MPFM Porting Guide

Version 2.2

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision D
November 1999

Copyright and publication information

This manual reflects version 2.5 of DAVID.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

November 1999

Table of Contents

Chapter 1: Overview of Porting MPFM

MPFM (Motion Picture File Manager) is a dual-ported I/O product that can be compiled for any OS-9 supported processor. MPFM manages both MPEG audio and video decoder devices. The following sections are included in this chapter:

Overview

The Motion Picture File Manager (MPFM) is responsible for:

- Permission and parameter checking
- Memory allocation/de-allocation for map descriptors
- Unit/map/path detection and synchronization
- Parameter distribution/insertion
- Dispatching calls to the audio/video device driver

The MPFM has four associated modules:

- `mpfm` File manager
- `mv` Descriptor for video decoder
- `ma` Descriptor for audio decoder
- `mpxxxx` Driver

The MPFM is a dual-ported I/O product that can be compiled for OS-9. The MPFM handles both MPEG audio and video decoder devices. The MPFM and part of the device driver will be the same for different ports.

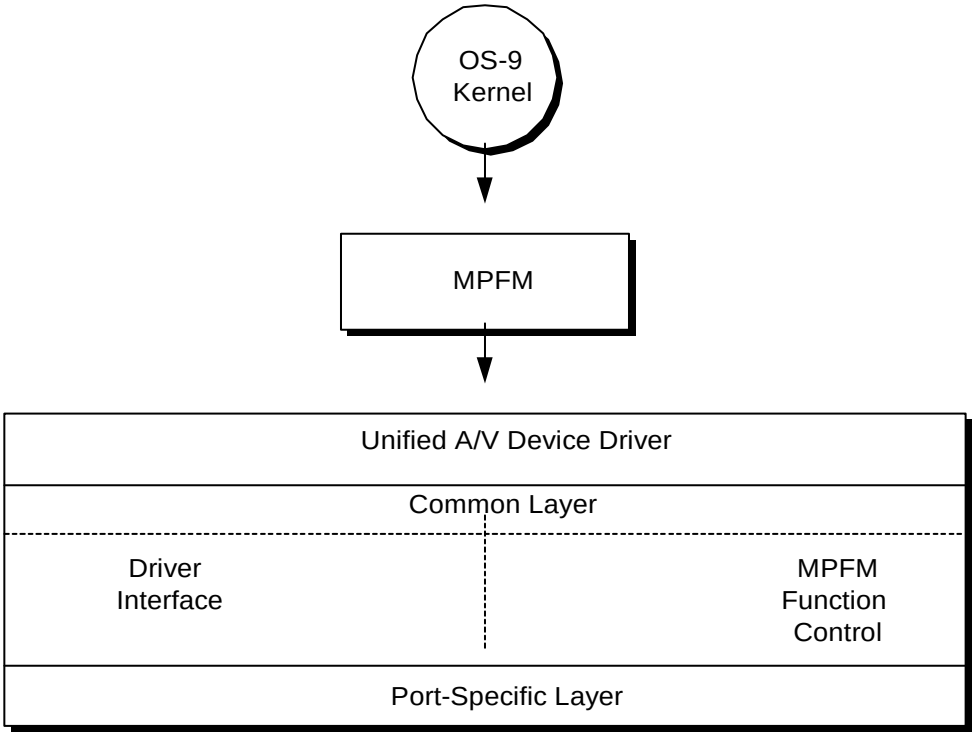
The other part of the device driver and the initialized values of the two device descriptors are different for different ports.

This chapter discusses the functions of the unified audio/video device driver. It breaks the MPFM drivers into two layers: the common layer and the port-specific layer. The common layer of the MPFM driver implements the functions of MPEG audio/video control that are common to all decoders on all hardware platforms. These functions can be divided into the following categories:

- Driver interface to the file manager
- MPFM function control

The port-specific layer of MPFM drivers handles the functions that directly control the decoder hardware. The common layer is built above the port-specific layer as illustrated in **Figure 1-1**:

Figure 1-1 MPEG Driver Layers



To port the MPFM driver to different hardware and environments, some knowledge of the MPFM device driver's internal structure and the structure's relationships is in order.

The device driver has one static storage, and each logical unit (`ma` and `mv`) has its own static storage. All these static storage areas are shared between common code and port-specific code. The complete definition of driver static storage is done in the hardware layer header file, most likely in `drvstat.h`. It includes a section that is common to all drivers and some port-specific definitions. The common code of the driver accesses only this

common section of the static storage, while port-specific code can access any field. The actual declaration and initialization of this static storage are also done in the port-specific layer, mostly in the file `hdr_stat.c`.

It is a requirement that all fields be initialized properly, or unpredictable behavior may occur. Within the driver's static storage, a pointer has been defined for each of the two logical unit static storage areas. A logical unit's static storage contains information that is specific to the related logical unit. It is initialized with the data stored in the related device descriptor (`ma` or `mv`) when the device is initialized.

Unified Audio/Video Device Driver

The MPFM audio/video device driver has two sections, the common layer and the port-specific (hardware) layer.

Common Layer Responsibilities

The common layer is comprised of software that is not hardware-specific. The common layer includes the driver interface and MPFM function control software.

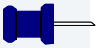
Driver Interface

The common-layer driver interface software implements the pre-processing of each MPFM call passed from the file manager. Pre-processing activities include:

- Allocating and de-allocating memory for the MPEG co-processor and unit buffers.
- General purpose initialization/de-initialization of the MPFM driver. For example, the driver interface allocates memory used by the common layer.
- Dispatching MPFM audio/video calls to the MPEG hardware layer.

Note

RAM that is accessible to the system when MPEG decoding is not required must have an entry in the system memory list and have the color `MPEG_VIDRAM (0x90)`. The audio and video descriptors must have the appropriate size and color entries detailing the size of this RAM. The common-layer driver interface software allocates and de-allocates the `MPEG_RAM` at initialization and termination.



Note

There must be a hardware direct data link between the network interface and the decoders through a demultiplexing ASIC (play from network mode).

MPFM Function Control

The MPFM function control is part of the common layer and is responsible for:

- Process driven interactive controls
- Audio/video play controls
- Dispatching the hardware-specific elements of these controls to the hardware layer

Port-Specific Layer Responsibilities

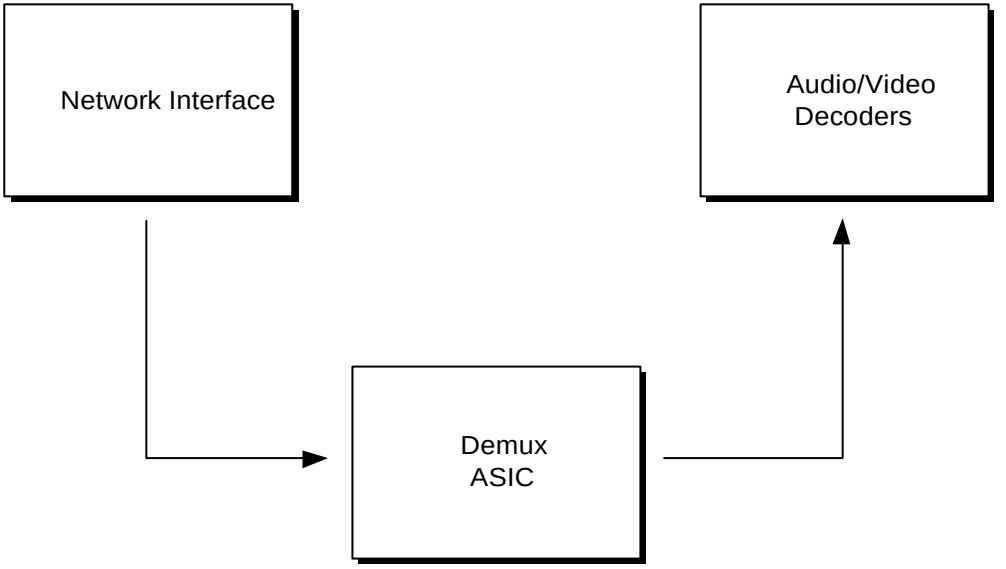
The hardware-layer is port-specific. It is responsible for:

- Hardware initialization and de-initialization
- Communication with hardware
- Implementation of hardware command and control
- Hardware-specific audio/video synchronization, if needed

MPEG Data Path

There must always be a direct data link between the MPEG demultiplexer and the MPEG decoders. The data path for MPEG is illustrated in **Figure 1-2**.

Figure 1-2 MPEG Data Hardware Paths



.

Synchronization Scheme

MPFM adopts a master-clock-driven approach to synchronize the audio and video streams. The master clock can be an external clock locked to the Program Clock Reference (PCR) of the incoming MPEG-2 transport stream, the System Clock Reference (SCR) of the MPEG-1 system stream, or even the audio sample clock. The master clock should meet the requirement for the clock frequency resolution specified in the MPEG specification. In this way, audio and video playback speed is controlled by the Presentation Time Stamps (PTS) in the stream based on the master clock such that audio and video are synchronized to each other.

The specific algorithm used for audio/video playback synchronization is port dependent. MPEG data is delivered directly from the demultiplexer ASIC; the time-stamp information, such as PCR, PTS, and Decoding Time Stamp (DTS) data can be received by calling corresponding DUXMan system calls.

For More Information

Refer to ***Using DUXMan*** for more information on the DUXMan system calls.

Other Hardware Actions

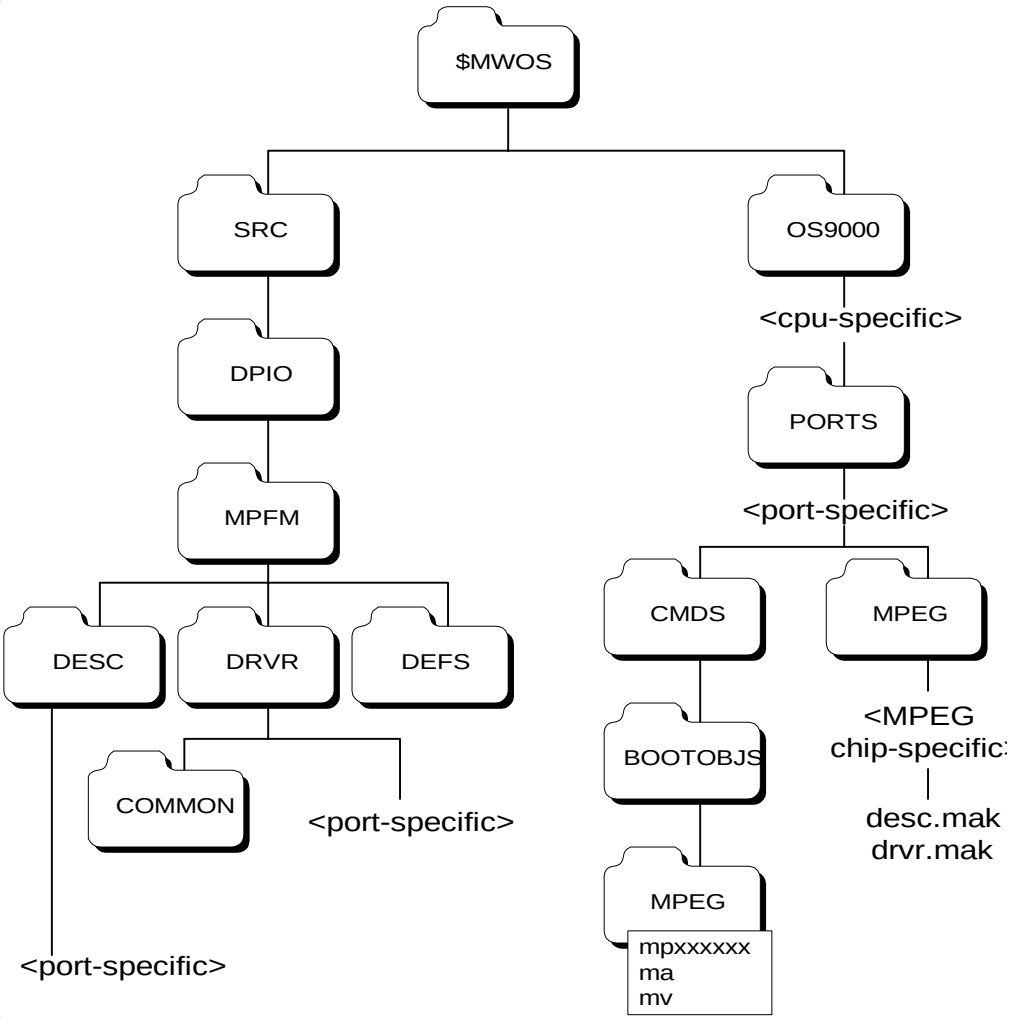
The hardware level code must also maintain the current MPEG Audio and Video Map Descriptor (`mad` and `mvd`). Some fields in `mad` and `mvd` should be updated frequently, such as:

- `md_picrt`
- `md_tmpref`

Directory Structure

The MPFM directory structure is shown in **Figure 1-3**. (**NOTE:** You may have a different CPU directory.)

Figure 1-3 The MPFM Directory Structure



Customizing Descriptors

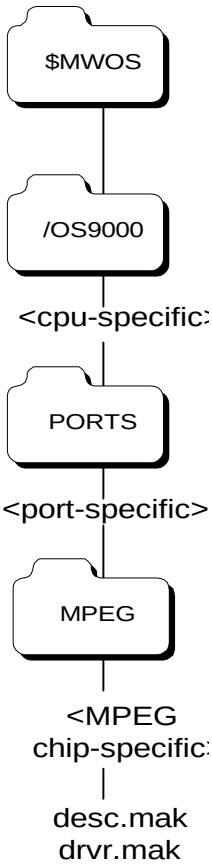
The unified audio/video device driver has two logical units associated with it, the audio and video devices. Two OS-9 descriptors are used, one for each unit.

This section discusses the descriptor fields and how to customize the sample descriptors to fit your hardware decoders.

Directory Structures and File Locations

To customize the MPFM descriptors, you must change the configuration file, `desc.h`. Next, use the makefile, `desc.mak`, to make the descriptors. The location of these files is as shown in **Figure 1-4**:

Figure 1-4 Directory Structure for Configuration Files



Header File desc.h

The header file `desc.h` contains define statements that specify the interrupt vectors that the driver and file manager must use.

The section beginning at the line:

```
#define DRIVERNAME “mpxxxxx”
```

must be customized for the target system. The various fields to customize are described in **Table 1-1**.

Table 1-1 mpfm.h Fields

Symbol Name	Symbol Value
`AUDIO_BUFCOLOR`	Memory color to use when allocating audio packet buffers
`AUDIO_DMA_CHANNEL`	Direct Memory Access (DMA) channel for transferring data to the Audio decoder
`AUDIO_IRQLEVEL`	Interrupt level generated by the audio hardware
`AUDIO_MAP_COLOR`	Memory color of RAM allocated for audio map descriptors
`AUDIO_MAXPACKSIZE`	Maximum audio packet size
`AUDIO_MPEG_COLOR`	Color of audio decoder RAM, if accessible to the CPU
`AUDIO_MPEG_SIZE`	Size of the audio decoder RAM area which is accessible to the system. If not applicable, use `0`.
`AUDIO_PORT_ADDRESS`	Audio hardware port address

Table 1-1 mpfm.h Fields (continued)

Symbol Name	Symbol Value
AUDIO_PRIORITY	Audio interrupt polling priority level
AUDIO_UCODE_MOD	Name of the module containing the audio microcode or a blank string if none is required
DATA_DELIVERY_TYPE	DDT_ELEM, DDT_PACKET or DDT_PACK. See the **Modes** section in this chapter.
DMA_SIZE	DMA transfer size
DRIVERNAME	The name chosen for the driver. Generally the MPFM driver names start with mp and include letters that identify the chip name.
FILEMANAGERNAME	The file manager name for the driver is mpfm
TIME_IRQLEVEL	Interrupt level of the timer
TIME_PORT	Port address of the timer
TIME_PRIORITY	Polling priority of the timer
TIME_SOURCE	VIDEO_TIMESRC, AUDIO_TIMESRC, or OTHER_TIMERSRC to indicate the timer source
TIME_VECTOR	Interrupt vector of the timer
VIDEO_BUFCOLOR	Memory color to use when allocating video packet buffers

Table 1-1 mpfm.h Fields (continued)

Symbol Name	Symbol Value
VIDEO_DMA_CHANNEL	DMA channel for transferring data to the video decoder
VIDEO_IRQLEVEL	Interrupt level generated by the video hardware
VIDEO_MAP_COLOR	Memory color of RAM allocated for video map descriptors
VIDEO_MAXPACKSIZE	Maximum video packet size
VIDEO_MPEG_COLOR	Color of video decoder RAM, if accessible to the CPU; normally MPEG_VIDRAM (0x90)
VIDEO_MPEG_SIZE	Size of the video decoder RAM area that is accessible to the system. If not applicable, use 0.
VIDEO_PORT_ADDRESS	Video hardware port address
VIDEO_PRIORITY	Video interrupt polling priority level
VIDEO_UCODE_MOD	Name of the module containing the video microcode or a blank string if none is required

Modes

An MPEG audio/video decoder may accept data with some levels of MPEG-1 system-layer information or MPEG-2 system-layer information stripped away. Depending on the hardware, a decoder may accept elementary streams or PES streams.

Making the Descriptors

After `desc.h` is customized to suit the specific hardware, create the descriptors by running `os9make` with the following commands:

```
$ cd $MWOS/OS9000/821/PORTS/HELLCAT/MPEG/MPCL9100
$ os9make -uf=desc.mak
```

The newly-created descriptors will be located in:

```
$ $MWOS/OS9000/821/PORTS/HELLCAT/CMDS/BOOTOBJS/MPEG
```

Chapter 2: Function Reference

There are two types of MPFM functions: common layer functions and hardware functions. Common layer functions are called by the hardware functions, so there is no need to know the specifics of those functions. This chapter provides a reference to the MPFM **Hardware Functions**.

Hardware Functions

The MPFM hardware functions are summarized in the following table. These functions are called by the related common layer functions. To port the MPFM, you must adjust each function to match your hardware. See the following pages for detailed descriptions of the functions.

Table 2-1 Summary of the MPFM Hardware Functions

Function	Description
`hw_aud_abort()`	Aborts Decoding in the MPEG Audio Hardware
`hw_aud_get_attenval()`	Determines Current Attenuation
`hw_aud_get_dsc()`	Gets Current Audio Decoder System Clock
`hw_aud_get_header()`	Gets Audio Frame Header Information
`hw_aud_get_stream()`	Gets Audio Stream Being Decoded
`hw_aud_init()`	Initializes Driver-Specific Audio Hardware
`hw_aud_mute()`	Mutes MPEG Audio Output
`hw_aud_play()`	Starts Audio MPEG Decoder Hardware
`hw_aud_term()`	De-initializes Driver-Specific Audio Hardware
`hw_aud_trigger()`	Defines Audio Trigger Events
`hw_aud_unmute()`	Un-mutes Audio

Table 2-1 Summary of the MPFM Hardware Functions

Function	Description
`hw_drv_init()`	Initializes Driver-Specific Hardware
`hw_drv_term()`	De-initializes Driver-Specific Hardware
`hw_getstat()`	Dispatches Driver-Specific GetStat
`hw_setstat()`	Dispatches Driver-Specific SetStat
`hw_vid_abort()`	Aborts Decoding in MPEG Video Hardware
`hw_vid_at_config()`	Sets Up Video Anti-Taping Configuration
`hw_vid_at_off()`	Turns Off Video Anti-Taping
`hw_vid_at_on()`	Turns On Video Anti-Taping
`hw_vid_blank()`	Blanks the Video Screen
`hw_vid_cc_off()`	Turns Off Video Closed-Caption
`hw_vid_cc_on()`	Turns On Video Closed-Caption
`hw_vid_decinit()`	Initializes Video Decoder
`hw_vid_get_dsc()`	Gets Current Video Decoder System Clock
`hw_vid_get_picrate()`	Gets Picture Display Rate
`hw_vid_get_stream()`	Gets Video Stream Being Decoded
`hw_vid_get_tempref()`	Get the Temporal Reference of the Current Picture

Table 2-1 Summary of the MPFM Hardware Functions

Function	Description
`hw_vid_init()`	Initializes Driver-Specific Video Hardware
`hw_vid_play()`	Starts Video MPEG Decoder Hardware
`hw_vid_set_border()`	Sets Border Color and Size
`hw_vid_show()`	Turns On Video Display
`hw_vid_term()`	De-initializes Driver-Specific Video Hardware
`hw_vid_trigger()`	Defines Video Trigger Events

hw_aud_abort()

Aborts Decoding in the MPEG Audio Hardware

Syntax

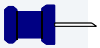
```
#include <mpfm_sys.h>
error_code hw_aud_abort(void);
```

Description

`hw_aud_abort()` aborts the decoding of audio data in the MPEG audio hardware.

`hw_aud_abort()` does the following:

- Sends an abort command to the audio decoder
- Flushes any data that was sent to the hardware but not yet played
- Mutes the output



Note

`hw_aud_abort()` does not completely de-initialize the audio decoder.

Direct Errors

OS-9 error code or SUCCESS (0) if no error occurred

Called By

MPEG level function `ma_abort()`

hw_aud_get_attenval()

Determines Current Attenuation

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_get_attenval(u_int32 *attenval);
```

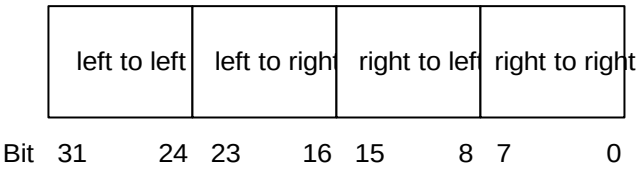
Description

`hw_aud_get_attenval()` determines the current audio attenuation by querying the audio hardware.

Parameters

`attenval`	Contains address to store the current attenuation value The attenuation value (`attenval`) format is shown in the figure below.

Figure 2-1 Attenuation Value Format



Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called by

Common layer function `ma_status()`

hw_aud_get_dsc()

Gets Current Audio Decoder System Clock

Syntax

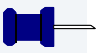
```
#include <mpfm_sys.h>
error_code hw_aud_get_dsc(u_int32 *dsc);
```

Description

`hw_aud_get_dsc()` retrieves the Decoder System Clock (DSC) from the audio decoder hardware and stores it at the location pointed to by `dsc`.

Parameters

`dsc`	Points to the location where the Audio Decoder System clock is to be stored



Note

Audio decoders not supporting this operation should return zero.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `ma_status()`

hw_aud_get_header()

Gets Audio Frame Header Information

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_get_header(u_int32 *hdr);
```

Description

`hw_aud_get_header()` retrieves the MPEG audio frame header information from the audio decoder hardware and stores it at the location pointed to by `hdr`.

Parameters

`hdr`	Points to location where the frame header information is to be stored

Note

Audio decoders not supporting this operation should store zero in the location pointed to by `hdr` and return SUCCESS.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `ma_status()`

hw_aud_get_stream()

Gets Audio Stream Being Decoded

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_get_stream(u_int16 *stream);
```

Description

`hw_aud_get_stream()` returns the number of the audio stream currently being decoded.

If the decoder does not provide the stream number, `hw_aud_get_stream()` should return the value in the current audio map descriptor.

Parameters

`stream`	Contains the address where the stream number is to be stored

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_status()`

hw_aud_init()

Initializes Driver-Specific Audio Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_init(Mp_lu_stat lua_statics);
```

Description

`hw_aud_init()` initializes variables specific to the driver's audio unit hardware. More specifically, it performs the following actions as appropriate:

- Resets the interface to the chip
- Installs interrupt service routine
- Downloads microcode for the chip
- Sets the hardware to the proper state
- Sets up the decoding mode to accept either PES stream or elementary stream.

Parameters

`lua_statics`	Points to the logical unit static storage for the audio unit

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer function `drv_init()` after `hw_drv_init()` has been called

hw_aud_mute()

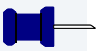
Mutes MPEG Audio Output

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_mute(void);
```

Description

`hw_aud_mute()` mutes the volume in the MPEG audio hardware.



Note

A distinction is made between muting and lowering the volume to nil, in particular, maximum attenuation need not necessarily be zero volume.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `sna_play()`

hw_aud_play()

Starts Audio MPEG Decoder Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_play(void);
```

Description

`hw_aud_play()` starts the audio MPEG decoder hardware. For some hardware, it may be necessary to enable the data delivery from the demultiplexer at this point.

Direct Errors

OS-9 error code or `SUCCESS` (`0`) if no error occurred

Called By

Common layer function `sna_play()`

hw_aud_term()

De-initializes Driver-Specific Audio Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_term(mp_lu_stat *lua_statics);
```

Description

`hw_aud_term()` provides driver-specific de-initialization for the audio hardware.

`hw_aud_term()` de-initializes the audio unit hardware, releases dynamically allocated memory, removes interrupt request routines, and completely shuts down the audio decoder.

`hw_aud_term()` is called if `hw_aud_init()` fails. Therefore, it does not assume that the audio decoder is completely initialized.

Parameters

`lua_statics`	Points to the logical unit static storage for the audio unit

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer functions `drv_init()` or `drv_term()`

hw_aud_trigger()

Defines Audio Trigger Events

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_trigger(u_int16 statmask);
```

Description

`hw_aud_trigger()` sets the driver's trigger mask.

The driver sets up the corresponding interrupts and sends the appropriate signal when an event (for which a signal has been requested) occurs.

A copy of `statmask` is maintained in the logical unit structure (`v_statmask`). The driver sends the appropriate signal(s) to the application requesting them when it is notified of their occurrence.

Parameters

`statmask`	Contains the signal mask. The `statmask` format is shown in the following figure.

Figure 2-2 `statmask` Format

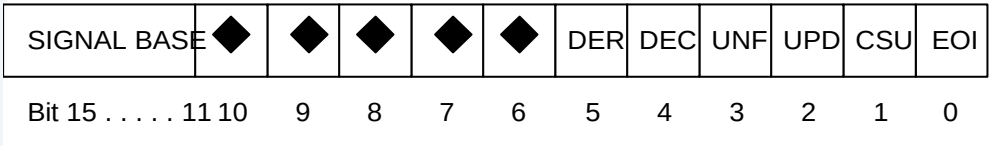


Table 2-2 `statmask` Format

Bit	Code	Description
15...11		Signal base: upper 5 bits of the 16-bit signal to send (value must be between 00001 and 11111 binary)
6...10		Reserved, must be zero
5	DER	Data error received during play
4	DEC	Decoder started decoding
3	UNF	Decoder does not have data to decode (underflow)
2	UPD	Decoder updated the frame header (see `ma_status`)
1	CSU	Decoder changed to a new audio stream
0	EOI	End of Program or End of Stream Code detected

hw_aud_unmute()

Un-mutes Audio

Syntax

```
#include <mpfm_sys.h>
error_code hw_aud_unmute(void);
```

Description

`hw_aud_unmute()` turns the MPEG audio back on following a mute command. The volume of the audio returns to the level that was set before the `hw_aud_mute()` command was called.

Direct Errors

OS-9 error code or `SUCCESS` (`0`) if no error occurred

Called By

Common layer functions `sna_play()` or `do_vid_start_play()`

hw_drv_init()

Initializes Driver-Specific Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_drv_init(Dev_list dev_entry);
```

Description

`hw_drv_init()` initializes any hardware or variables specific to the driver and common to both audio and video.

Note

Use `hw_aud_init()` and `hw_vid_init()`, respectively, to initialize hardware or variables specific to audio or video.

`hw_drv_init()` is called once when initializing the audio device and once when initializing the video device. If some actions need to take place only once, then `hw_drv_init()` takes note that it has already been called.

Parameters

`dev_entry`	Points to the device list entry for the unit being initialized.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer function `drv_init()`

See Also

`hw_aud_init()`
`hw_vid_init()`

hw_drv_term()

De-initializes Driver-Specific Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_drv_term(Dev_list dev_entry);
```

Description

`hw_drv_term()` de-initializes hardware that is specific to the driver but not specific to audio or video. The actions performed by this function are the counterpart to actions taken by `hw_drv_init()`.

Note

Use `hw_aud_term()` and `hw_vid_term()`, respectively, to de-initialize items specific to audio or video.

Parameters

`dev_entry`	Points to the device list entry for the unit being initialized

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer function `drv_term()`

See Also

`hw_aud_term()`
`hw_vid_term()`

hw_getstat()

Dispatches Driver-Specific GetStat

Syntax

```
#include <mpfm_sys.h>
error_code hw_getstat(
     I_getstat_pb ctrl_block,
     Mp_path_desc pd);
```

Description

hw_getstat() dispatches hardware-specific level GetStat calls to the appropriate functions that are specific to this hardware.

Parameters

ctrl_block	Points to the GetStat parameter block
pd	Points to the MPFM path descriptor

Direct Errors

OS-9 error code or SUCCESS (0) if no error occurred

EOS_UNKSVC returned if GetStat is unrecognized

Called By

Driver common layer function drv_getstat()

hw_setstat()

Dispatches Driver-Specific SetStat

Syntax

```
#include <mpfm_sys.h>
error_code hw_setstat(
     I_setstat_pb ctrl_block,
     Mp_path_desc pd);
```

Description

hw_setstat() dispatches hardware-level SetStat calls to the appropriate functions that are hardware dependent.

Parameters

ctrl_block	Points to the SetStat parameter block
pd	Points to the MPFM path descriptor

Direct Errors

OS-9 error code or SUCCESS (0) if no error occurred

EOS_UNKSVC returned if SetStat is unrecognized

Called By

Driver common layer function drv_setstat()

hw_vid_abort()

Aborts Decoding in MPEG Video Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_abort(void);
```

Description

`hw_vid_abort()` aborts decoding in the MPEG video hardware.

This function sends an abort command to the video decoder, flushes any data that was read but not played, and freezes the display. It does not perform a complete de-initialization of the video decoder.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_abort()`

hw_vid_at_config()

Sets Up Video Anti-Taping Configuration

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_at_config(
     u_char*      key,
     u_int32      keylen,
     u_char       *confstr,
     u_int32      strlen);
```

Description

`hw_vid_at_config()` sets up the initial configuration at the hardware-layer level for the video output anti-taping.

If the anti-taping hardware is not inside the video decoder (such as inside an NTSC encoder), this function can call other device drivers or managers to implement the required configuration.

Parameters

`key`	Contains the key string for authentication
`keylen`	Contains the key string length
`confstr`	Contains the anti-taping configuration string
`strlen`	Contains the configuration string length

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_at_ctrl()`

hw_vid_at_off()

Turns Off Video Anti-Taping

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_at_off(
     u_char       *key,
     u_int32      keylen);
```

Description

`hw_vid_at_off()` turns off the anti-taping function in the hardware.

If anti-taping hardware is not inside the video decoder (such as inside an NTSC encoder), this function may call other device drivers or managers to turn off the anti-taping function.

Parameters

`key`	Contains the key string for authentication
`keylen`	Contains the key string length

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_at_ctrl()`

hw_vid_at_on()

Turns On Video Anti-Taping

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_at_on(
     u_char*      key,
     u_int32      keylen,
     u_int32      mode);
```

Description

`hw_vid_at_on()` turns on the anti-taping function in the specified mode in hardware.

If the anti-taping hardware is not inside the video decoder (such as inside an NTSC encoder), this function can call other device drivers or managers to turn on the anti-taping function.

Parameters

`key`	Points to the key string for authentication
`keylen`	Contains the key string length
`mode`	Contains the anti-taping modes

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_at_ctrl()`

hw_vid_blank()

Blanks the Video Screen

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_blank(u_int32 color);
```

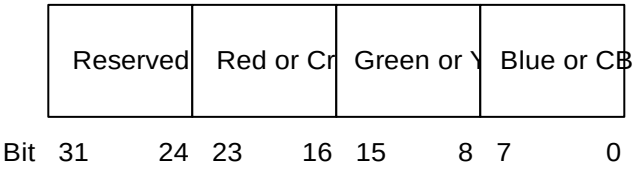
Description

`hw_vid_blank()` clears the full-motion video plane to the color specified.

Parameters

`color` — YUV- or RGB-value color format is shown in the following figure.

Figure 2-3 YUV or RGB Color Format



Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_hide()`

hw_vid_cc_off()

Turns Off Video Closed-Caption

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_cc_off(void);
```

Description

`hw_vid_cc_off()` turns off the video closed-caption output in hardware.

This function also stops the extracting and parsing of the closed-caption data.

If closed-caption control hardware is not present inside the video decoder (such as an NTSC encoder), this function may call other device drivers or managers to turn off the closed-caption function.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer `drv_setstat()` function

hw_vid_cc_on()

Turns On Video Closed-Caption

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_cc_on(void);
```

Description

`hw_vid_cc_on()` turns on the video closed-caption output in the video decoder hardware.

This function also starts the extracting and parsing of the closed-caption data from the MPEG stream. It may also need to enable the V-SYNC interrupt in the system to synchronize the delivery of closed captioning data.

If the closed-caption control hardware is not present inside the video decoder (such as an NTSC encoder), this function may call other device drivers or managers to turn on the closed-caption function.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer `drv_setstat()` function

hw_vid_decinit()

Initializes Video Decoder

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_decinit(void);
```

Description

`hw_vid_decinit()` initializes or re-initializes the video decoder (performs a “warm boot” of the video decoder).

This function is provided for video decoders to use when starting a play. For some hardware devices, this function need only return SUCCESS.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_play()`

See Also

`hw_vid_play()`

hw_vid_get_dsc()

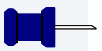
Gets Current Video Decoder System Clock

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_get_dsc(u_int32 *dsc);
```

Description

`hw_vid_get_dsc()` retrieves the video decoder hardware Decoder System Clock (DSC) and stores it at the location pointed to by `dsc`.



Note

Video decoders not supporting this operation should set `*dsc` to `0` and return SUCCESS.

Parameters

`dsc`	Points to the location where the Video Decoder System clock is to be stored

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_status()`

hw_vid_get_picrate()

Gets Picture Display Rate

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_get_picrate(u_char *picrate);
```

Description

`hw_vid_get_picrate()` retrieves the current picture display rate, in frames per second.

On some hardware, this value is available in a register, for other hardware types, it may not be available.

If the display rate is unavailable on your hardware, this function returns 0.

Parameters

`picrate`	Points to the picture rate

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_status()`

hw_vid_get_stream()

Gets Video Stream Being Decoded

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_get_stream(u_int16 *stream);
```

Description

`hw_vid_get_stream()` returns the number of the video stream currently being decoded.

If the decoder accepts only elementary streams or it does not provide the stream number, `hw_vid_get_stream()`returns the value in the current video map descriptor.

Parameters

`stream`	Contains the address where the stream number is stored.

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_status()`

hw_vid_get_tempref()

Get the Temporal Reference of the Current Picture

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_get_tempref(u_int16 *tempref);
```

Description

This function returns the temporal reference of the current picture.

Parameters

`tempref`	Points to temporal reference number

hw_vid_init()

Initializes Driver-Specific Video Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_init(Mp_lu_stat luv_statics);
```

Description

`hw_vid_init()` initializes the hardware and variables specific to the driver's video unit hardware. This function performs the following actions as appropriate:

- Sets up port address for the video chip
- Installs interrupt service routines
- Initializes the video decoder
- Sets up interface registers
- Downloads microcode for video chip
- Sets up interface with demultiplexer

Parameters

`luv_statics`	Points to the video logical unit static storage

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer function `drv_init()`

hw_vid_play()

Starts Video MPEG Decoder Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_play(u_int16 speedval);
```

Description

`hw_vid_play()` enables and starts video decoding in the MPEG video hardware.

Parameters

`speedval`	Always given a value of 0 (full speed play)

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `snv_play()`

hw_vid_set_border()

Sets Border Color and Size

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_set_border(
     u_int32      scroff,
     u_int32      colorval,
     u_char       scrflag);
```

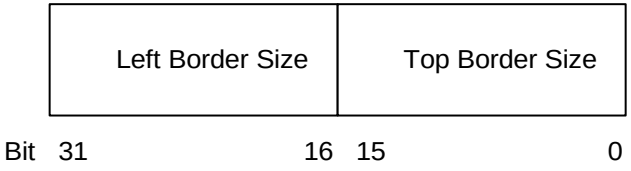
Description

`hw_vid_set_border()` sets the border color and size of the full-motion video plane.

Parameters

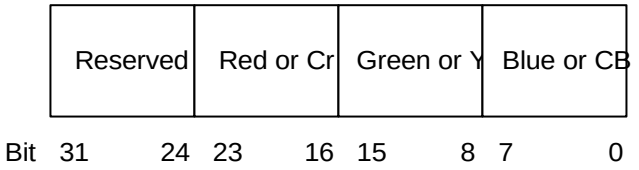
`scroff` — Contains the size of the display plane. The right and bottom border sizes are determined by the window's left and top border sizes.The scroff format is shown in the following figure.

Figure 2-4 scroff Format



`colorval` Contains the color mix values for the display plane border. The `colorval` format is shown in the following figure.

Figure 2-5 colorval Format



Direct Errrors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer functions `mv_bcolor()`, `snv_play()`, or `mv_pos()`

hw_vid_show()

Turns On Video Display

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_show(void);
```

Description

This function turns the output of the video decoder to `ON`.

Direct Errors

OS-9 error code or `SUCCESS` (`0`) if no error occurred

Called By

Common layer function `mv_show()`

hw_vid_term()

De-initializes Driver-Specific Video Hardware

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_term(Mp_lu_stat luv_statics);
```

Description

`hw_vid_term()` de-initializes the video hardware, which may involve removing the interrupt service routine and removing the events that were registered in `hw_vid_init()`.

Parameters

`luv_statics`	Points to the logical unit static storage for the video unit

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Driver common layer functions `drv_init()` or `drv_term()`

hw_vid_trigger()

Defines Video Trigger Events

Syntax

```
#include <mpfm_sys.h>
error_code hw_vid_trigger(u_int16 statmast);
```

Description

`hw_vid_trigger()` sets the driver's trigger mask.

The driver sets the interrupts and sends the appropriate signal when an event occurs for which a signal has been requested.

A copy of `statmask` is maintained in the logical unit statics (`v_statmask`). The driver sends the appropriate signal(s) to the application requesting them, when it is notified of their occurrence.

Parameters

`statmask` Contains the signal mask. The `statmask` format is shown in the following table.

Table 2-3 statmask Format

Bit	Type	Description
0	DER	Signal when data error detected
1	PIC	Signal on picture displayed
2	GOP	Signal on group of pictures
3	SOS	Signal on start of sequence
4	LPD	Signal when last picture is displayed
5	CNP	Not used
6	EOI	Signal when end of ISO header detected at input
7	EOS	Signal when end of sequence detected at input
8	BUF	Signal when buffer under-flow detected
9	NIS	Signal when new sequence parameters are found
10		Reserved and must be zero
15...11		Signal base: upper 5 bits of 16-bit signal to send (value must be between 00001 and 11111 binary

Direct Errors

OS-9 error code or SUCCESS (`0`) if no error occurred

Called By

Common layer function `mv_trigger()`

Index

A

B

C

D

E

F

G

H

I

L

M

O

R

S

T

U

V

W

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:_____

Company:_____

Phone:_____

Fax:_____Email:_____

Product Name: MPFM

Description of Problem:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Host Platform_____

Target Platform_____

RadiSys.

MICROWARE SOFTWARE