

# Geode™ SC1200/SC1210 IAOC Devices: Digital Video MPEG (DVM) API Implementation

## 1.0 Introduction

National Semiconductor® has developed a Linux Digital Video API (application programming interface) for set-top box platforms based on National's Information Appliance On a Chip (IAOC) devices: Geode™ SC1200 Set-Top Box On a Chip and SC1210 Multimedia System On a Chip.

Picture Expert Group) decoder, based on Sigma Designs' EM8400 MPEG-2 decoder.

The EM8400 consists of a driver level (kernel) interface to the hardware and upper layer. The upper layer traps the interrupt raised by the driver (in the user space), and data is processed in the user space. The DVM API wraps around the EM8400 API functions.

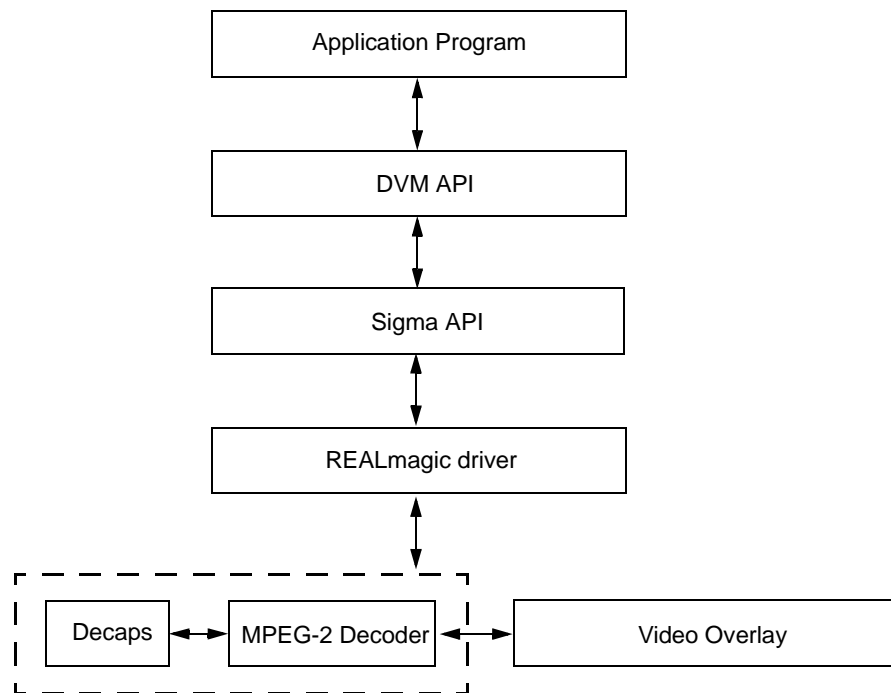
## 2.0 Discussion

This document describes the implementation of the Linux Digital Video API in the DVM (Digital Video MPEG) subsystem. The DVM subsystem employs MPEG-2 (Motion

The figure below depicts the DVM API software architecture and hardware interface in the DVM subsystem. The subsystem plays streams, such as the transport stream, system stream, audio and video.

---

### Linux Digital Video Application Programming Interface - DVM Subsystem



National Semiconductor is a registered trademark of National Semiconductor Corporation.  
 Geode is a trademark of National Semiconductor Corporation.  
 For a complete listing of National Semiconductor trademarks, please visit [www.national.com/trademarks](http://www.national.com/trademarks).

## 3.0 DVM API Functions

### 3.1 VMW\_DVM\_OPEN

**Description:**

Open the MPEG decoder device.

**Syntax:**

```
U32 VMW_DVM_Open(U32 flags,U32 size,U32 count,(void*)Callback);
```

**Parameters:**

Parameter	Description
flags	Specifies initial demux and stream playback model: MPEG_TRANSPORT: Transport stream demux MPEG_SYSTEM: System stream demux MPEG_DVM: DVM demux MPEG_VIDEO: Video system MPEG_AUDIO: MPEG audio stream MPEG_SVCD: SVCD/VCD demux
size	Size of buffers
count	Number of buffers to allocate
Callback	Callback function to receive the stream notifications

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_OPEN	Could not open device
VMW_DVME_MPEGOPEN	Error code from MPEGDriverEntry

**Implementation:**

Initialize the MPEG2 decoder driver. This should be done before opening the stream for playback. Open the driver with stream type, size, counts, and a pointer to callback function. Callback function pointer is passed to get the stream notifications.

**Pseudocode:**

```
U32 VMW_DVM_Open(U32 flags,U32 size,U32 count,(void*)Callback)
{
    if(MPEGDriverEntry(NO_DRIVE)) // error
        return VMW_DVME_MPEGOPEN; // error code from MPEGDriverEntry
    else { // no error
        if(FMPOpen(flags,size,count,Callback,0)) //error
            return VMW_DVME_OPEN; //error code from FMPOpen
        else
            return VMW_OK; // On success
    }
}
```

**Also see:**

VMW\_DVM\_Close()

### 3.2 VMW\_DVM\_CLOSE

**Description:**

Close the MPEG decoder driver. This stops decoding and unloads the driver.

**Syntax:**

```
U32 VMW_DVM_Close(void);
```

**Parameters:**

None

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_CLOSE	On failure

**Implementation:**

Close the stream and the driver. It is better to call FMP\_Close only after calling FMP\_Stop. After closing, the driver is unloaded.

**Pseudocode:**

```
U32 VMW_DVM_Close(void)
{
    if (!FMPStop() && !FMPClose() && !MPEGDriverUnload())
        return VMW_OK;
    else
        return VMW_DVME_CLOSE;
}
```

**Also see:**

VMW\_DVM\_Open()

### 3.3 VMW\_DVM\_PLAY

**Description:**

Set the driver to play mode.

**Syntax:**

```
U32 VMW_DVM_Play(void);
```

**Parameters:**

None

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_PLAY	On failure

**Implementation:**

This function plays the stream from the buffer.

**Pseudocode:**

```
U32 VMW_DVM_Play(void)
{
    if (FMPPPlay())                //error
        return VMW_DVME_PLAY;
    else
        return VMW_OK;
}
```

**Also see:**

VMW\_DVM\_Stop()

### 3.4 VMW\_DVM\_STOP

**Description:**

Stop the ongoing decoding process.

**Syntax:**

```
U32 VMW_DVM_Stop(void);
```

**Parameters:**

None

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_STOP	On failure

**Implementation:**

This function stops decoding the stream from the buffer. FMPStop is used to stop decoding the stream, and returns zero on success and non-zero on failure.

**Pseudocode:**

```
U32 VMW_DVM_Stop()
{
    if (FMPStop())                // error
        return VMW_DVME_STOP;
    else return VMW_OK;
}
```

**Also see:**

VMW\_DVM\_Play()

### 3.5 VMW\_DVM\_PAUSE

**Description:**

This function will either pause the decoder or resume the play from the paused state, depending on the parameter state.

**Syntax:**

```
U32 VMW_DVM_Pause(U32 pauseOnOff);
```

**Parameters:**

Parameter	Description
pauseOnOff	0: Resumes play from the paused state 1: Pauses the decoder

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_PAUSE	On failure

**Implementation:**

If pauseOnOff is '1', FMPPause() function is called, and the stream will be paused. If pauseOnOff is '0', FMPPPlay() function is called, and decoding is resumed.

**Pseudocode:**

```
U32 VMW_DVM_Pause(U32 pauseOnOff)
{
    if (pauseOnOff == 1)
        return ( FMPPause());
    else
        return ( FMPPPlay());
}
```

**Also see:**

None

### 3.6 U32 VMW\_DVM\_GETBUFFERPOINTER

**Description:**

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

**Syntax:**

```
U32 VMW_DVM_GetBufferPointer(DECODE_BUFFER_INFO *pDecodeBufferInfo);
```

**Parameters:**

Parameter	Description
pDecodeBufferInfo	<pre>typedef struct{     void *pBuffer;     U32 buffer_size;     U32 data_size;     U32 flags;     U32 reserved[8]; }DECODE_BUFFER_INFO;</pre>

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_NOMEMORY	On failure

**Implementation:**

This function allocates the buffer and returns the buffer pointer in the decoder where the decoding stream is going to be placed. FMPGetBuffer function allocates the buffer memory in the memory manager and returns an error if the memory is not available.

The DECODE\_BUFFER\_INFO struct has to be defined globally in the header.

**Pseudocode:**

```
U32 VMW_DVM_GetBufferPointer(DECODE_BUFFER_INFO *pDecodeBufferInfo);
{
    if (FMPGetBuffer((FMP_BUFFER*)pDecodeBufferInfo, TRUE)) //error
        return VMW_DVME_NOMEMORY;
    else
        return VMW_OK;
}
```

**Also see:**

None

### 3.7 VMW\_DVM\_STARTMPEGDECODE

**Description:**

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

**Syntax:**

```
U32 VMW_DVM_StartMPEGDecode(void* pBuffer);
```

**Parameters:**

Parameter	Description
*pBuffer	Pointer to the buffer that contains the data to be decoded.

**Returns:**

Return	Description
VMW_OK	On success
VMW_DVME_PUSH	On failure

**Implementation:**

The application has to copy the data to buffer of the specified size, and push it into the decoder.

**Pseudocode:**

```
U32 VMW_DVM_StartMPEGDecode(void* pBuffer)
{
    if (FMPPush((PFMP_BUFFER*) pBuffer))
        return VMW_DVME_PUSH;
    else
        return VMW_OK;
}
```

**Also see:**

None

## Appendix A Support Documentation

### A.1 REVISION HISTORY

This document is a report of the revision/creation process. Any revisions (i.e., additions, deletions, parameter corrections, etc.) are recorded in the table(s) below.

**Table A-1. Revision History**

<b>Revision # (PDF Date)</b>	<b>Revisions / Comments</b>
1.0 (7/06/01)	Release 1.0
1.1 (08/17/01)	Rewrote all pseudocode.

### 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](http://www.national.com)