MangoAV_HW

User Guide

Release 1.9

Legal Information

Copyright

Disclaimer

Mango DSP Ltd. reserves the right to make changes in specifications at any time without notice. The information furnished by Mango DSP in this material is believed to be accurate and reliable. However, Mango DSP assumes no responsibility for its use.

Trademarks

Mango DSP is a trademark and "Processing the Digital Vision" is a service mark of Mango DSP Ltd.
Windows, the Windows logo, Windows 98/2000/Millenium/XP, and Windows NT, are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
Apple, the Apple logo, and QuickTime are trademarks of Apple Computer, Inc., registered in the U.S. and other countries.
Linux is a registered trademark of Linus Torvalds.
Other product and company names mentioned herein may be the trademarks of their respective owners.

Contact Information

Mango DSP Ltd.

Postal: Har Hotzvim Industrial Park
P.O. Box 45116
Jerusalem, 91450 Israel

Tel: +972-2-588 5000
Fax: +972-2-532 8705

Detailed Revision History

Rev.	Date	Author	Approved	Description
1	14/11/04	Itay Chamiel		Preliminary Release
2	20/9/05	Itay Chamiel		Updates

1. Introduction

The MangoAV Hardware support library is a DSP side library responsible for abstracting access to audio, video and other peripherals. The API is identical regardless of which card you are using, although the internal implementation may be completely different. Note that not all functionality is supported for all cards; for example the Phoenix (PC-104) board does not have a video output.

The important functions of this library include opening of access to video/audio inputs and outputs, and sending and receiving data from them. The library can also enable access to extra peripherals such as TTL inputs/outputs, if your card has them.

2. Setting up your project

Which MangoAV_HW?

There are several different flavors of the MangoAV_HW library, each suitable for a different range of cards. Their API is identical, but the library file you must include with your project is different. Here is a table listing the various cards and the suitable version.

Card	Lib name
Seagull PMC	MangoAV_HW_C6415
Lark	MangoAV_HW_C6415
Phoenix (PC-104)	MangoAV_HW_DM
Raven-X	MangoAV_HW_DM
Raven-HX	MangoAV_HW_DM
Raven-PH	MangoAV_HW_DM
Raven-D	MangoAV_HW_C6412
Raven-V	MangoAV_HW_C6412
Raven-C	MangoAV_HW_C6412

MangoAV_HW_Init

The first thing your application must do is initializing the MangoAV_HW library, which is done by calling the function MangoAV_HW_init. Some of the things this function does must occur very early, even before your main function (before some DSP/BIOS initializations). You should therefore place the call to this function within a function of your own, named (for example) SYSTEM_Init.

To cause SYSTEM_Init to be called, open your project CDB file. If there are any entries under Input/Output>Device Drivers>User-Defined Devices (as there should be if your system uses the Vport or the FPGA Stream), set _SYSTEM_Init (note the underscore prefix) as the init function of one (and not more) of the entries. If not (and only if not), go to System>Global Settings and set the User Init Function to _SYSTEM_Init.

Using the Global Settings is the less recommended method because in this way your function will be called before the system heaps are available, while MangoAV_HW needs these heaps to allocate data. On some cards you will have no choice but to use this method (such as Seagull PMC). However, the MangoAV_HW version for these cards was written with this fact in mind.

The contents of SYSTEM_Init will be discussed shortly.

Additional libraries required

Some Mango boards have a basic driver used to access the card's various hardware features such as configuring the on-board FPGA. You must include the appropriate library in your project. If such a lib exists, it will have the same name as the board, for example Raven_D.lib.
The DM642-based boards (Raven-X, Phoenix) require the Vport.lib library to be included and installed in your CDB.
Boards that use the FPGA for audio/video data transfer may need the FPGA Stream library present and installed in the CDB.

Internal and external heap

The library requires pointers to two global ints, named in this example `seg_isram` and `seg_sdram`. They must contain valid memory segment identifiers, contained in internal and external memory, respectively. Normally your CDB will contain two MEM objects named ISRAM and SDRAM, and these will each contain a heap. In such a case the following sample code can be used.

System_INIT

Here is a sample System_INIT, suitable for the Raven D.

```
// globals
int seg_isram, seg_sdram;
Raven_D_handle_t raven_d;

void SYSTEM_Init()
{
	int my_dsp_num;

	seg_isram = ISRAM; // defined in CDB
	seg_sdram = SDRAM;

	if (Raven_D_Open(&raven_d, NULL) != MANGOERROR_SUCCESS)
		while(1);
	my_dsp_num = !(raven_d.loads_fpga);

	// Open the GPIO handle
	gpio_handle = GPIO_open(GPIO_DEV0, GPIO_OPEN_RESET);

	MangoAV_HW_init(MANGOCARD_RAVEND, my_dsp_num, gpio_handle, &raven_d,
			&seg_isram, &seg_sdram);
}
```

First we assign the memory segment identifiers. Note for Seagull PMC/Lark: If System_INIT is called by the Global module (as explained above), these lines belong in `main` and not here! This will mean that the contents of seg_isram and seg_sdram will not be valid at the time of the call to MangoAV_HW_Init, but in this case the init function will not use the contents but only store a copy of the pointer for future use.

If your card uses a driver, it is initialized here by opening the card handle.

The next line determines the DSP number. Since the Raven-D has two DSPs, only one of which can load the FPGA, the easiest way to know which DSP we are is to check whether the FPGA can be loaded by the current DSP. The card driver checks for this capability.

The next line opens the GPIO handle. Note that on some cards this call will fail! This is because the card driver has already opened the GPIO handle for its own internal use. In this case you do not open the handle yourself, but rather use the handle opened by the driver, accessible using `<card_handle>.GPIO_handle`.

Finally, MangoAV_HW_init is called, with the card type, DSP number, GPIO handle and card handle. The card handle may be NULL on those boards that don't require a driver.

Why is the card type necessary? As shown in the table above, one library version can suit several different cards. The library has different behavior on different cards, and has no other way to know which card type it is operating on. (This cannot be inferred from the card handle, as it is not always given).

main()

Your main() function also needs some additions. First, you should repeat these two lines from System_INIT:

```
seg_isram = ISRAM; // defined in CDB
seg_sdram = SDRAM;
```

They are needed here because on some systems the ISRAM and SDRAM values only become valid in main().
The second change is that you must call MangoAV_HW_init_main(). This initializes hardware components such as the Video Port that cannot be initialized during System_INIT.

3. Working with the library

Once the library has been properly initialized, working with it is easy, as most details are taken care for you. See the chapter “Specifics for various cards” for information on any specific details needed by your card that aren’t directly covered by the lib. Also, please note that this is only an introduction – please see the complete reference guide for full information on the API.

The following examples should be performed within a task, not in your main function. This is because many of the functions are blocking and cannot be called from main.

Video

Here is some example code for receiving video:

```
{
	MangoAV_HW_open_cam_input(MANGO_AV_PAL,
		COMPOSITE_VIDEO,
		0, 3, MANGOBIOS_FOREVER);

	while(1)
	{
		MangoAV_HW_get_img_from_cam(videoBuff,
			MANGO_AV_4CIF_INTERLEAVED_PAL,
			0,
			MANGOAV_HW_420PLANAR,
			&time_tag,
			scratch_buf,
			scratch_share_SEM);

		// process video
	}
}
```

The first line opens a video input. Here we ask for a connection to open for a PAL source format, through a composite video input (there are some custom boards built with S-Video connectors instead of composite; in that case use S_VIDEO). The next parameter is the camera number. Note that this parameter is the input number relative to the current DSP. For example, on a board that has four inputs and two DSPs, where each DSP can access two of the inputs, the third and fourth inputs will be numbered 0 and 1 on the second DSP.
The next parameter indicates the number of buffers allocated for video input. Each buffer contains a single video frame, so this parameter determines the amount of buffers that can be queued before frame loss occurs. See the reference manual for allowed values.
The final parameter indicates the timeout for reeciving frames. `MANGOBIOS_FOREVER` will cause the get image function to block indefinitely until a frame is available. Any other value will indicate the number of clock ticks (normally milliseconds) to wait, at which point a timeout value will be returned if a frame was not available. See the reference guide for full details.

Inside your loop, calling `MangoAV_HW_get_img_from_cam` receives the next frame from the video source. Its first parameter is a buffer to which the image, properly sized and separated to 4:2:0, will be copied. The size of this buffer should be: *width* x *height* x 1.5 bytes. The second parameter specifies the requested size of the image. The third parameter is the camera number, which must have been opened

previously with the open_cam function. The fourth parameter is the image format; this usually cannot change, see the reference for details. The fifth parameter is a pointer to a `Time_tag_T` struct. Upon succesfully returning from this function, the struct will contain the relative time at which the frame was completely received from the video input (which could be before or after the call to `MangoAV_HW_get_img_from_cam`). This time tag has the resolution of `CLK_gethtime()`, but it adds a `t_msb` field thus giving a 64-bit value.
(**Note: the following paragraph is not relevant for DM-based systems.** Use `NULL`, `NULL` for the last two parameters.) The sixth parameter is a pointer to an ISRAM scratch buffer that the function needs for its processing. See the reference guide for the size you must allocate. Since this buffer is not required outside of this function call, it is free to be used by other tasks when this call is not busy. However, this presents a problem: it is quite likely that the function will block for a long time (up to almost 40 ms) before even needing the buffer, and it is therefore wasteful to block this buffer for so long. Therefore, the seventh parameter allows you to (optionally) supply a semaphore on which the function will pend when it is ready to process an image, so it can wait for its turn to use the shared scratch buffer. This semaphore will be posted before the function returns. You can use `NULL` if you do not need this functionality.

Outputting video is a similar process: Call `MangoAV_HW_open_tv_output` with the requested video standard and timeout, and then `MangoAV_HW_send_img_to_monitor` to output frames.

Audio

Here is some example code for receiving audio:

```
{
	MangoAV_HW_open_mic_input(0, 96000/sample_rate);

	while(1)
	{
		MangoAV_HW_get_audio_frame_from_mic(audiobuf, 0,
			1, &size, &time_tag);

		// process audio
	}
}
```

Opening the mic input is as simple as specifying the input number and the sample rate divider. For example, for 48000 Hz you would input 96000/48000 or 2.

To receive an audio frame, the parameters are: A pointer to your buffer; the input number; whether this input is stereo (a stereo input can be treated as two mono channels); an integer pointer that will be filled with the size of the returned buffer; and finally a time tag, similar to the video. The returned buffer is always 1152 samples long, with each sample being a 16-bit signed short. The buffer is doubled for a stereo input, with samples from the two channels interleaved.

Other functionality

The library provides additional functionality, depending on your hardware's capabilities. For example, if your card has TTL inputs/outputs, the **MangoAV_HW_get_ttl_in**, **MangoAV_HW_get_ttl_out** and **MangoAV_HW_set_ttl_out** functions may be used to access them.

On those cards that require it, **MangoAV_HW_C64_i2c_load** is used to configure the I2C. Also, **MangoAV_HW_set_cam_input_param** can be used to change video input parameters such as brightness, contrast and color. Note that these two functions can only be done from the DSP(s) responsible for I2C loading (see "Specifics for various cards" for details).
On the Raven-D, a general-purpose SIO data stream is supplied that can transfer data between the two DSPs; these streams may be accessed using **MangoAV_HW_C64_get_gpStreams**.
For standalone (non-PCI) boards, the **MangoAV_HW_resetBoard** function can be used to reset the board. This is identical to pressing the board's Reset button manually. It will physically reset the DSPs and cause them to reboot from Flash.
Finally, a few helpful **QDMA memory copy** functions are given. They may be used as a slightly faster (but less flexible) alternative to DAT_copy.

4. Application code limitations

The MangoAV_HW presents various limitations imposed by the DSP hardware architecture. These limitations must be respected by the application developer. Failure to adhere to these guidelines will cause various malfunctions such as flickering monitor output, failure to receive video correctly, etc.

- **Task Priority.** Priority 15 (`TSK_MAXPRI`) is reserved for use by the lib's generated tasks only. This is required so that events such as incoming video are handled promptly.
- **EDMA Priority.** If you are not satisfied by the supplied EDMA/QDMA functionality and wish to define your own, you must use the `EDMA_OPT_PRI_LOW` priority queue only. Using any other queue may interfere with the lib's functionality. If you wish to use DAT functions, open the module with low priority.
- **Interrupt Latency.** The lib's video and audio functionality cannot tolerate long periods in which interrupts and task switching are disabled. The maximum latency is about 1 ms. Here are a few points to keep in mind:
  - There are cases where an algorithm disables task switching for its own internal reasons using `HWI_disable` or `TSK_disable`. Keep such segments short.
  - Interrupts cannot be taken while the CPU is in the delay slots of a branch. This means that tight loops (4 cycles or less) are not interruptible at all. This will not occur when writing regular C code in Debug mode, but can occur when optimizing code or in Assembly. Use the Interrupt Threshold (`-mi`, we use `-mi128`) compiler option to prevent this in optimized C. (Assembly code must be checked and modified if needed.)
  - Many function calls in algorithm libraries such as img64x are not interruptible. If you use them you may have to split your calls into smaller ones.
  - Some functions from the Real Time Support (rts) package are not interruptible. This includes memcpy. Using memcpy on SDRAM addresses can block interrupts for too long if the transfer is over 100KB or so.
  - Some Chip Support Library (csl) functions can also cause problems. For example, DAT_copy2d can take an extremely long time if its parameters are not 4-byte aligned. A subsequent DAT_wait (or another DAT_copy, which may have to wait for the previous one to finish) will run in a tight loop until the transfer is finished. Note that this specific problem can be solved by recompiling the CSL with interrupt threshold enabled; see SPRU187 for information on the mk6x utility which you should use to compile csl6400.src. (Note that recompiling RTS cannot help as functions like memcpy are written in Assembly.)
- **Overloading of system buses.** The video and audio input and output is constantly operating in the background. Normally this shouldn't bother the application developer. However, an operation that makes extensive use of the EMIF bus may slow down the background operations, possibly to the point of data loss. Symptoms of this problem include corrupted frames coming from the video input, or problems in the video output such as a large vertical stripe appearing in the middle of the image. Causes can be things like very large and frequent QDMA memory moves, or very large transfers of data between the DSPs (on the Raven-D).
- **Overflowing the EDMA priority queue.** This is a rare condition that will not normally happen unless you use your own QDMA/EDMA access routines. It can occur if too many requests are submitted on a single priority queue. In this case, **ALL** EDMA operations are stalled until the

current operation completes, which can cause data loss in the background transfers. See SPRA994 section 6.3, "Priority Inversion Due to TR Stalls" for more details.

5. Specifics for various cards

This chapter deals with specific details regarding every card supported by this library. But first, some conventions:

- **I2C** – The Philips chips used to input and output analog video require initialization. This initialization is done by loading its registers via the Inter-Integrated Circuit (I2C) bus. The process will be referred to as "loading I2C" or just "I2C".
- **FPGA** – Some cards have an on-board FPGA that deals with data transfers required for audio and/or video. This FPGA must be configured by the DSP. The process will be referred to as "loading the FPGA". **Note:** The FPGA load must always occur during `main` or later, never during `System_INIT`!
- DSPs are numbered starting with 0 and count up. Their order is defined by their order in the JTAG.

Refer to the section relevant to the card you are developing on:

Seagull PMC

4 DSPs. 3 video inputs, 1 video output. Audio and TTL not currently supported.
DSP 0 loads FPGA, loads all I2C, and outputs to monitor.
DSPs 1-3 each receive one video input.
User is responsible for FPGA and I2C loading. This is not taken care of by the library. See example code for details.
Chip IDs for I2C: 0-2 are the video ins, 3 is the video out.

Lark

11 DSPs. 8 video inputs, 1 video output. Audio and TTL not currently supported.
DSP 0 loads FPGA, all I2C, outputs to monitor.
DSPs 1-8 each receive from one video input.
User is responsible for FPGA and I2C loading. This is not taken care of by the library. See example code for details.
Chip IDs for I2C: 0-7 are the video ins, 8 is the video out.

Phoenix (PC-104)

2 DSPs. 8 video inputs, of which 4 can be used at once. 8 stereo audio inputs.
4 cameras per DSP. Of each pair (0-1 and 2-3), only one can be used at once. (If you try receiving from both simultaneously, frame rate will drop to about 1 fps due to switching.)
(To identify the video inputs, look at the board with the video ports facing you and the DSP side facing down. If the board has 8 video connectors installed, from left to right, they are: DSP 0 cams 0, 1, 2, 3, DSP 1 cams 0, 1, 2, 3. If there are only 4 connectors, they are DSP 0 cams 0, 2, and DSP 1 cams 0, 2.)
4 stereo audio inputs per DSP.
I2C fully handled by library.

Raven-X

2 DSPs, 4 video inputs, of which 2 can be used at once. 4 stereo audio inputs.

2 cameras per DSP, only one of which can be used at once. (If you try receiving from both simultaneously, frame rate will drop to about 1 fps due to switching.)
2 audio inputs per DSP.
I2C fully handled by library.

Raven-HX (Half-X)

1 DSP, 2 video inputs, 1 video output. 1 stereo audio input, 1 stereo output.
Only one of the video inputs can be used at once. (If you try receiving from both simultaneously, frame rate will drop to about 1 fps due to switching.)
I2C fully handled by library.

Raven-PH

1 DSP, 2 video inputs, 1 video output. 1 stereo audio input, 1 stereo output.
Only one of the video inputs can be used at once. (If you try receiving from both simultaneously, frame rate will drop to about 1 fps due to switching.)
I2C fully handled by library.
Additional capabilities: UART capable of speeds up to 115200 baud, Real Time Clock.

Raven-D

2 DSPs. 4 video inputs, 1 video output, 2 mono audio outputs, 2 mono inputs.
2 cameras per DSP.
Video output from DSP 1.
All audio accessed by DSP 0.
I2C loaded with default (PAL) parameters at first lib access, any subsequent change must be done by user. All access is from DSP 0 only.
FPGA and I2C will be loaded automatically at the first time any lib function (other than MangoAV_HW_init) is called. If you want to force init, you can call a harmless function like MangoAV_HW_C64_get_ttl_in.
Chip IDs for I2C: 0-3 are the video ins, 4 is the video out.
Additional capabilities: UART capable of speeds up to 56600 baud, TTL in/out.

Raven-V

4 DSPs. 4 video inputs, 4 audio mono inputs.
1 camera per DSP.
1 audio in per DSP.
User is responsible for FPGA loading.
I2C fully handled by library.

Raven-C

1 DSP. 1 audio mono input, 1 output. 2 video outputs.
Both video outputs must be initialized (I2C) but only one may be accessed normally (the other sends video from the FPGA).
User is responsible for FPGA and I2C loading.
Chip IDs for I2C: 0-1 are the video outs.