId	Specifies the audio stream number to be played (between 0 and 7).

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STREAM	On failure.

Comments:

None

Also see:

None

2.4.16 VMW_DVD_AngleSelect

Description:

Select the specified angle.

Syntax:

U32 VMW_DVD_AngleSelect(U32 angle)

Parameters:

Parameter	Description
angle	Specifies the AngleNumber (must be between 1 and 9).

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STREAM	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.4.17 VMW_DVD_SubPictureStreamSelect

Description:

Select the specified sub-picture stream of the available stream.

Syntax:

U32 VMW_DVD_SubPictureStreamSelect(U32 subPicStream,U32 setDisplay)

Parameters:

Parameter	Description
subPicStream	Specifies the sub-picture stream number (must be between 0 and 31).
setDisplay	If '1' on display the substream and '0' sets off the substream display.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STREAM	On failure.

Comments:

None

Also see:

None

2.4.18 VMW_DVD_SetParentalControlLevel

Description:

Set the country and level of parental management system to be applied.

Syntax:

U32 VMW_DVD_SetParentalControlLevel(char *pCountry, U32 level)

Parameters:

Parameter	Description
pCountry	Gives the 2-character country code according to ISO3166.
level	Specifies the level (0 [Disabled], 1…8) for parental management system like PG, PG13, etc.

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STREAM	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.4.19 VMW_DVD_Karaoke

Description:

Set the audibility of the different karaoke tracks.

Syntax:

U32 VMW_DVD_Karaoke(U32 mixingMode)

Parameters:

Parameter	Description
mixingMode	Specifies the source to mix into the left/right channels: Bit off: Don't mix, Bit on: Do the mixing. If bit 2 is on, mix Ch2 to Ch1 (Melody to Right) If bit 3 is on, mix Ch3 to Ch1 (Voice1 to Right) If bit 4 is on, mix Ch4 to Ch1 (Voice2 to Right) If bit 10 is on, mix Ch2 to Ch0 (Melody to Left) If bit 11 is on, mix Ch3 to Ch0 (Voice1 to Left) If bit 12 is on, mix Ch4 to Ch0 (Voice2 to Left)

Returns:

Return	Description
VMW_OK	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.4.20 VMW_DVD_SetVideoPresentationMode

Description:

Set the video presentation mode.

Syntax:

U32 VMW_DVD_SetVideoPresentationMode(U32 aspectRatio,U32 displayMode)

Parameters:

Parameter	Description
aspectRatio	Specifies the aspect ratio: 0: Aspect ratio of 4:3 1: Aspect ratio of 16:9
displayMode	Specifies the display mode: 0: Normal (4:3 or 16:9) 1: Pan-scan 2: Letterbox

Returns:

Return	Description
VMW_OK	On success.
VMW_DVDE_STREAM	On failure.

Comments:

None

Also see:

None

2.5 DIGITAL VIDEO BROADCAST (DVB) FUNCTIONS

2.5.1 VMW_DVB_Open

Description:

Open the device driver and perform any necessary initialization of the driver or the device.

Syntax:

U32 VMW_DVB_Open()

Parameters:

None

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 2.5.2 "VMW_DVB_Close" on page 48.

2.5.2 VMW_DVB_Close

Description:

Close the device driver and performs any necessary clean up of the driver or the device.

Syntax:

U32 VMW_DVB_Close()

Parameters:

None

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 2.5.1 "VMW_DVB_Open" on page 48.

Digital Video Subsystem (Continued)

2.5.3 VMW_DVB_GetProgramAssociationTable

Description:

Fill the Program Association Table (PAT) structure. The PAT provides the correspondence between a program number and the PID value of the transport stream packets, which carry the program definition.

Syntax:

U32 VMW_DVB_GetProgramAssociationTable (PAT *pTable)

Parameters:

Parameter	Description

pTable	Typedef struct {
	U16 transportStreamId;
	U8 versionNumber;
	U16 numberOfPrograms; // number of programs or channels
	PROGRAM_INFO programs[MAX_PROGRAMS];
	} PAT;
	typedef struct
	{
	U16 programNumber; // The program_number may be
	// used as a designation for a
	// broadcast channel, for example.
	U16 programMapPID; // PID of the Transport Stream
	// packets which shall contain the
	// program_map_section.
	} PROGRAM_INFO;

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 2.5.4 "VMW_DVB_GetProgramMapTable" on page 50.

Digital Video Subsystem (Continued)

2.5.4 VMW_DVB_GetProgramMapTable

Description:

Fills the Program Map Table (PMT) structure. The PMT provides the mappings between program numbers and the program elements that comprise them.

Syntax:

U32 VMW_DVB_GetProgramMapTable(U32 programNumber, PMT *pMap)

Parameters:

Parameter	Description
programNumber	programNumber is an input parameter from PAT.

PMT *pMap	Typedef struct {
	U16 programNumber;
	U8 versionNumber;
	U16 PCR_PID;
	U16 numberOfStreams;
	STREAM_INFO streams[MAX_STREAMS];
	} PMT;
	typedef struct {
	U8 streamType;
	U16 elementaryPID; // PID of the stream
	} STREAM_INFO;

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 2.5.3 "VMW_DVB_GetProgramAssociationTable" on page 49.

Digital Video Subsystem (Continued)

2.5.5 VMW_DVB_GetNetworkInformationTable

Description:

Fill the Network Information Table (NIT) structure. The NIT contains information related to the physical organization of the multiplexes/Transport streams carried via a given network (a collection of MPEG-2 transport stream multiplexes), and the characteristics of the network itself. Information related to satellite delivery is given in this table. See the DVD Specification from Toshiba Corporation for more details.

Syntax:

U32 VMW_DVB_GetNetworkInformationTable (NIT *pNetworkTable)

Parameters:

Parameter	Description

	typedef struct {
	U8 tableId;
	U16 networkId;
	U8 versionNumber;
	TS_MUX_INFO tsMuxes[MAX_STREAMS];
	} NIT;
	typedef struct {
	U16 transportStreamId;
	U16 originalNetworkId;
	U32 frequency; // frequency in GHz
	U16 orbitalPosition; // orbital position of
	// the satellite in degrees
	U8 westEastFlag; // 0 for western and
	// 1 for eastern position
	U8 polarization; // polarization of the transmitted
	// signal
	// 0 for linear – horizontal
	// 1 for linear – vertical
	// 2 for circular – left
	// 3 for circular – right
	U8 modulation; // modulation scheme used
	U32 symbolRate; // Msymbols/s
	} TS_MUX_INFO

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.5.6 VMW_DVB_GetServiceDescriptionTable

Description:

Fill the Network Information Table (NIT) structure. The NIT contains information related to the physical organization of the multiplexes/Transport streams carried via a given network (a collection of MPEG-2 Transport stream multiplexes), and the characteristics of the network itself. Information related to satellite delivery is given in this table. See the DVD Specification from Toshiba Corporation for more details.

Syntax:

U32 VMW_DVB_GetServiceDescriptionTable(SDT *pSDT)

Parameters:

Parameter	Description

PSDT	Pointer to the Service Description Table.
	typedef struct {
	U8 versionNumber;
	U16 transportStreamId;
	U16 originalNetworkId; // the label identifying the network_id
	// of the originating delivery system.
	U16 numberOfServices;
	TS_SERVICE_INFO tsServices[MAX_SERVICES];
	} SDT;
	typedef struct {
	U16 serviceId; // label to identify this service from any other
	// service within the TS.
	U8 EITScheduleFlag; // when set to "1" indicates that EIT schedule
	// information for the service is present in
	// the current TS.
	U8 EITPresentFollowingFlag;
	U8 runningStatus; // indicates the status of the service as follows:
	// 0 undefined,1 not running
	// 2 starts in a few seconds (e.g. for video recording)
	// 3 pausing, 4 running
	// 5 to 7 reserved for future use
	U8 freeCAMode; // When set to "0" indicates that all the component
	// streams of the Service is not scrambled. When set
	// to "1" it indicates that access to one or more
	// streams may be controlled by a CA system.
	SERVICE_NAME serviceNames[MAX_NAMES];
	} TS_SERVICE_INFO;
	typedef struct {
	U8 serviceType;
	char * pServiceProviderName;
	char * pServiceName;
	} SERVICE_NAME;

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.5.7 VMW_DVB_GetEventInformationTable

Description:

Fill the Event Information Table (EIT) structure. EIT provides information in chronological order regarding the events contained within each service.

Syntax:

Int VMW_DVB_GetEventInformationTable(EIT *pEIT)

Parameters:

Parameter	Description

EIT pEIT	Pointer to Event Information Table.
	typedef struct {
	U8 versionnumber;
	U16 transportStreamId;
	U16 originalNetworkId; // the label identifying the network_id
	// of the originating delivery system.
	U16 numberOfServices;
	TS_EVENT_INFO tsEvents[MAX_EVENTS];
	} EIT;
	typedef struct {
	U16 eventId; // Label to identify this service from any other
	// service within the TS.
	U64 startTime;
	U32 duration;
	U8 runningStatus; // indicates the status of the service as follows:
	// 0 undefined,1 not running
	// 2 starts in a few seconds (e.g. for video recording)
	// 3 pausing, 4 running
	U8 freeCAMmode; // When set to "0" indicates that all the component
	// streams of the Service is not scrambled. When
	// set to "1" it indicates that Access to one or more
	// streams may be controlled by a CA system.
	EVENT_NAME eventNames[MAX_NAMES];
	} TS_EVENT_INFO;
	typedef struct {
	Char * pEventName;
	Char * pEventDescription;
	} EVENT_NAME;

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.5.8 VMW_DVB_SetUpVideoCallBacks

Description:

Set up the callback functions for video capture. This function is called when a video PES or TS packet is captured by the hardware.

Syntax:

U32 VMW_DVB_SetUpVideoCallBacks(VCB_INFO * pVCBInfo)

Parameters:

Parameter	Description

pVCBInfo	typedef struct {
	U32 pid; // selected video PID to be filtered
	Void (* videoCallBackFunc) (U8 *pBuffer, U32 BufferSize, U8 PacketDone);
	U8 streamType; // 0 for PES, 1 for TS packets
	U32 bufferSize; // size of the buffer to reserve
	U32 numberOfBuffers; // number of buffers to reserve
	}VCB_INFO
	void (* videoCallBackFunc)(U8 *pBuffer, U32 BufferSize, U8 PacketDone)
	Address of a function that will be called when some video data (a complete PES or a number of TS packets) is captured by the hardware. Three parameters will be passed to the function,
	U8 *pBuffer: A pointer to the buffer, which contains the data.
	U32 BufferSize: The size of the data in bytes
	U8 PacketDone: 1 if the complete PES packet is transferred,
	0 otherwise.
	For TS packets this will be always 1.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

VMW_DVB_ReleaseBuffer should be called in the callback function in order for the DVB middleware API layer to reuse the buffer. For TS packet capture, BufferSize should be a multiple of 188 bytes.

It is recommended to have a BufferSize of 64K for PES capture to facilitate the transfer of whole PES in just one call-back for most of the cases.

Also see:

Section 2.5.12 "VMW_DVB_ReleaseBuffer" on page 57.

Digital Video Subsystem (Continued)

2.5.9 VMW_DVB_SetUpAudioCallBacks

Description:

Set the callback functions for audio capture. This function is called when an audio PES or TS packet is captured by the hardware.

Syntax:

U32 VMW_DVB_SetUpAudioCallBacks(ACB_INFO *pACBInfo)

Parameters:

Parameter	Description

pACBInfo	typedef struct {
	U32 pid; // selected audio PID to be filtered
	Void (* AudioCallBack_Func)(U8* pBuffer, U32 BufferSize, U8 PacketDone);
	U8 streamType; // 0 for PES, 1 for TS packets
	U32 bufferSize; // size of the buffer to reserve
	U32 numberOfBuffers; // number of buffers to reserve
	}ACB_INFO
	void (* AudioCallBack_Func)(U8* pBuffer, U32 BufferSize, U8 PacketDone)
	Address of a function that will be called when some audio data (a complete PES or a number of TS packets) is captured by the hardware. Three parameters will be passed to the function:
	U8 *pBuffer: A pointer to the buffer, which contains the data.
	U32 BufferSize: The size of the data in bytes
	U8 PacketDone: 1 if the complete PES packet is transferred,
	0 otherwise.
	For TS packets this will be always 1.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

VMW_DVB_ReleaseBuffer should be called in the call back function in order for the DVB middleware API layer to reuse the buffer. For TS packet capture, BufferSize should be a multiple of 188 bytes.

Also see:

Section 2.5.12 "VMW_DVB_ReleaseBuffer" on page 57.

Digital Video Subsystem (Continued)

2.5.10 VMW_DVB_StartStreaming

Description:

Start feeding the audio and video data to the application. After calling this function, the application starts receiving callbacks. This function should be called after calling the SetUpVideoCallBacks or SetUpAudioCallbacks function.

Syntax:

U32 VMW_DVB_StartStreaming(U8 mediaType)

Parameters:

Parameter	Description
U8 mediaType	mediaType should be 1 for video stream and 2 for audio stream.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 2.5.11 "VMW_DVB_StopStreaming" on page 56.

2.5.11 VMW_DVB_StopStreaming

Description:

Stop feeding the audio and video data to the application.

Syntax:

U32 VMW_DVB_StopStreaming(U8 mediaType)

Parameters:

Parameter	Description
U8 mediaType	mediaType should be 1 for video stream and 2 for audio stream.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

Section 2.5.10 "VMW_DVB_StartStreaming" on page 56.

Digital Video Subsystem (Continued)

2.5.12 VMW_DVB_ReleaseBuffer

Description:

Mark the buffer as available for reuse by the DVB middleware API layer.

Syntax:

U32 VMW_DVB_ReleaseBuffer(U8 *pBuffer, int mediaType)

Parameters:

Parameter	Description
pBuffer	The address of the buffer to mark.
mediaType	Type of the media (Audio /Video)

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

This function should be called with different parameters inside the Audio/Video callback functions.

Also see:

None

2.6 DIGITAL VIDEO TUNER (DVT) FUNCTIONS

2.6.1 VMW_DVT_InitDigitalTuner

Description:

Initialize the digital tuner (HM1821).

Syntax:

U32 VMW_DVT_InitDigitalTuner(U32 manualTune, U8 refDivider);

Parameters:

Parameter	Description
manualTune	1 = Yes, tune manually 0 = No

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

2.6.2 VMW_DVT_TuneFrequency

Description:

Tune to the specified frequency.

Syntax:

U32 VMW_DVT_TuneFrequency(U32 frequency)

Parameters:

Parameter	Description
U32 frequency	The frequency to be tuned to.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Digital Video Subsystem (Continued)

2.6.3 VMW_DVT_SetSymbolRate

Description:

Set the symbol rate.

Syntax:

U32 VMW_DVT_SetSymbolRate(U32 symbolRate)

Parameters:

Parameter	Description
U32 symbolRate	The symbol rate to be set.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

2.6.4 VMW_DVT_GetPllLockStatus

Description:

Determine if PLL lock-detect is high.

Syntax:

U32 VMW_DVT_GetPllLockStatus(U8 * pMode)

Parameters:

Parameter	Description
U8 pMode	PLL Mode.

Returns:

Return	Description
0	On success.
Non-zero error code	On failure.

Comments:

None

Also see:

None

3.0 Overlay Subsytem

The Overlay subsystem provides the ability to overlay a video window on top of a graphics window. It is possible to alpha blend the video and graphics data with a specified blending value.

Up to three simultaneous alpha blending windows are supported. Alpha blending can be enabled only when video overlay is enabled. The coordinates and dimensions of the alpha windows should lie within the overlay window. Alpha blending for the portions lying outside the Overlay window will not work.

The application should know the physical location of the video data buffers with respect to the start of frame buffer in order to do overlays and alpha blending. This can be done by calling the VMW_OVL_GetVIPBufferInfo function.

This Overlay functionality can be used to display video data that comes as output of the analog TV tuner or the MPEG decoder through the VIP. The MPEG decoder can get inputs from either DVD or DVB. The video data coming through the VIP will be put in to the off-screen frame buffer area. Data can also come from an MPEG or .AVI file stored on the hard disk.

The overlaying of data coming through the VIP on top of graphics data is done using colorkey. The overlay hardware looks for the specified colorkey and overlays the video data wherever it finds the colorkey. When alpha blending is enabled, the overlay data is blended with the graphics data based on the alpha value specified, and the colorkey is ignored.

Based on the size of the buffer and the size of the destination rectangle, video data will be scaled to fit in to the destination rectangle. Only upscaling is supported. If any downscaling is needed, it is done in the analog tuner or the MPEG decoder. Once the video data comes into the frame buffer and if the destination rectangle is smaller than the buffer data, then the image will be cropped and some portion of the picture will be missing.

3.1 Overlay (OVL) Functions

3.1.1 VMW_OVL_EnableOverlay

Description:

Create an overlay window at the given coordinates with the given dimensions.

Syntax:

U32 VMW_OVL_EnableOverlay(U32 colorKey, U32 colorMask, int graphics)

Parameters:

Parameter	Description
colorKey	The colorkey that is used for the overlay.
colorMask	The bits of colorkey that will be masked. All bits set to zero in the mask are ignored from the colorkey. This can be used to make the colorkey to be between a range of values and not just one fixed value.
graphics	A flag that indicates whether graphics data or video data is to be compared with the colorkey.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Overlay Subsytem (Continued)

3.1.2 VMW_OVL_DisableOverlay

Description:

Disable the overlay from being displayed.

Syntax:

U32 VMW_OVL_DisableOverlay()

Parameters:

None

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

3.1.3 VMW_OVL_EnableAlphaBlendWindow

Description:

Create an alpha blended window. This function specifies alpha blended window parameters (i.e., coordinates, dimensions and alpha value). At any given time, only a maximum of three alpha windows can be created. The user can control parameters of any of the three alpha blending windows by specifying the alpha window number. Coordinates of an alpha blend window should lie within the video overlay window.

Syntax:

U32 VMW_OVL_DisableOverlay()

Parameters:

Parameter	Description
startX	X coordinate of top-left of alpha blend window.
startY	Y coordinate of top-left of alpha blend window.
width	Width of alpha blend window.
height	Height of alpha blend window
alphaValue	The alpha blend value.
alphaIncValue	Value by which alpha blend value is incremented at start of every new frame.
windowNumber	This should be 1, 2, or 3, depending on which window out of the three available alpha windows is being controlled.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
Non-zero error code	On failure.

Comments:

None

Also see:

None

Overlay Subsytem (Continued)

3.1.4 VMW_OVL_DisableAlphaWindow

Description:

Disable alpha window "x". This function allows the programmer to disable one of the three alpha windows.

Syntax:

U32 VMW_OVL_DisableAlphaWindow(U32 windowNumber)

Parameters:

Parameter	Description
windowNumber	This should be 1, 2, or 3; depends upon which alpha window is desired to be disabled.

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
	Non-zero error code on failure.

Comments:

None

Also see:

Section 3.1.1 "VMW_OVL_EnableOverlay" on page 60.

3.1.5 VMW_OVL_GetVIPBufferInfo

Description:

Retrieve information about the video data buffer in off-screen memory. This function returns information about the buffer in the off-screen memory where the VIP puts the video data. This function is used to program overlay with the right parameters.

Syntax:

U32 VMW_OVL_GetVIPBufferInfo(VIP_BUFFER_INFO * *pVIPBufferInfo)

Parameters:

Parameter	Description
VIP_BUFFER_INFO* pVIPBufferInfo	Pointer to a structure that has information about the video data buffer in off-screen memory.
typedef struct {	
U32 bufferOffset;	Offset of off-screen buffer from start of frame buffer memory.
U32 widthBytes;	Width in bytes of VIP buffer.
} VIP_BUFFER_INFO;	

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
	Non-zero error code on failure.

Comments:

None

Also see:

None

Overlay Subsytem (Continued)

3.1.6 VMW_OVL_SetVIPBufferInfo

Description:

Sets up the VIP hardware with information about the video data buffer in off-screen memory. This function sets up the VIP hardware with information about the buffer in off-screen memory where the VIP puts the video data.

Syntax:

U32 VMW_OVL_SetVIPBufferInfo(U32 evenAdd, U32 oddAdd, U32 pitch)

Parameters:

Parameter	Description
evenAdd	The start address of the even frame.
oddAdd	The start address of the odd frame.
pitch	The pitch (length in between two consecutive scan lines).

Returns:

Return	Description
VMW_OK	On success.
VMW_AVSE_INVSTAT	If the device is not opened.
	Non-zero error code on failure.

Comments:

None

Also see:

None

Appendix A Support Documentation

A.1 RELATED DOCUMENTS

This document specifically references or assumes knowledge of the following sources:

- EN 300 468 DVB Service Information (SI) Specification.
- DVD Specification from Toshiba Corporation
- ISO/IEC 13818 MPEG-2 Specification

A.2 ACRONYMS

This specification uses several abbreviations. Table 3-1 lists the acronyms and their meanings.

Table A-1. Acronym Definitions

Acronym	Definition
ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
AVS	Analog Video Subsystem
CAM	Conditional Access Module
CSS	Contents Scrambling System
DVB	Digital Video Broadcast
DVD	Digital Video Disc
DVM	Digital Video MPEG
EPG	Electronic Program Guide
ESG	Electronic Service Guide
HAL	Hardware Abstraction Layer
IOCTL	I/O Control
IAOC	Information Appliance on a Chip
IP	Internet Protocol
MCPC	Multiple Channel Per Carrier
MPEG	Motion Pictures Expert Group
PES	Packetized Elementary Stream
PID	Package Identifier
SCPC	Single Channel Per Carrier
SPU	Sub-Picture Unit
TDM	Time Division Multiplex
TS	Transport Stream
TSF	Transport Scrambling Flags
WWS	Wide Screen Signalling
VBI	Vertical Blanking Interrupt

Support Documentation (Continued)

A.3 SPECIFICATION REVISION HISTORY

This document is a report of the revision/creation process of the National Linux Middleware API Specification Programmer's Reference. Any revisions (i.e., additions, deletions, parameter corrections, etc.) are recorded in the table(s) below.

Table A-2. Revision History

Revision # (PDF Date)	Revisions / Comments
0.80 (8/1/00)	Initial creation - Merged previous documents into one.
0.99 (8/21/00)	Final Review
1.0 (8/28/00)	Release 1.0
1.01 (12/5/00)	Draft version 2
1.02 (01/12/01)	Drafter version 2 rev.2 1) Changed VMW_OVL_EnableOverlay (simplified). 2) Added new VBI functions. 3) Updated error codes.
1.03 (05/15/01)	Edited/formatted by tech pubs. Returned to software group for proofing.
1.04 (08/17/01)	Removed "Confidential" document status. Elaboraded on the term "Middleware" throughout the document.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Email:
new.feedback@nsc.com

National Semiconductor
Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 87 90

National Semiconductor
Asia Pacific Customer
Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

National Semiconductor
Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507
Email: nsj.crc@jksmtp.nsc.com

www.national.com