i.MX50 ePxP User Library

User’s Guide

Document Number: 09-7521-USRGD-ZCH70
Rev. 1.0
11/2010

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Contents

About This Book

The PxP driver provides access to an array of useful hardware-accelerated image processing transformations, including color space conversion, resizing, rotation, and combining of multiple images. In order to provide user-space access to the PxP driver, the PxP user-space library was created. This library provides a clean, simple interface to the PxP driver. This document explains how to build and use this library.

Audience

This document is intended for software, hardware, and system engineers who are planning to use the ePxP user library and for anyone who wants to understand more about the ePxP.

References

1. ltib_build_host_setup.pdf

Chapter 1
Contents

1.1 PxP User-space Library

The PxP user library resides in `imx-lib` package in LTIB release package. The following steps should be taken to build the PxP user library:

1. Run LTIB to build all of the libraries and place them in your root filesystem

(for PxP, namely libpxp.so and libpxp.a under /usr/lib):

```
./ltib -p imx-lib -f
```

1.2 PxP Test Application

A PxP test application is provided to demonstrate use of the PxP user-space library to perform PxP operations. The application is provided as a patch to be applied to the Linux BSP unit tests package. The application currently supports two means for handling the processed PxP output data: writing it to an output file, and copying it to the EPDC framebuffer (Y8 format) and subsequently updating the E-Ink panel display content through the EPDC framebuffer driver APIs.

The PxP test application resides in `imx-test` package in LTIB release package. The following steps should be taken to build and run the PxP test application:

1. Run LTIB to build all of the unit tests and place the executables in your root filesystem:

   ```
   ./ltib -p imx-test -f
   ```

2. After booting to the Linux prompt and connecting to the root filesystem, run the PxP test:

   ```
   /unit_tests/pxp_test.out
   ```

Notes

1. The compilation of PxP Test Application depends on PxP User-space Library. To compile PxP Test Application, compile PxP User-space Library first.

Chapter 2
Key Data Structs

The following key data structs are shared between the PxP driver and PxP user library.

2.1 struct pxp_config_data

```
struct pxp_config_data {
        struct pxp_layer_param s0_param;
        struct pxp_layer_param ol_param[8];
        struct pxp_layer_param out_param;
        struct pxp_proc_data proc_data;

        /* Users don't touch */
        int chan_id;
};
```

This structure contains the key configuration parameters for a PxP processing operation. The s0 layer, ol layers, and out layers define, respectively, the input, overlays, and output image planes. See below for detailed info on struct pxp_layer_param.

2.2 struct pxp_layer_param

```
struct pxp_layer_param {
        unsigned short width;
        unsigned short height;
        unsigned int pixel_fmt;

        // layers combining parameters
        // (these are ignored for S0 and output
        // layers, and only apply for OL layer)
```

```
    bool combine_enable;
    u32 color_key_enable;
    u32 color_key;
    bool global_alpha_enable;
    u8 global_alpha;
    bool local_alpha_enable;

    dma_addr_t paddr;
};
```

This is the configuration data for each layer (S0, Output, Overlay). The variables *width* and *height* provide the dimensions of the buffer. *pixel_fmt* is pixel format, which could be one of the FOURCC values that the PxP supports. *paddr* is the physical address of the image buffer.

2.3 struct pxp_proc_data

```
struct pxp_proc_data {
    /* S0 Transformation Info */
    int scaling;
    int hflip;
    int vflip;
    int rotate;

    // Source rectangle (srect) defines the sub-rectangle
    // within S0 to undergo processing.
    struct rect srect;
    // Dest rect (drect) defines how to position the processed
    // source rectangle (after resizing) within the output frame,
    struct rect drect;

    /* Current S0 configuration */
    u32 bgcolor;
```

```
        /* Output overlay support */

        int overlay_state;


        /* LUT transformation on Y data */

        int lut_transform;

};
```

This structure provides non-channel-specific processing parameters for PxP operations. *scaling* is only for YUV format. *hflip* and *vflip* control horizontal and vertical flipping of the output buffer. *rotate* defines the rotation that should be applied to the output image. Valid values are 0, 90, 180, and 270. *srect and drect* can be used conjointly to configure cropping and masking. *lut_transform* is for LUT transformation on Y data, the value of which could be PXP_LUT_NONE (no transformation) or PXP_LUT_INVERT (inverts the Y pixel data).

2.4 struct pxp_mem_desc

```
struct pxp_mem_desc {

        u32 size;

        dma_addr_t phys_addr;

        u32 cpu_addr;          /* cpu address to free the dma mem */

        u32 virt_uaddr;        /* virtual user space address */

};
```

This structure is used to facilitate the allocation and freeing of memory blocks for use with the PxP. Users of the PxP library need only be concerned about *size*, *phys_addr,* and *virt_uaddr*. The user should call API *pxp_get_mem()* to acquire a physically contiguous memory region of size *size*. If successful, this call should provide valid values for *virt_uaddr* and *phys_addr*. When the buffers are no longer needed, the *pxp_put_mem()* should be called to free the memory.

Chapter 3
APIs User Guide

The APIs are listed as follows:

int pxp_init();

void pxp_uninit();

*int pxp_request_channel(pxp_chan_handle_t *pxp_chan);*

void pxp_release_channel(pxp_chan_handle_t pxp_chan);

*int pxp_config_channel(pxp_chan_handle_t pxp_chan, struct pxp_config_data *pxp_conf);*

int pxp_start_channel(pxp_chan_handle_t pxp_chan);

*int pxp_wait_for_completion(pxp_chan_handle_t *pxp_chan, int times);*

*int pxp_get_mem(struct pxp_mem_desc *mem);*

*int pxp_put_mem(struct pxp_mem_desc *mem);*

The steps to use APIs to perform PxP operations are as follows.

1. For each process using the PxP library, *pxp_init()* should be called once before any other PxP APIs. If, for example, there are two processes, each of which performs one kind of PxP operation separately, then each process must call *pxp_init()* once separately. *pxp_uninit()* should be called when use of the PxP has ended.
2. Call *pxp_request_channel()* to acquire a free channel. Detailed elements of channel management are handled by the driver and are transparent to users.
3. Call *pxp_config_channel()* to configure the channel. This step provides everything that the PxP needs to execute a processing task.
4. Call *pxp_start_channel()* to trigger the start of PxP operations.
5. Call *pxp_wait_for_completion()* to wait for the completion of PxP operations. Once this function has completed, the user may access the output data of PxP, as well as the histogram status.
6. Call *pxp_release_channel()* to release the channel after all PxP tasks are done.
7. Repeat the steps from 2 to 6 (or from 3 to 5).
8. Call *pxp_uninit()* when PxP is no longer being used.

For *pxp_get|put_mem()* APIs, please refer to the description of **struct pxp_mem_desc**.