Crystal Image through
Imaging Innovation

PIXELPLUS

SURROUND VIEW MONITORING SYSTEM

PI5008K SDK USER GUIDE

Rev 0.7

Last Update: 2018.12.06

6th Floor, 105, Gwanggyo-ro, Yeongtong-gu,
Suwong-si, Gyeonggi-do, 16229, Korea
Tel : +82-31-888-5300, FAX : +82-31-888-5399

Contents

Figure

Tables

1. Introduction

1.1. PI5008K

PI5008K is SVM SOC based on dual core 32 bit RISC processor and has various hardware accelerators. It provides essential functions for SVM system such as high quality 4 channel image stitching, fish eye correction and 3D seamless merging by using dedicated hardware accelerators.

PI5008K also provides various useful functions such as ISP for 4 camera inputs, Quad output for recording etc.

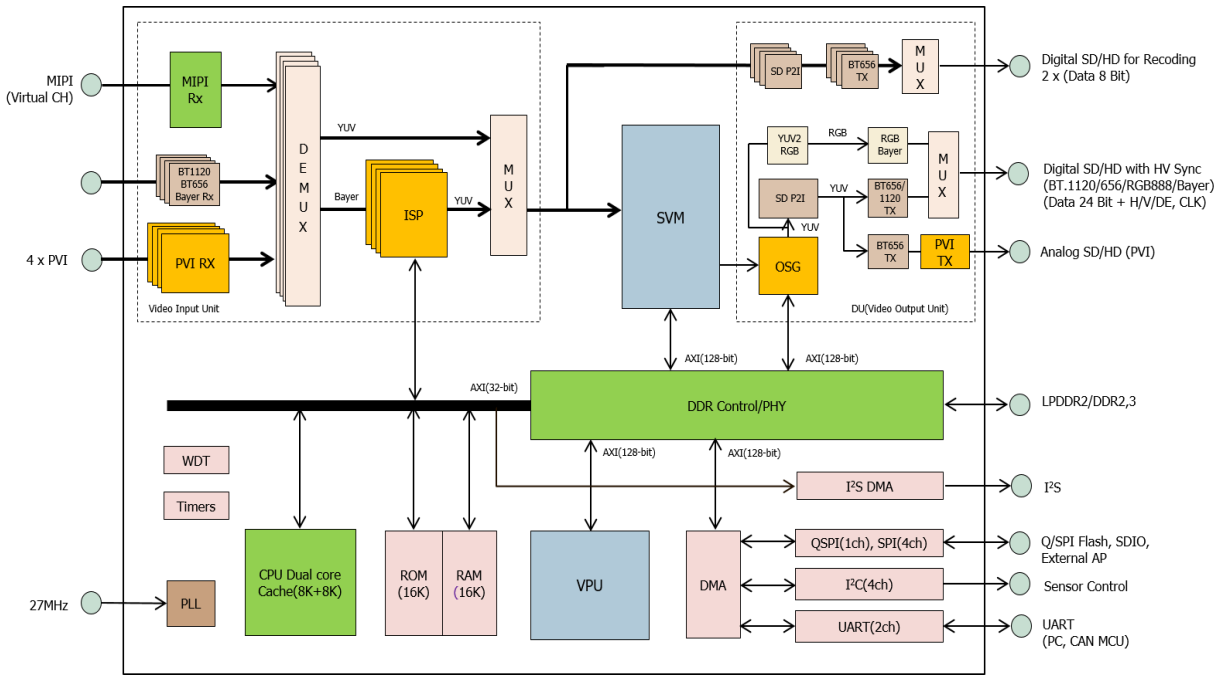


Figure 1 PI5008K Block Diagram

1.2. SVM SDK

PI5008X SVM SDK is software development platform for SVM solution. It is based on RTOS. With PC Tools, SVM SDK makes it possible to implement SVM solution and integrate it into PI5008K.

1.3. System features

PI5008K SVM SDK supports various features.

There are many options and definitions in board_config.h to configure system features. System features have to be set before building the binary image.

Option shows how many choices are available for a feature.

Each feature can be defined by combining options.

1) Options for each feature

A. Video input type

: Camera input type

Video input type	Description
VIDEO_IN_TYPE_MIPI_BAYER	Video input type is MIPI Bayer
VIDEO_IN_TYPE_MIPI_YUV	Video input type is MIPI YUV
VIDEO_IN_TYPE_PVI	Video input is PVI(analog)

Table 1 Video input type

B.MIPI input format

: MIPI Bayer/YUV raw bit format

MIPI input format	Description
VIDEO_IN_FMT_RAW6	Bayer raw 6bit
VIDEO_IN_FMT_RAW7	Bayer raw 7bit
VIDEO_IN_FMT_RAW8	Bayer raw 8bit
VIDEO_IN_FMT_RAW10	Bayer raw 10bit
VIDEO_IN_FMT_RAW12	Bayer raw 12bit
VIDEO_IN_FMT_RAW14	Bayer raw 14bit
VIDEO_IN_FMT_YUV8_2XRAW8	YUV 8bit or Bayer raw 16bit
VIDEO_IN_FMT_2XRAW10	Bayer raw 20bit

Table 2 MIPI input format

C. Video signal format

: Video signal format: YUV 8/16, embedded/external sync, RGB, Bayer 8/10

Video signal format	Description
VID_TYPE_YC8_EMB	YUV 8bit embedded sync
VID_TYPE_YC8_EXT	YUV 8bit external sync

VID_TYPE_YC16_EMB	YUV 16bit embedded sync
VID_TYPE_YC16_EXT	YUV 16bit external sync
VID_TYPE_RGB24	RGB 24bit
VID_TYPE_BAYER_8BIT	Bayer 8bit
VID_TYPE_BAYER_10BIT	Bayer 10bit

Table 3 Video signal format

D. Video frame

: Video frame 25/30/50/60

Video frame	Description
VID_FRAME_NTSC_30	30frame(NTSC)
VID_FRAME_PAL_25	25frame(PAL)
VID_FRAME_NTSC_60	60frame(NTSC)
VID_FRAME_PAL_50	50frame(PAL)

Table 4 Video frame

E. Video resolution

: Video resolution

Video resolution	Description
VID_RESOL_SD720H	CVBS 720x480i, 720x576i
VID_RESOL_SD960H	CVBS 960x480i, 960x576i
VID_RESOL_SDH720	Reserved. 720x480p, 720x576p
VID_RESOL_SDH960	960x480p, 960x576p
VID_RESOL_HD720P	1280x720p
VID_RESOL_HD960P	1280x960p
VID_RESOL_HD1080P	1920x1080p
VID_RESOL_HD800_480P	800x480p
VID_RESOL_HD1024_600P	1024x600p

Table 5 Video resolution

F. Analog Video Standard

: Analog video standard

Analog video standard	Description
VID_STANDARD_CVBS	CVBS (720x480i, 720x576i, 960x480i, 960x576i)

VID_STANDARD_SDH	Reserved (720x480p, 720x576p)
VID_STANDARD_PVI	Analog HD PVI(pixelplus) standard.
VID_STANDARD_CVI	Analog HD CVI(Dahwa) standard.
VID_STANDARD_HDT	Analog HD TVI(HikVision) standard.
VID_STANDARD_HDA	Analog HD AHD(Nextchip) standard.

Table 6 Analog video standard

G. Parallel Video Type

: Parallel digital video type.

PARALLEL video type	Description
VID_PARALLEL_TYPE_VIN_BAYER	Bayer video type
VID_PARALLEL_TYPE_VIN_YUV	YUV video type

Table 7 Parallel video type

H. Analog Tx source

: Analog tx (PVITX) source type

Analog Tx source	Description
PVITX_SRC_NONE	Don't use Analog TX(PVITX)
PVITX_SRC_DU	Analog TX(PVITX) source is DU.
PVITX_SRC_QUAD	Analog TX(PVITX) source is QUAD.

Table 8 Analog tx source

I. Flash type

: System flash type

Flash type	Description
FLASH_TYPE_NOR	Flash type is NOR
FLASH_TYPE_NAND	Flash type is NAND

Table 9 Flash type

J. Upgrade option

: Upgrade method

Upgrade method	Description
UPGRADE_METHOD_SDCARD	Upgrade by SD card
UPGRADE_METHOD_SPI	Upgrade by external SPI host

Table 10 Upgrade option

2) User defined features

The feature is defined by bitwise OR operation of several options.

Description	Name of feature	option 1	option 2	option 3	option 4
Video In/Out Feature setting					
Video Input Type	VIDEO_IN_TYPE	A			
Video Input Format	BD_VIN_FMT	D	E		
Analog Video Input format	BD_CAMERA_IN_FMT (VIDEO_IN_TYPE=MIPI)	BD_VIN_FMT		B	
	BD_CAMERA_IN_FMT (VIDEO_IN_TYPE=PVI)	BD_VIN_FMT		F	
SVM VIDEO In/Out	BD_SVM_IN_FMT	BD_VIN_FMT			
	BD_SVM_OUT_FMT	D	E		
DU VIDEO In/Out	BD_DU_IN_FMT	BD_SVM_OUT_FMT			
	BD_DU_OUT_FMT	BD_DU_IN_FMT		C	
QUAD VIDEO OUT	BD_QUAD_OUT_FMT	BD_VIN_FMT			
Record Out(RO) VIDEO OUT	BD_RO_OUT_FMT	BD_VIN_FMT		C	
VPU VIDEO IN	BD_VPU_IN_FMT	BD_QUAD_OUT_FMT			
Analog Video Output	BD_PVITX_OUT_FMT (SOURCE=DU)	BD_DU_IN_FMT		F	H (SRC_DU)
	BD_PVITX_OUT_FMT (SOURCE=QUAD)	BD_QUAD_OUT_FMT		F	H (SRC_QUAD)
FLASH Feature setting					
FLASH Type	BD_FLASH_TYPE	I			
SPI NAND Flash feature	SPI_NAND_NON_FTL_BLOCKS	The number of blocks for NAND flash non-FTL region.			
Flash controller feature	FLASH_CTRL_DIV*	0 ~ 15			
	FLASH_CTRL_DELAY**	0 ~15			
FLASH SPI Controller	SPI_CTRL_DIV	0 ~ 15			

Feature					
Upgrade Feature setting					
Upgrade option	UPGRADE_METHOD	J			
UI & Display Feature setting					
USE_PP_GUI	Decide whether to use Pixel Plus GUI				
USE_BOOTING_IMG	Decide whether to use Pixel Plus Booting CI				
USE_GUI_MENU	Decide whether to use Pixel Plus GUI Menu				
USE_CAR_WHEEL	Decide whether to use Car wheel animation				
USE_CAR_DOOR	Decide whether to use Car door display				
USE_SEPERATE_SHADOW	Decide whether to use shadow and car separate Images				
USE_SEPERATE_PGL	Decide whether to use static pgl and dynamic pgl images				

Table 11 Features

Note1> This feature is used to make flash memory access speed slow if it is too fast.
0 is default value. As the value is increased, the speed becomes slow.

Note2> This feature is used to adjust the read timing after each SPI clock. Delay will be set in the unit of SPI core clock.

Example Video In/Out Feature setting

1) Input: MIPI 720p25 bayer raw12bit camera, Output: 720p25 YUV8bit Embedded Sync

```
#define VIDEO_IN_TYPE          (VIDEO_IN_TYPE_MIPI_BAYER)
#define BD_VIN_FMT             (VID_FRAME_PAL_25 | VID_RESOL_HD720P)
#define BD_CAMERA_IN_FMT       (BD_VIN_FMT | VIDEO_IN_FMT_RAW12)
#define BD_SVM_IN_FMT          (BD_VIN_FMT)
#define BD_SVM_OUT_FMT         (VID_FRAME_PAL_25 | VID_RESOL_HD720P)
#define BD_DU_IN_FMT           (BD_SVM_OUT_FMT)
#define BD_DU_OUT_FMT          (BD_DU_IN_FMT | VID_TYPE_YC8_EMB)
#define BD_QUAD_OUT_FMT        (BD_VIN_FMT)
#define BD_RO_OUT_FMT          (BD_VIN_FMT | VID_TYPE_YC8_EMB)
#define BD_VPU_IN_FMT          (BD_QUAD_OUT_FMT)
#define BD_PVITX_OUT_FMT       (BD_DU_IN_FMT | PVITX_SRC_NONE)
```

2) Input: MIPI 960p25 bayer raw12bit camera, Output: 720p25 YUV16bit External Sync

```
#define VIDEO_IN_TYPE          (VIDEO_IN_TYPE_MIPI_BAYER)
#define BD_VIN_FMT             (VID_FRAME_PAL_25 | VID_RESOL_HD960P)
#define BD_CAMERA_IN_FMT       (BD_VIN_FMT | VIDEO_IN_FMT_RAW12)
#define BD_SVM_IN_FMT          (BD_VIN_FMT)
#define BD_SVM_OUT_FMT         (VID_FRAME_PAL_25 | VID_RESOL_HD720P)
#define BD_DU_IN_FMT           (BD_SVM_OUT_FMT)
#define BD_DU_OUT_FMT          (BD_DU_IN_FMT | VID_TYPE_YC16_EXT)
#define BD_QUAD_OUT_FMT        (BD_VIN_FMT)
#define BD_RO_OUT_FMT          (BD_VIN_FMT | VID_TYPE_YC8_EMB)
#define BD_VPU_IN_FMT          (BD_QUAD_OUT_FMT)
#define BD_PVITX_OUT_FMT       (BD_DU_IN_FMT | PVITX_SRC_NONE)
```

3) Input: HDA 720p25 camera, Output: Digital [720p25 YUV8bit External Sync], Analog HD[HDA 720p25 Du]

```
#define VIDEO_IN_TYPE          (VIDEO_IN_TYPE_PVI)
#define BD_VIN_FMT             (VID_FRAME_PAL_25 | VID_RESOL_HD720P)
#define BD_CAMERA_IN_FMT       (BD_VIN_FMT | VID_STANDARD_HDA)
#define BD_SVM_IN_FMT          (BD_VIN_FMT)
#define BD_SVM_OUT_FMT         (VID_FRAME_PAL_25 | VID_RESOL_HD720P)
#define BD_DU_IN_FMT           (BD_SVM_OUT_FMT)
#define BD_DU_OUT_FMT          (BD_DU_IN_FMT | VID_TYPE_YC16_EXT)
#define BD_QUAD_OUT_FMT        (BD_VIN_FMT)
#define BD_RO_OUT_FMT          (BD_VIN_FMT | VID_TYPE_YC8_EMB)
#define BD_VPU_IN_FMT          (BD_QUAD_OUT_FMT)
#define BD_PVITX_OUT_FMT       (BD_DU_IN_FMT | PVITX_SRC_DU |
 VID_STANDARD_HDA)
```

1.4. S/W Layers

<table>
<tr><td rowspan="4">OS
KERNEL</td><td colspan="7">Tasks/APP</td></tr>
<tr><td>File system
API</td><td colspan="3">API(Application Programming Interface)s</td><td colspan="3">Algorithms</td></tr>
<tr><td>File system
library</td><td>FTL</td><td>Display
Driver</td><td>SVM
Driver</td><td>Video I/O
Driver</td><td>ISP
Driver</td><td>Quad/Mux
Driver</td></tr>
<tr><td>SD/MMC
Controller</td><td>Flash
Controller</td><td>Display
Unit</td><td>SVM
Block</td><td>Video I/O
Block</td><td>ISP</td><td>Quad/Mux
Block</td></tr>
<tr><td>External
H/W</td><td>SD/MMC
Card</td><td>NAND/N
OR Flash</td><td colspan="2">Display Device</td><td colspan="2">Camera Module</td><td>Record
Out</td></tr>
</table>

Table 12 S/W Layers

APP is the application layer which implements user-level functions of PI5008K SDK such as selecting view mode, menu operation, calibration, etc. This layer uses APIs to perform these functions.

How to handle command from ECU

Command from ECU will be handled by UART interrupt handler. This handler sends an event to UITask. After receiving the event, UITask parses command and calls a function in APP layer. This function executes the command by calling APIs directly or sends the event to other task. For example, The event will be sent to DisplayTask If the command needs to update display

1.5. Task Architecture

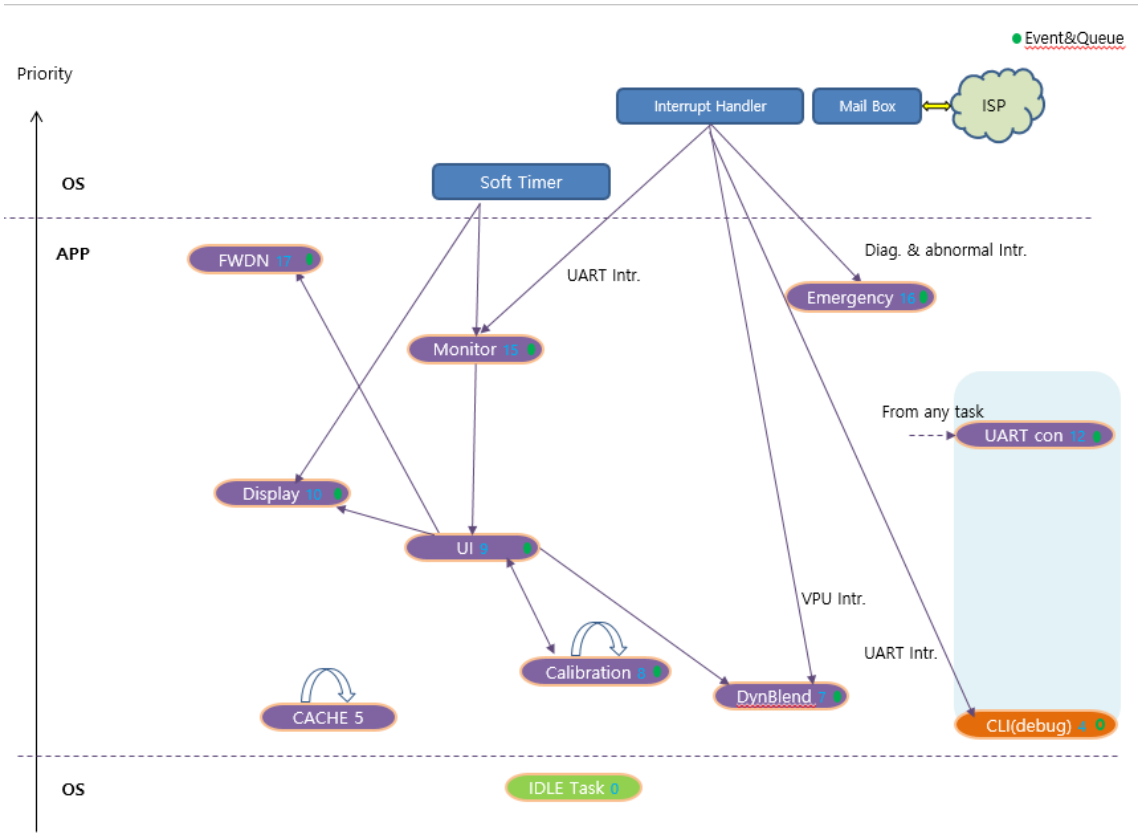


Figure 2 Task Architecture

PI5008K SDK has several tasks for view generation, on-board calibration, system monitoring etc. Each task is scheduled by OS kernel and communicates with each other using message queue, Semaphore, and events.

Task Name	Priority	Description
FWDNTask	17	Upgrade binary image
EmergencyTask	16	Handles system error
MonitorTask	15	Runs at every 20 msec Check User Input, SD Card Execute periodic functions
UARTConTask	12	UART Console Task for debugging
DisplayTask	10	Display Update
UITask	9	User Interface
CalibrationTask	8	Calibration and Video Input Image Save
DynBlendTask	7	Executes Dynamic Blending function
CacheTask	5	Load LUT & Car image during 360 degree swing

Table 13 Task description

Notes> Higher number means higher priority.

1.6. Source Tree

1.6.1. SDK folder structure

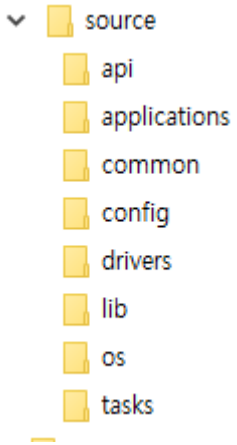


Figure 3 SDK folder Structure

1.6.2. Detailed Structure of SDK folder

depth 0	depth 1	depth2	depth3	description
source	api			Application Interfaces
	applications	Application		User Application
		Calibration		Sample Program of Offline Calibration & View Generation.
		Display		
	common			
	config			
	drivers	adc		
		cannyedge		
		diag		
		dma		
		du		
		gpio		
		i2c		
		include		

		interrupt		
		ipc		
		pvi	tx	
			rx	
		sflash		
		spi		
		svm		
		system		
		timer		
		uart		
		vin		
		vpu		
		wdt		
	lib	Calibration	Doc	calibration library documents
			libOfflineCalib	offline calibration library object files
			libSvmView	view generation library object files
		ObjectDetect	Doc	
			libObjectDetect	Dynamic blending library object files
	os	debug		
		kenel	license	
			source	
			nds32	
			osal	OS Adaptation layer
	task	Cache		Cache task
		Calibration		Calibration task
		CLI		CLI task
		Display		Display task
		DynBlend		Dynamic Blending task
		Emergency		Emergency task
		Fs		File System task
		FWDN		Firmware download task
		Monitor		Monitor task

		UARTCon		UART Control task
		UI		UI Task

Table 14 Detailed Structure of SDK Folder

2. SDK Overview

2.1. Basic Operation

2.1.1. Video Input

1) Video Input block diagram

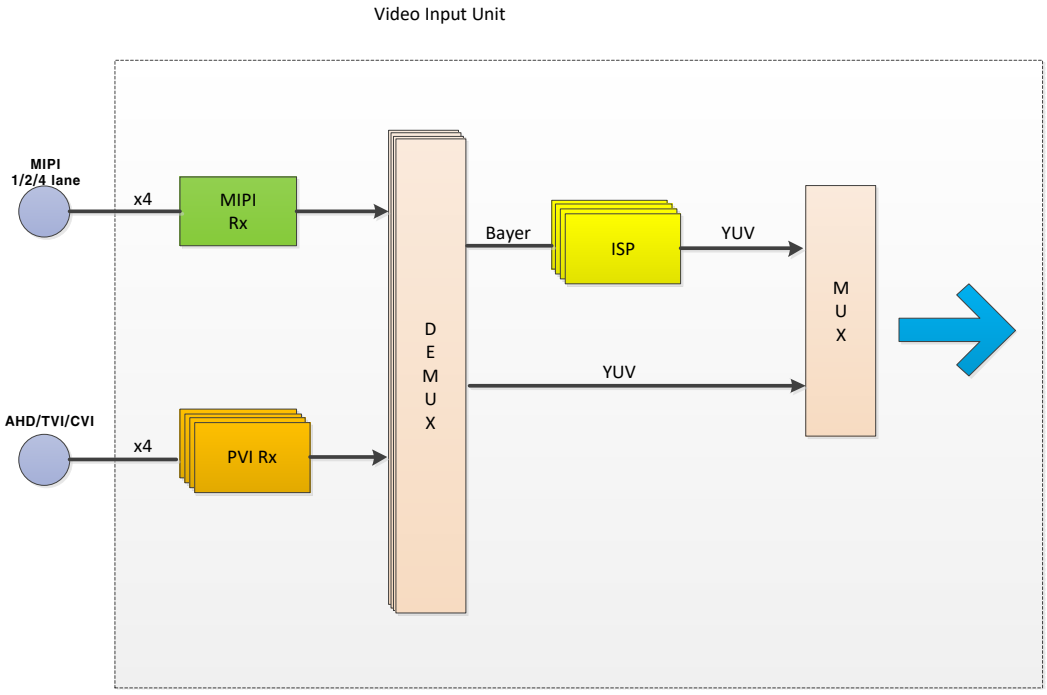


Figure 4 Video Input Unit

2) Video Input Types

A. MIPI

i. 1/2/4 lane

B. Analog SD/HD(AHD/TVI/CVI)

i. NTSC/PAL

ii. Analog HD(AHD/TVI/CVI) 720p/1080p

C. Parallel digital

i. BT1120/BT656/Bayer

3) How to select video Input

Please refer to PI5008K Video IO User Guide

2.1.2. Surround View Generation

Surround view generation consists of 3 basic operations: correcting the fish-eye distortion for input video frames and converting them to a common birds-eye perspective. Generating surround view after correction and correcting the brightness and color mismatch between adjacent views to seamless stitching. PI5008K performs these operation using 3 kinds of Look-Up Tables (LUT): Front+Rear, Left+Right and BC(Brightness Control).

1) SVM Block Diagram

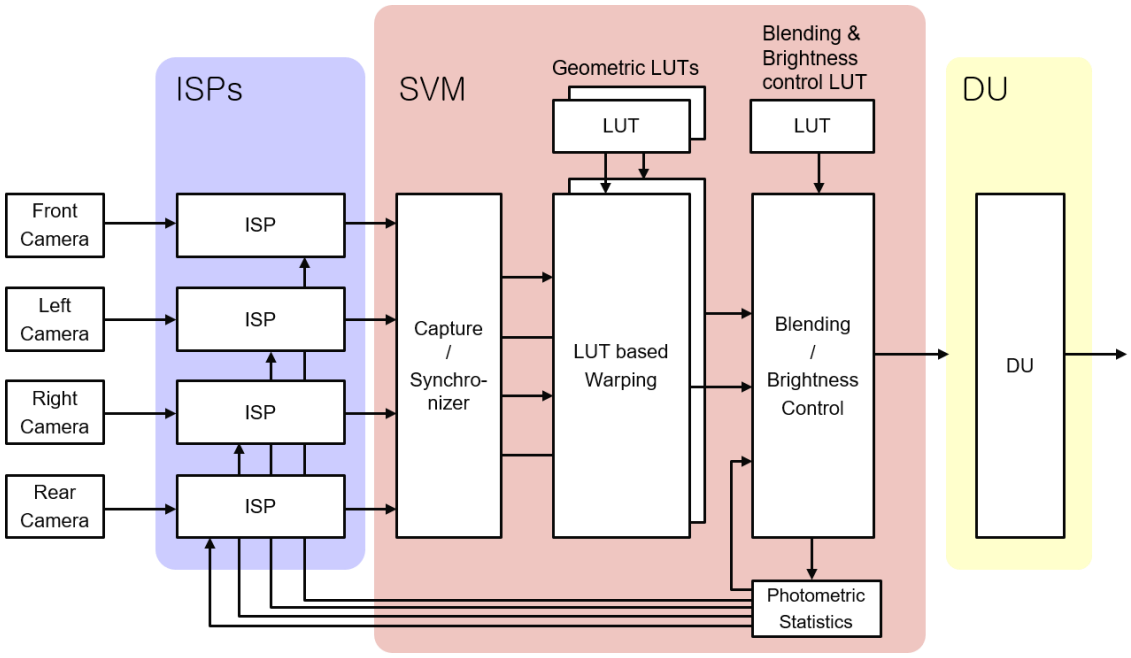


Figure 5 SVM Block Diagram

2) LUTs

PI5008 SVM block uses 3 Look-up tables to generate a surround view. Two LUTs

(Front/Back, Left/Right) holds the information how each pixel of 4 cameras are mapped into output surface. These LUTs are used to warp 4 camera inputs.
BC LUT is used for blending and brightness control. These LUTs are generated during view generation and stored at the flash memory. When a user selects a view mode, LUT for the selected view is loaded from flash memory and used to generate surround view by SVM block.

3) View generation process

The procedure to make surround view using LUTs is shown in below figure.

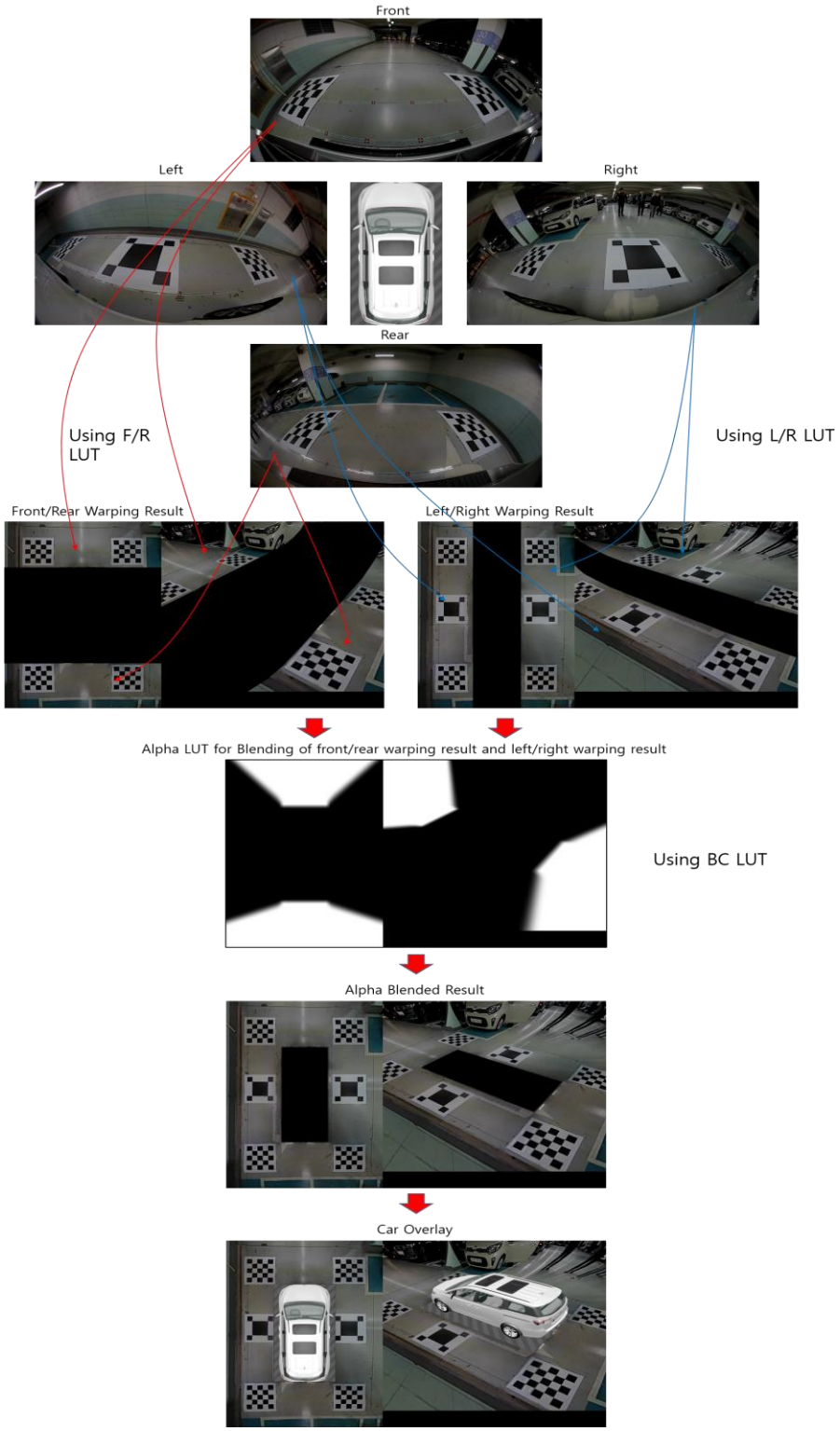


Figure 6 How to use LUT to make surround view

4) View mode change

To change view mode, registers of SVM block should be set to access new LUTs.

There is also a timing requirement to change SVM registers.

A. Procedure to change view mode

1. Loading LUT & Car Image for a new view

LUT and car image have to be loaded into DDR before changing the view.

2. Setting SVM and DU block

Setting for new view mode will be done in Vsync interrupt handler as follows.

a) Setting SVM block with LUT address for new view

b) Setting DU block after one frame

3. Latency

SVM output is delayed by 2 frames after setting.

On the other hand, DU output is delayed one frame after setting.

B. Timing for view mode change

Below timing chart shows continuous view mode change at every frame.

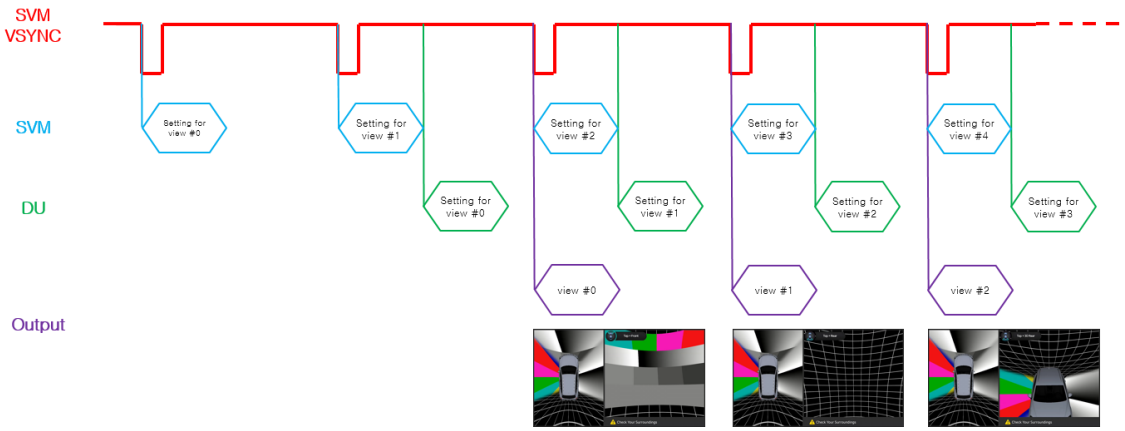


Figure 7 Timing chart for view mode change at every frame (1/2)

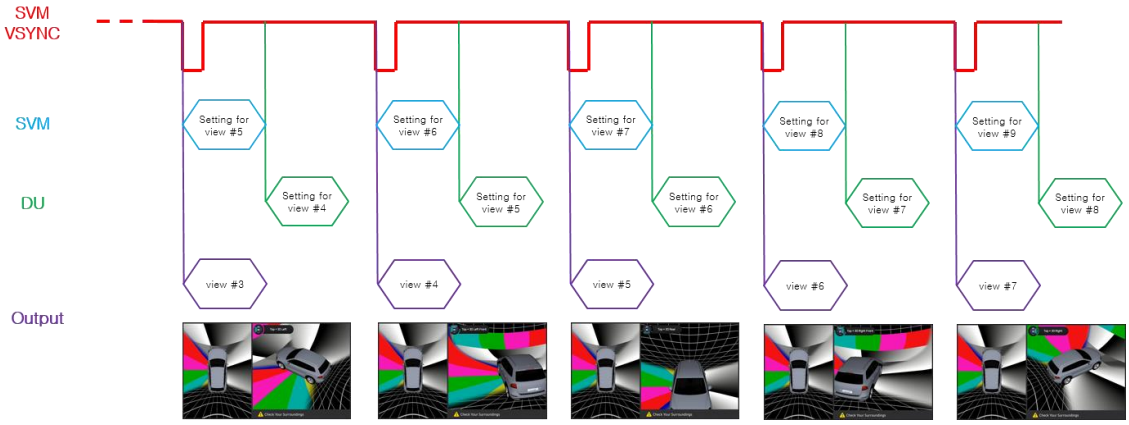


Figure 8 Timing chart for view mode change at every frame (2/2)

5) View modes

PI5008K supports 2 view modes: 3D and 2D modes.

5.1) 2D view modes

Top 2D view will be displayed on the left side of the screen.

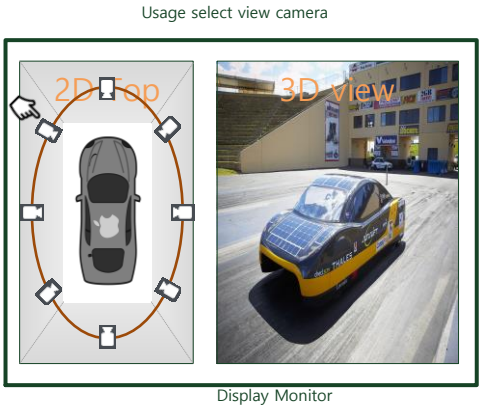
2D view will be displayed on the right side.

There are 6 views : Front/Left/Right/Rear/Wide Front*/Wide Rear*

Notes> 2D view is not displayed.

5.2) 3D View modes

Top 2D view will be displayed on the left side of the screen same as 2D mode. But there is an oval around the car and 8 points which represent viewing points. By choosing a point, user can see 3D view of the point on the right side of the screen.



There are two ways to move between 2 points.

a) Jump to new view point

b) Smooth movement to new point

To support smooth movement, large amount of memory (Flash and DDR) is required. And on-board calibration time will be also increased very much.

6) Car image overlay of 3D mode

In 3D view, car image will be displayed by DU block according to the selected view mode. Car image will be displayed on top of shadow image*.

Notes> Shadow area and image

There is an area where no camera image exists because it is below the car. This area needs to be covered by an image. This image is called shadow image.

2.1.3. On-Board Calibration

PC tools will extract extrinsic camera data and LUT data for various view mode using images from SVM device installed in the reference vehicle and this data can be used for other vehicles of the same model. But the camera position and angle of each car can be different with the reference vehicle. On-board calibration can be used to get the optimized LUTs for the car.

On-board calibration includes two processes – camera calibration and view generation. Camera calibration is used to get extrinsic camera data – camera position and angle. View generation makes LUTs for each view mode using extrinsic data and view configuration which made by PC Tool.

1) Camera calibration Process

Input: Camera intrinsic data, Captured image of each camera.

Output: Camera extrinsic data

Image Capture -> Extract extrinsic data for 4 cameras using intrinsic data and captured image -> Show results -> decide OK or NG.

If the result is OK, view generation will be started automatically.

2) View generation process

Input: Captured Image, camera intrinsic/extrinsic data, configuration for views

Output: L/R, F/B, BC LUT

After generating LUT for 2D Top view and 2D Front view, it will be displayed on the screen to decide whether the result is correct and view generation will be continued until LUT for all view is generated.

3) Saving result

After making LUTs for all view modes, LUTs are stored at flash memory.

And calibration menu will be displayed again.

2.1.4. Analog Video Output

1) Analog video output block diagram

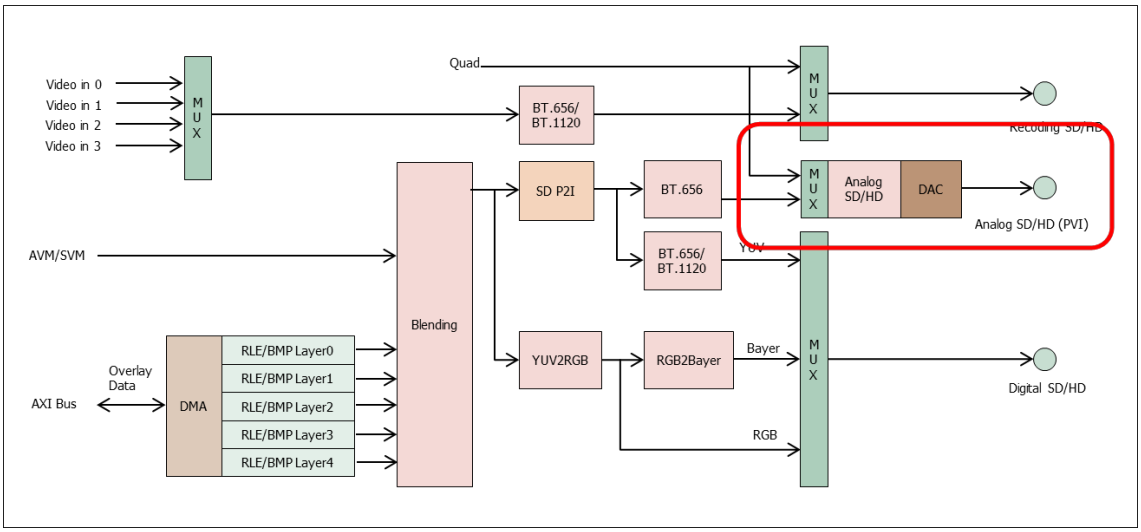


Figure 9 Analog Video Output Block Diagram

2) Analog video output type

A. Select source

i. Quad output

ii. DU(Display Unit) output

B. Analog SD/HD(AHD/TVI/CVI)

i. NTSC/PAL

ii. Analog HD(AHD/PVI/CVI) 720p/1080p

3) How to set analog video output

Please refer to PI5008K Video IO User Guide

2.1.5. Digital Record Output

1) Digital record output block diagram

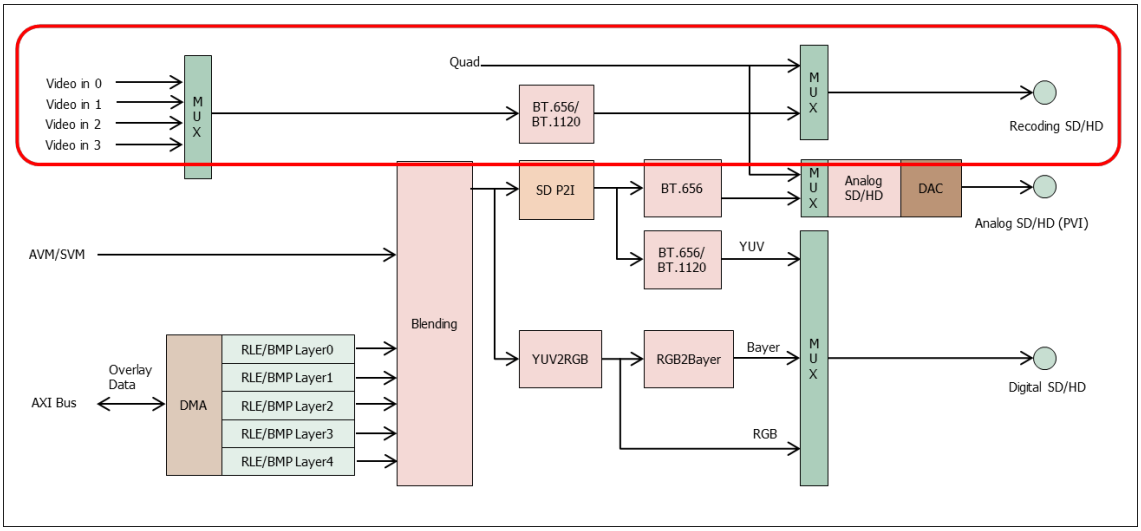


Figure 10 Digital Record Output Block Diagram

2) Digital record output type

A. Select source

i. Quad output

ii. Video Input

B. Parallel digital

i. BT1120/BT656

ii. 8/16bit

iii. Multiplexing channel(1/2/4)

3) How to set digital record output

Please refer to PI5008K Video IO User Guide

2.1.6. Brightness Control

SVM block needs F/B LUT and L/R LUT to generate F/B image and L/R image respectively for 4 camera inputs. If there is a big difference of brightness level in overlapped regions (Front + L/R, Back + L/R), the merged image will not seem to be natural. To decrease the difference of brightness level between overlapped regions, PI5008 SDK provides brightness control function.*

1) Block diagram

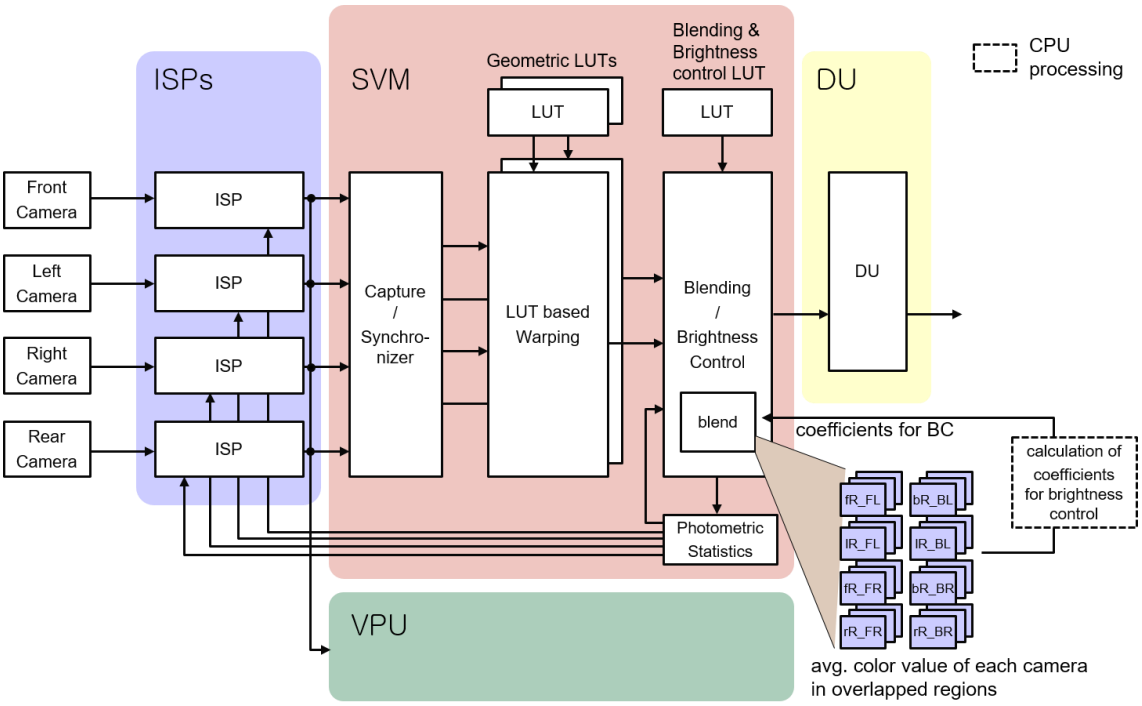


Figure 11 Brightness Control Block Diagram

2) How to control brightness

It is possible to detect the brightness level of each camera by reading SVM registers. After detecting the level, PI5008 extract the brightness ratio between two adjacent cameras and control the brightness of output image of SVM block to decrease the difference. Brightness control will be done at every V sync.

Notes >Brightness control will be done by sub CPU.

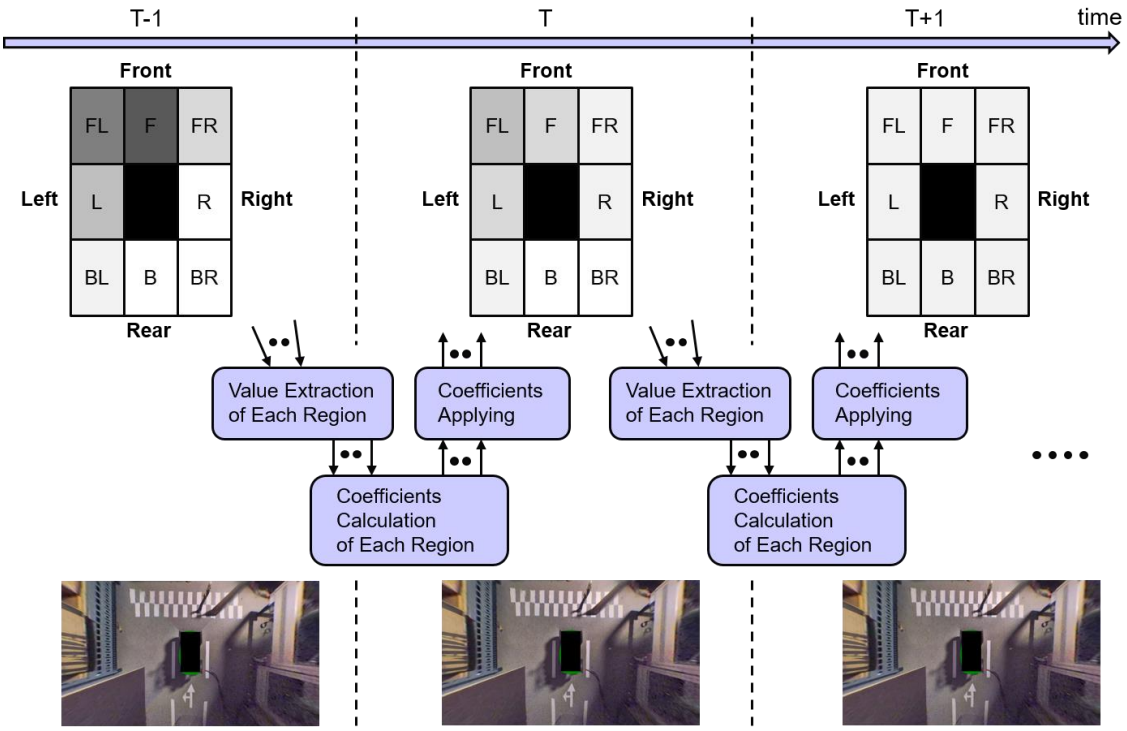


Figure 12 How to Control Brightness

2.1.7. Dynamic blending

If there is an object above a certain height, the upper part of it can be disappeared in the overlapped area. PI5008 SDK provides dynamic blending function to fix this problem by changing blending area dynamically according to the movement of the object.

1) Block diagram

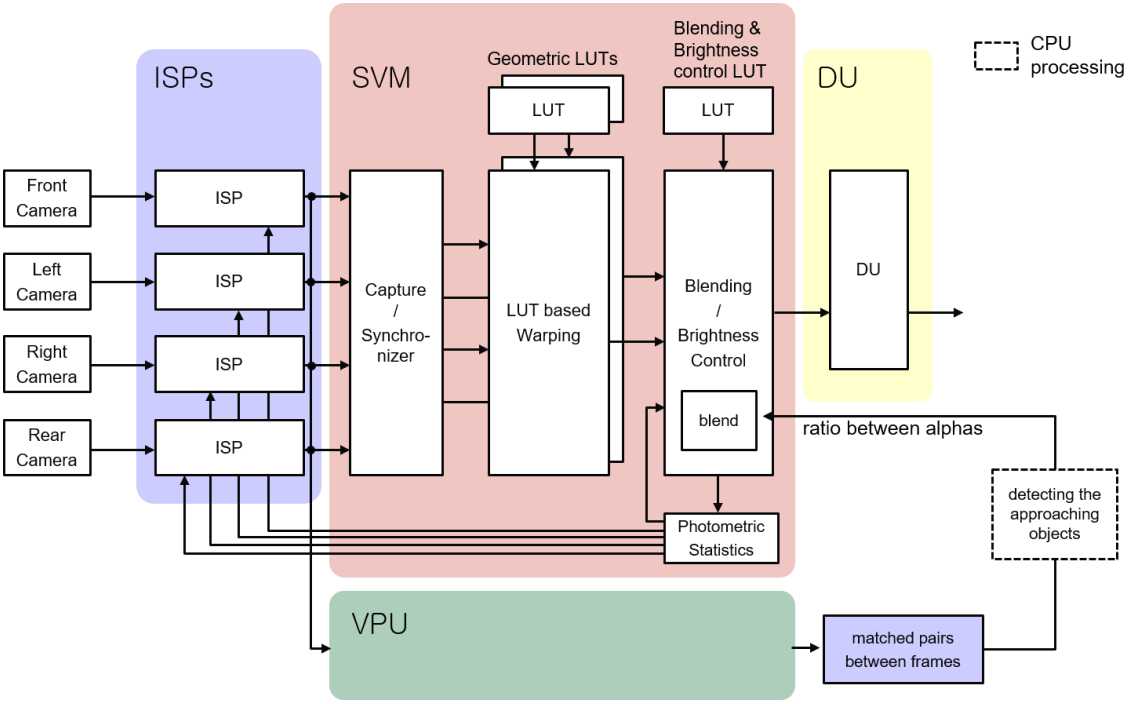


Figure 13 Dynamic Blending Block Diagram

2) Dynamic Blending Process

The core function of the dynamic blending process is object detection which is done by VPU block. Object detection will be done for an object not only approaching to vehicle but also moving away from the vehicle in the opposite direction. Proper blend map is chosen based on the result of object detection. Dynamic Blending can be disabled by the menu.

A. Procedure to dynamic blending

1. Setting VPU block.

   Select the object detection region of the camera image.

   Set proper VPU parameters. (Fast Brief, Hamming Distance, etc.)

2. Initialize object detection library.

3. Running object detection library using VPU block.

   Run VPU block. VPU block return Fast Brief and Hamming Distance result.

   Object detection software detect moving vehicle direction and apply proper blend map.

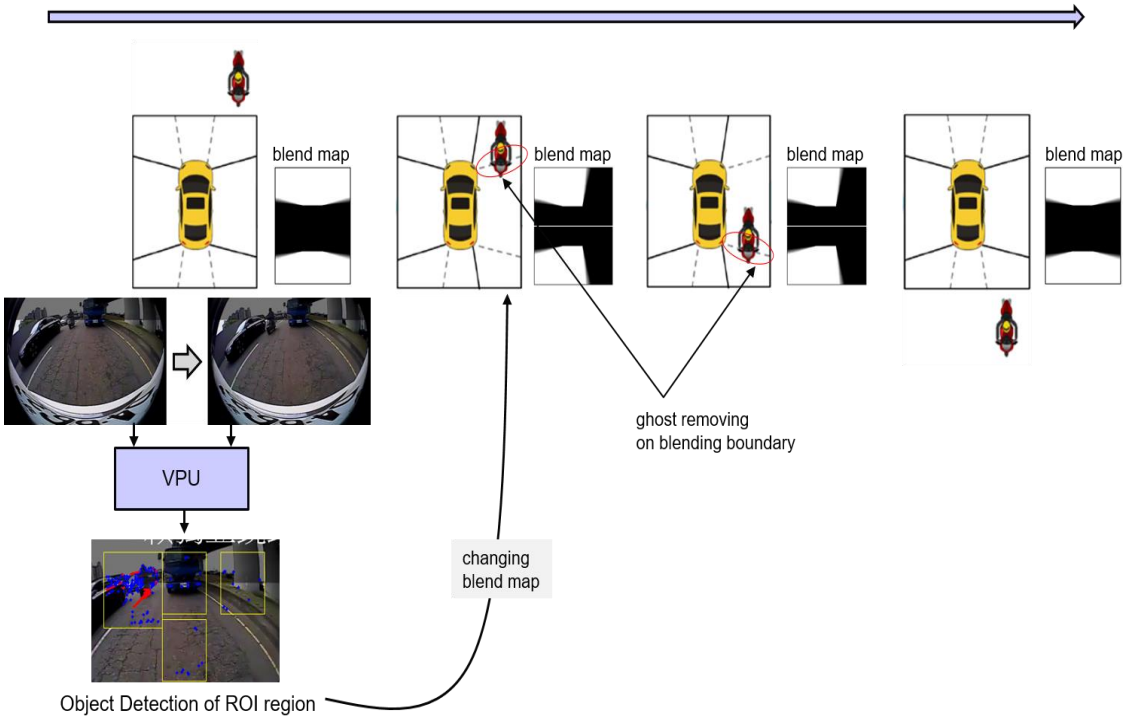


Figure 14 Dynamic Blending Process

3) Limitations

Dynamic blending can be used only while the vehicle is stopped. (Gear position: N or P)

The object detection of PI5008 of SDK will be done PI5008 hardware and s/w algorithm. The algorithm of SDK is made only for test purpose and the accuracy is not so high. For real SVM device, it is recommended to use 3rd party solution for object detection.

2.1.8. Display Sub System

1) Display Unit Block diagram

Display Unit has 5 overlay layers and controls final video output.

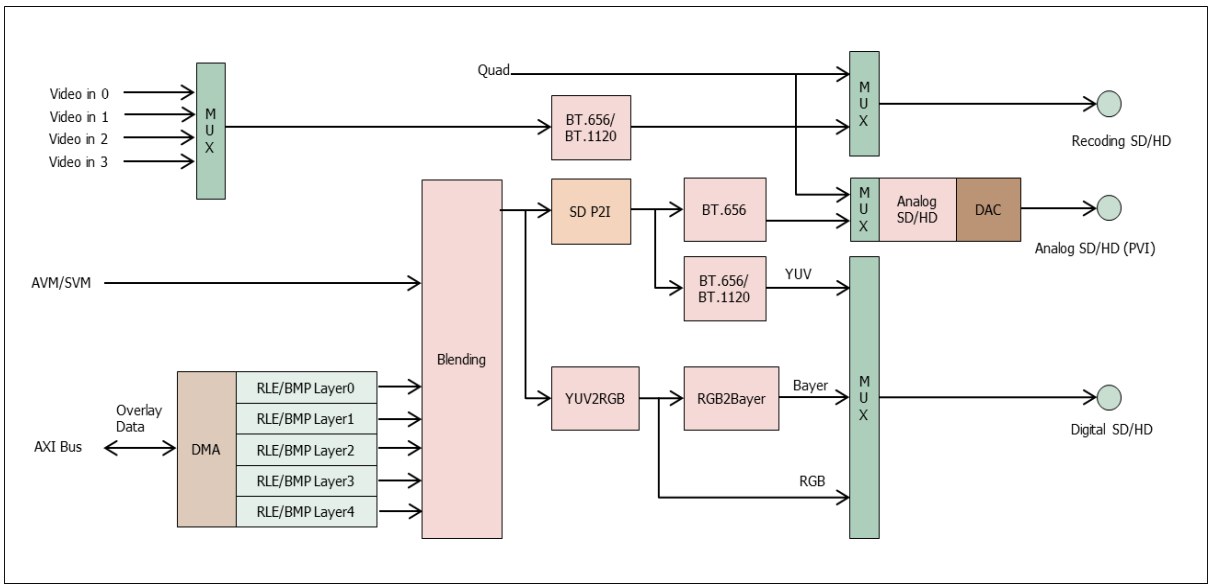
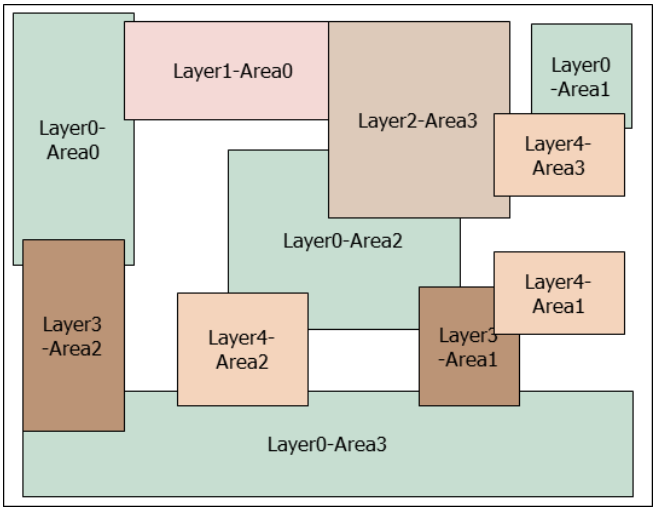


Figure 15 Display Unit Block Diagram

2) Overlay structure

DU has 5 overlay layers. Layer0 is the lowest layer, and layer 4 is the highest one. Each layer has 4 regions: Area0 ~ Area3. Areas of a layer cannot be overlapped, but those of different layers can be overlapped. Below figure shows an example of area setting for all layers.



2.1.9. File System

PI5008 File system is based on HCC FAT File system and used for SD/MMC.

1) HCC FAT File system feature

The main features of the system are the following:

ANSI 'C'.

Supports long filenames.

Supports multiple open files.

Cache options give improved performance.

Reentrant.

FAT-compatible.

Standard drivers are available for SD, SDHC, SDXC, MMC.

2) Cache operation

The file system includes two caching mechanisms to enhance performance: FAT and Write caching. FAT caching enables the file system to read several sectors from the FAT in one access, so that it's not necessary to read new FAT sectors so frequently.

Write cache defines the maximum number of sectors that can be written in one operation from the caller's data buffer.

3) How to use File system

Initializing file system

fs_init is used to initialize file system. It is called once at start-up.

f_enterFS

If the target system allows multiple tasks to use the file system, this function must be called by a task before it uses any other file management API functions.

f-initvolume

This function is used to initialize a volume. PI5008 SDK supports file system on SD/MMC.

To initialize a volume on SD/MMC, this function is called with following parameters.

f_initvolume(2, mmc_initfunc, 0);

Opening a file

PPAPI_FATFS_Open is used to open a file. The following opening modes are allowed:

Mode	Description
"r"	Open existing file for reading. The stream is positioned at the beginning of the file.
"r+"	Open existing file for reading and writing. The stream is positioned at the beginning of the file.
"w"	Truncate file to zero length or create file for writing. The stream is positioned at the beginning of the file.
"w+"	+" Open a file for reading and writing. The file is created if it does not exist; otherwise, it is truncated. The stream is positioned at the beginning of the file.
"a"	Open for appending (writing to the end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
"a+"	Open for reading and appending (writing to the end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

Table 15 PPAPI_FATFS_OPEN Opening Mode Table

For file system API, please refer to File System User Guide.

2.1.10. Exceptional Cases

PI5008 provides various ways to detect an abnormal status of the target system.

1) Abnormal hardware status

PI5008 generates an interrupt when a hardware problem is detected.

VPU error interrupt (intc 0x21010 [31:0])

VPU timeout error

PVI Rx PTZ error interrupt

PVI Rx PTZ Timeout error

2) Diagnosis function

PI5008 provides diagnosis function. If an abnormal case happens, diagnosis

interrupt happens. By checking diagnosis register, error type can be known.

A. gadc_diagnosis_1/gadc_diagnosis_0
   Extract ADC value of PVI Tx output and decide if PVI Tx is connected
B. du_freeze_3/du_freeze_2/du_freeze_1/du_freeze_0
C. Divide DU output into 4 areas which have the same size and decide if the output of each area is frozen.
D. sync_loss_det_4/sync_loss_det_3/sync_loss_det_2/sync_loss_det_1/sync_loss_det_0 : Monitors if sync of input camera is lost.
E. format_loss_det_4 ~ format_loss_det_0
   Monitors if the format of input camera is lost
F. genlock_sync_det/genlock_vref_loss
   Monitors genlock state
G. ch4_crc_bist_freeze ~/ch0_crc_bist_freeze
   Montiors CRC bit of 4 input channel and decide it is frozen
H. Quad_crc_bist_freeze
   Montiors CRC bit of 4 input Quad channel and decide it is frozen

2.1.11. IPC

PI5008 includes two CPUs – CPU0 and CPU1.
CPU0 is the main CPU and CPU1 is dedicated for specific functions: ISP and brightness control etc. PI5008K includes a mailbox to make two CPUs communicate with each other.
Mailbox has several registers.
MBOX_DATA 0~7 : used for common message
MBOX_CORE0_MSG, MBOX_CORE1_MSG: used to generate an interrupt for CPU 0 and 1 respectively. These registers are also used to send additional information.
[0] bit is used for interrupt request signal and MS 31bit is used to send additional information.

ADDR	Register	Bit	Comment
0x00	MBOX Data 0	[31:0]	Core 0 message 0 (to core 0)
0x04	MBOX Data 1	[31:0]	Core 0 message 1 (to core 0)
0x08	MBOX Data 2	[31:0]	Core 0 message 2 (to core 0)
0x0C	MBOX Data 3	[31:0]	Core 1 message 0 (to core 1)
0x10	MBOX Data 4	[31:0]	Core 1 message 1 (to core 1)
0x14	MBOX Data 5	[31:0]	Core 1 message 2 (to core 1)
0x18	MBOX Data 6	[31:0]	ISP ready pattern(0x5008CAFE)
0x1C	MBOX Data 7	[31:0]	Main ready pattern(0x5008BEEF)
0x100	MBOX Core0 Irq	[31:30] sender Id	Core0: 0, Core1: 1
		[29:28] receiver Id	Core0: 0, Core1: 1
		[27:24] attr	ACK Req: 1
		[23:16] cmd	
		[15:1] reserved	
		[0] irq trig	Generate mail box interrupt for core 0
0x104	MBOX Core1 Irq	[31:30] sender Id	Core0: 0, Core1: 1
		[29:28] receiver Id	Core0: 0, Core1: 1
		[27:24] attr	ACK Req: 1
		[23:16] cmd	
		[15:1] reserved	
		[0] irq trig	Generate mail box interrupt for core 1킴

Table 16 Mailbox register

For more details, please refer to PI5008K data sheet

1) Role of each register

I. Data register

i. Data registers are used to send data. 3 registers are assigned to each core.

ii. MBOX Data 0 ~ 2 : holds data to be sent to core0

iii. MBOX Data 3 ~ 5 : holds data to be sent to core 1

J. IRQ registers

i. Used to send command and generate interrupt.

ii. Generate interrupt by setting IRQ trigger bit after setting Sender id, receiver id, attr, cmd.

2) How to send data

a. If there is data to be sent, data will be written to the data register of the target CPU.

b. Generate interrupt by writing command to IRQ register.

c. IRQ register will be cleared by receiver. If IRQ is not cleared, no more data is written.

d. If ACK/NACK needs to be received, please set ack req bit of attr field.

3) Protocol

a. Core 0 -> Core 1

Command	Sender	Receiver	Cmd	Attr	Data3	Data4	Data5	Data6	
Ack	0x00/0x01	0x00/0x01	0x80	0x0	Req Cmd	-	-	-	
Nack	0x00/0x01	0x00/0x01	0x81	0x0	Req Cmd	-	-	-	
Print memory assign	0x00	0x01	0x01	0x1	Addr	Size	-	-	
Main Ready	0x00	0x01	0x02	0x0	-	-	-	0x5008BEEF	Note.1
Mem set	0x00	0x01	0x03	0x1	Data	Size	-	-	
BC On/Off	0x00	0x01	0x04	0x1	On(1)/Off(0)	-	-	-	

Table 17 IPC protocol (core0 -> core1)

Note1> If core 1 is not ready, set only data register without generating interrupt

b. Core 1 -> Core 0

Command	Sender	Receiver	Cmd	Attr	Data3	Data4	Data5	Data6	
Ack	0x01	0x00	0x80	0x0	Req Cmd	-	-	-	
Nack	0x01	0x00	0x81	0x0	Req Cmd	-	-	-	
Print message	0x01	0x00	0x10	0x1	-	-	-	-	
ISP Ready	0x01	0x00	0x11	0x0	-	-	-	0x5008CAFE	Note.2
Camera Plug In/Out	0x01	0x00	0x21	0x0	Cam3/2/1/0 In(1)/Out(0)				Note.3
Camera Map	0x01	0x00	0x22	0x0	Cam3/2/1/0 map Camx				Note.4

Table 18 IPC protocol (core1 -> core0)

Note2> If core 1 is not ready, set only data register without generating interrupt.

Note3> [b3]=Cam3, [b2]=Cam2, [b1]=Cam1, [b0]=Cam0. Value '1' means Plug-In. Value '0' means Plug-out.

Note3> [b31:b24]=Cam3 map chan, [b23:b16]=Cam2 map chan, [b15:b8]=Cam1 map chan, [b7:b0]=Cam0 map chan (?)

2.1.12. ISP(TBD)

2.2. System Resources

2.2.1. Interrupts

1) Interrupt List

No.	Interrupt	source	Status	Action	Comments
25~31					Not used
24	SUB_IRQ3	INTC	Use	a) SVM Output Setting View Mode : Sync after 2 Vsync Tilt Adjust : Processing inside handler b) Quad Vsync Capture for Dynamic Blending : Sync after 2 Vsync	SUB Interrupt 3 (Vsync)
23	VPU 2	VPU	Use	To send event to Task for LDWS (TBD)	Canny Edge
22	VPU 1	VPU	Use	To sync with task by sema(Dynamic blending)	VPU DMA done
21	VPU 0	VPU	Use	Processing inside Handler	On The Fly done, Feature Overflow
20	SVM	SVM	Use	Means an error in SVM, No Action	Overflow/Underflow
19	DU	DU	Use	Means an error in DU, No Action	MVI, OSD, RLE,
18	DDR	DDR	No use		Command Flush
17					
16	I^2C	I^2C	No use		4 I^2C is combined.
15	UART 2	UART	No use		
14	UART 1	UART	No use	MCU I/F. Will be used	

13	UART 0	UART	No use	Debug Console	
12	I²S	I²S	Use	PCM Buffer Swap	
11	GPIO (8 GPIO combined)	GPIO	No use	Will be used	
10	TIMER 1	TIMER 1	No use		
9	TIMER 0	TIMER 0	Use	Tick Timer (Tick = 1 msec)	
8	WDT	WDT	No use	Will be used to print the status	
7	DMA	DMA		Each channel will be served for a specific purpose. To sync with Task by Sema	
6	SPI (3 SPI combined)	SPI	No use		
5	SPI-MEMC	SPI-MEMC	No use		
4	SW-INT				
3	D10-INTC	Mail-Box	Use	Will be used	Interrupt from Sub CPU
2	SUB_IRQ2	INTC	N/A	Handled by CPU#1	SUB Int 2 (ISP)
1	SUB_IRQ1	INTC	Use	System reset, will print the status	SUB Int 1 (Diagnosis)
0	SUB_IRQ0	INTC	Use	PVI Cable plug-in/out, Send Event to monitor task	SUB Interrupt 0 (PVI Rx)

Table 19 Interrupts Table

No.	Interrupt	source	Comments
4~31	-		Not used
3	PVI RX 3	Video Input	PVI 3 PTZ, No Video
2	PVI RX 2	Video Input	PVI 3 PTZ, No Video
1	PVI RX 1	Video Input	PVI 3 PTZ, No Video
0	PVI RX 0	Video Input	PVI 3 PTZ, No Video

Table 20 INTC SUB 1(First/Cascaded) – PVI Rx

No.	Interrupt	source	Comments

24~31	-		Not used
23	gadc_diagnosis_1	GADC	
22	gadc_diagnosis_0	GADC	
21	du_freeze_3	DU	
20	du_freeze_2	DU	
19	du_freeze_1	DU	
18	du_freeze_0	DU	
17	sync_loss_det_4	VIN	
16	sync_loss_det_3	VIN	
15	sync_loss_det_2	VIN	
14	sync_loss_det_1	VIN	
13	sync_loss_det_0	VIN	
12	format_loss_det_4	VIN	
11	format_loss_det_3	VIN	
10	format_loss_det_2	VIN	
9	format_loss_det_1	VIN	
8	format_loss_det_0	VIN	
7	genlock_sync_det	VIN	
6	genlock_vref_loss	VIN	
5	ch4_crc_bist_freeze	VIN	
4	ch3_crc_bist_freeze	VIN	
3	ch2_crc_bist_freeze	VIN	
2	ch1_crc_bist_freeze	VIN	
1	ch0_crc_bist_freeze	VIN	
0	quad_crc_bist_freeze	QUAD	

Table 21 INTC SUB 2 (Secondary/Cascaded) - Diagnosis

No.	Interrupt	source	Comments
4~31	-		Not used
3	ISP 3	Video Input	ISP 3 Vsync/AE/AWB
2	ISP 2	Video Input	ISP 2 Vsync/AE/AWB
1	ISP 1	Video Input	ISP 1 Vsync/AE/AWB
0	ISP 0	Video Input	ISP 0 Vsync/AE/AWB

Table 22 INTC SUB 3 (Third / Cascaded) - ISP

No.	Interrupt	source	Comments
8~31			Not used
7	Display Vsync	DU	Display Vsync
6	Video Output Vsync	SVM	SVM-Output/DU-Input Vsync
5	Video Input 4 Vsync	Video Input	Video Input 4 Vsync
4	Video Input 3 Vsync	Video Input	Video Input 3 Vsync
3	Video Input 2 Vsync	Video Input	Video Input 2 Vsync
2	Video Input 1 Vsync	Video Input	Video Input 1 Vsync
1	Video Input 0 Vsync	Video Input	Video Input 0 Vsync
0	Quad Vsync	Quad	

Table 23 INTC SUB 4 (Fourth /Cascaded) - vsync

2) How To Register Interrupt Handler

PI5008 SDK provides API to register an interrupt handler

PP_RESULT_E OSAL_register_isr(sint32 IN vector, sys_os_isr_t IN isr, sys_os_isr_t* OUT old)

If you want to register interrupt handler for SVM block, you can do it by calling OSAL_register_isr(IRQ_SVM_VECTOR, svm_error_isr, pstOldIsr)

IRT_SVM_VECTOR : vector number of SVM interrupt

svm_error_isr : SVM interrupt handler

2.2.2. Memory Map

1. PI5008K memory map

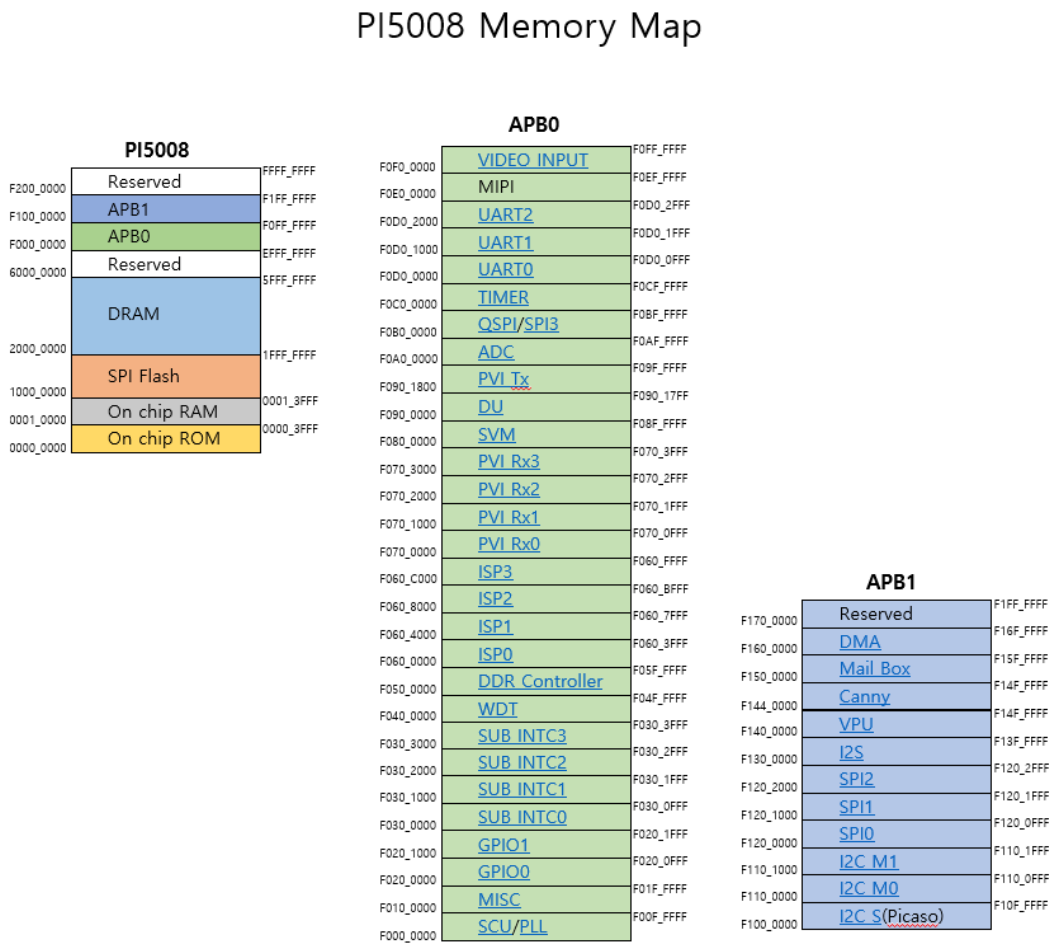


Figure 16 PI5008 Memory map

2. DRAM memory map*

PI5008 DRAM Map @720p(128MByte)

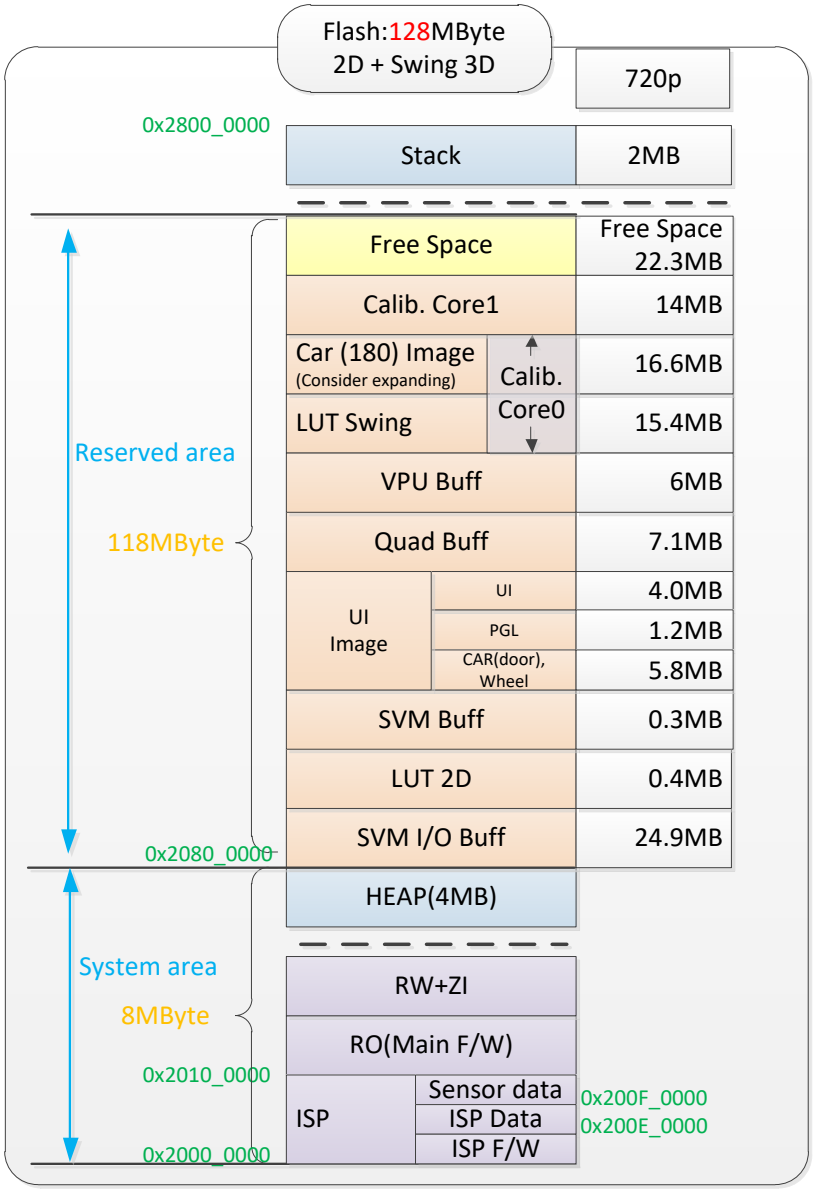


Figure 17 PI5008 DRAM Memory Map

3. Flash memory map*

PI5008 NOR Flash Map @32MByte

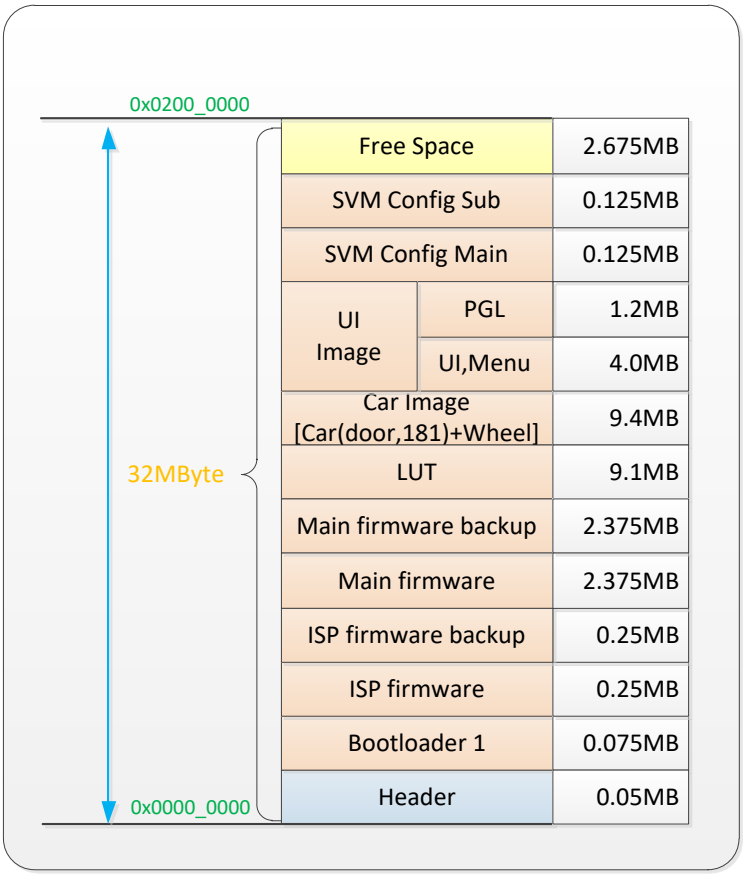


Figure 18 PI5008 NOR Flash Memory Map

PI5008 NAND Flash Map @64MByte

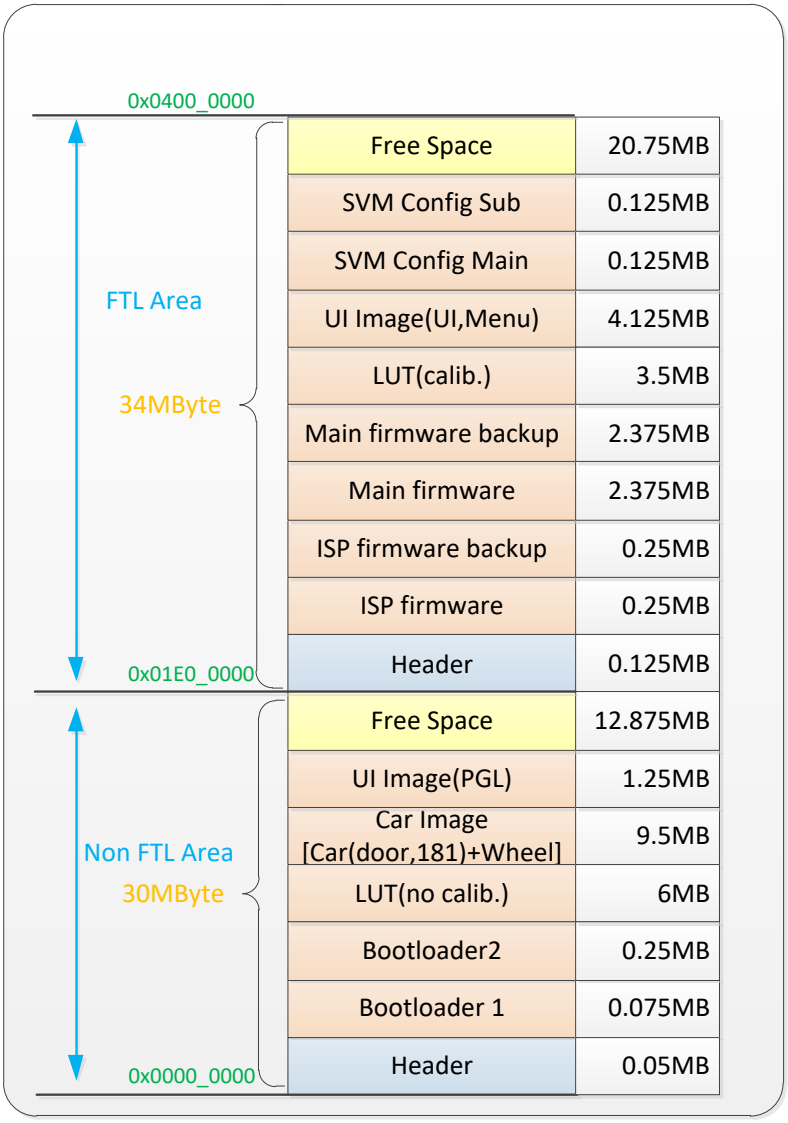


Figure 19 PI5008 NAND Flash Memory Map

Notes> DRAM/Flash memory map will be changed according to view modes.

2.3. Firmware Upgrade

Firmware upgrade can be done via SD card or SPI host and it is possible to select a method by setting system feature. It is possible to upgrade all sections or a specific part of flash memory. In case of NAND flash, only FTL area can be upgraded.

2.3.1. Firmware binary backup

If firmware upgrade is failed for some reason, system cannot work at all. To handle this case, PI5008 SDK has backup section in flash memory to save main and ISP firmware for emergency booting. These firmware is used only for system recovery.

During system booting, bootloader checks the validity of main firmware and boot up system using main firmware in backup section if main firmware is not valid.

After boot up, main firmware in backup section has to inform there is a problem in system by UI and recommends user to upgrade firmware again.

Main and ISP firmware in back up section can be upgraded only using PC download tool.

Update section define

Name	Value
FLASH_UPDATE_ISP_FW	0x0
FLASH_UPDATE_MAIN_FW	0x1
FLASH_UPDATE_SVM_LUT	0x2
FLASH_UPDATE_MENU	0x3
FLASH_UPDATE_GUI_IMG	0x4
FLASH_UPDATE_ISP_DATA	0x5
FLASH_UPDATE_SENSOR_DATA	0x6
FLASH_UPDATE_CALIB_MAIN	0x7
FLASH_UPDATE_CALIB_SUB	0x8

Table 24 Update section define

2.3.1. Flash upgrade by SD Card

When SD upgrade is selected and FWDN task receives command structure which includes flash memory section and file name to be upgraded, FWDN task read the file from SD card and start firmware upgrade. If a specific section needs to be upgraded, the size of new file to be smaller or same with the section. If the size of new file is bigger than the section size, upgrade will be failed. In this case, total flash image has to be upgraded by PC tool.

2.3.1.1. Flash Upgrade Command Structure

Name	Description
PP_CHAR szFileName[256]	Firmware file name
PP_U32 u32UpgradeSection	Upgrade section
PP_U32 u32Verify	Flag for verifying flash data or not
PP_U32 u32Version	Version

Table 25 Flash upgrade command structure

2.3.2. Flash upgrade by external SPI host

When SPI upgrade is selected and FWDN task received upgrade event, PI5008 set SPI slave mode and upgrade binary image which is sent from SPI. If a specific section needs to be upgraded, the size of new file to be smaller or same with the section. If the size of new file is bigger than the section size, upgrade will be failed. In this case, total flash image has to be upgraded by PC tool.

2.3.2.1. SPI communication protocol

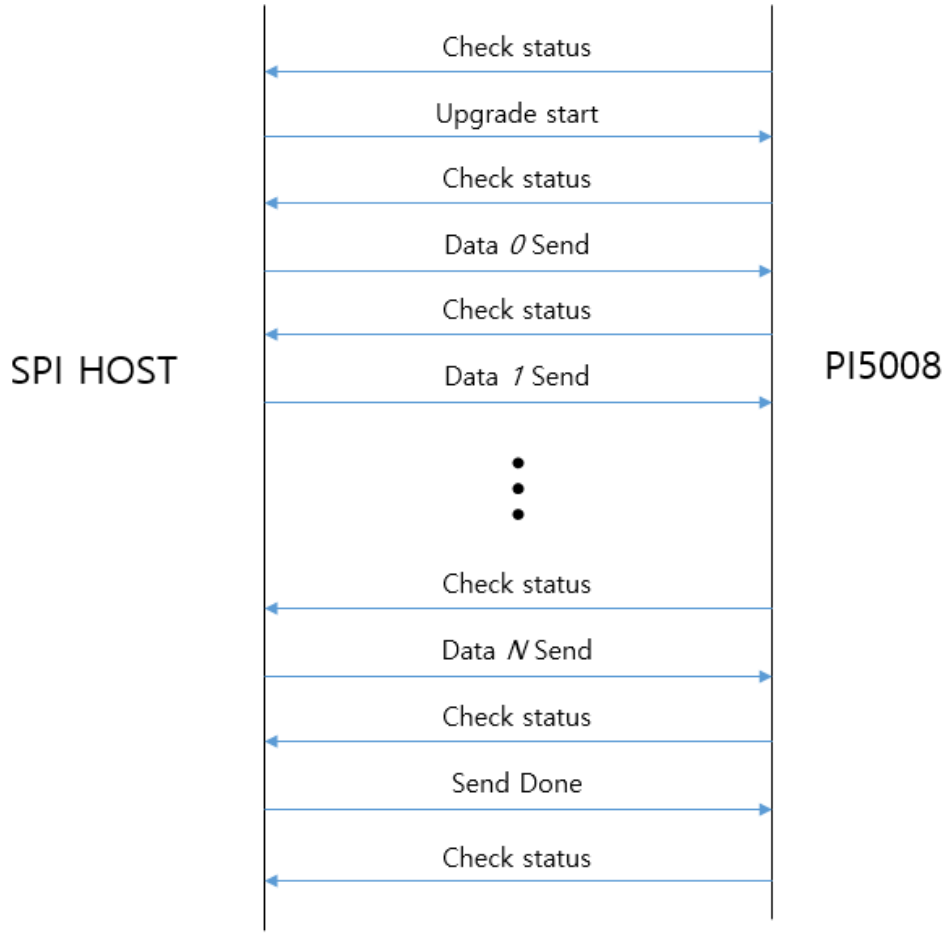


Figure 20 SPI communication sequence

1) Status check

This packet is used to check whether PI5008 is ready to start communication and to get last status. SPI Host will send this packet repeatedly until PI5008 is ready.

BYTE	MOSI Value	MISO Value
0	0x05	Dummy
1	Dummy	Dummy
2	Dummy	Dummy
3	Dummy	Bit[0] 0 : Not ready 1 : Ready
4	Dummy	Status high

5	Dummy	Status low

Table 26 Status check packet

2) Upgrade start

This packet is used to send information before starting upgrade. This packet has to be sent before starting upgrade each section.

BYTE	MOSI Value
0	SOF (0xbc)
1	Command 0 (0x20)
2	Command 1 (0x00)
3	Command 2 (0x00)
4 ~ 7	Packet size(0x14) The size from byte7 ~ last
8 ~ 11	Upgrade data size
12	Upgrade section
13	Verify flag
14	Current section count
15	Total section count
16 ~ 19	Version
20 ~ 23	Send data size at a time
24 ~ 27	Checksum

Table 27 Upgrade start packet

3) Data

BYTE	MOSI Value
0 ~	Upgrade data Data size is defined in upgrade start packet byte20 ~ 23. Last data packet could be less than this data size.
Last 4 bytes	4 bytes

Table 28 Data packet

4) Upgrade data send done

SPI host sends this packet to inform the end of upgrade data.

BYTE	MOSI Value
0	SOF (0xbc)
1	Command 0 (0x22)
2	Command 1 (0x00)
3	Command 2 (0x00)
4 ~ 7	Packet size (0x04)
8 ~ 11	Checksum

Table 29 Upgrade data send done packet

2.3.2.2. SPI Host Programming Sequence

1) Scenario

A. ISP firmware and main firmware will be upgraded

B. ISP firmware size is 128KB, main firmware size is 1MB.

2) Sequence

A. Check status

B. Upgrade start

1. Upgrade data size: 0x20000
2. Upgrade section: 0x0 (ISP firmware section)
3. Current section count: 0x1
4. Total section count: 0x2

C. Check status

D. Upgrade data send (ISP firmware)

E. Repeat C, D

F. Check status

G. Upgrade data send done

H. Check status

I. Upgrade start

1. Upgrade data size: 0x100000
2. Upgrade section: 0x1 (main firmware section)
3. Current section count: 0x2
4. Total section count: 0x2

J. Check status

K. Upgrade data send (main firmware)

L. Repeat J, K

M. Check status

N. Upgrade data send done

O. Check status

3. Task Reference Guide

3.1. Structure Definition

SVM SDK uses several structures to handle OS resources such as task, event group and message queue. There are 3 important structures.

1) stHandleCfg

This structure is used to hold the task handler. It is also used to hold event group and message queue handler which are used in the task.

```
typedef struct stHandleCfg
{
    TaskHandle_t phTask;// handler for task
    QueueHandle_t phQueue;// handler for message queue
    EventGroupHandle_t phEventGroup;// handler for event group
} HandleCfg_t;
```

SVM SDK declares global variable gHandle to hold handles for all tasks,
HandleCfg_t gHandle[MAX_TASK];

2) stQueueCfg

This structure is used to hold information for the message queue.

```
typedef struct stQueueCfg
{
    uint8 maxNumItem;// The maximum number of items
    char* name;
    uint32 sizeItem;// The size in bytes of each data item
    void *pvParameters;
} QueueCfg_t;
```

3) stEventGroupCfg

This structure is used to hold information for event group

```
typedef struct stEventGroupCfg
```

```
{
    uint32 maxNumBit;
    char* name;
    void *pvParameters;
} EventGroupCfg_t;
```

4. Tasks

4.1. Monitor Task

MonitorTask wakes up every 20msec and monitors system status. It is also woken up by user input from UART and send an event to the UI Task. MonitorTask handles progressive bar during the calibration process. It also send camera plug-in/out event to UI task.

1) Base code

```
PP_VOID vTaskMonitor(PP_VOID *pvData)
{
 PP_S32 timeOut = SYS_OS_SUSPEND; //msec
 EventBits_t eventWaitBits;
 EventBits_t eventResultBits;
 TaskParam_t *pstParam = (TaskParam_t *)pvData;
 PP_S32 i;

 SetTaskMonitorTimer(); //timer set

for(;;)
{
 pstParam->u32TaskLoopCnt++;

 if(gHandle[TASK_MONITOR].phEventGroup)
 {
 eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
 eventResultBits = OSAL_EVENTGROUP_CMD_WaitBits(gHandle[TASK_MONITOR].phEventGroup,
 eventWaitBits, pdTRUE, pdFALSE, timeOut);

 if ( eventResultBits == 0 )
 {
  continue;
 }
```

```
    else
    {
     // process event bit
     if(eventResultBits & (1<<EVENT_MONITOR_INIT))
     {
      //Called once when start task.
      //No message Queue.
     }/
     if(eventResultBits & (1<<EVENT_MONITOR_MSG))
     {
      if(gHandle[TASK_MONITOR].phQueue)
      {
       int queueCnt = 0;
       stQueueItem queueItem;
       if( (queueCnt = OSAL_QUEUE_CMD_GetCount(gHandle[TASK_MONITOR].phQueue)) > 0)
       {
        for(i = 0; i < queueCnt; i++)
        {
         if(  OSAL_QUEUE_CMD_Receive(gHandle[TASK_MONITOR].phQueue,  &queueItem,  0)  ==
eSUCCESS )
         {
          if(queueItem.u32Cmd == CMD_GET_REM_FROM_UART_ISR)
          {
           if( (queueItem.u32Length > 0) && (queueItem.pData != NULL))
           {
            AppTask_SendCmd_To_UI_Remocon_Cmd(queueItem.pData,queueItem.u32Length);
                    //parsing uart data & send cmd to UI Task
            memset(uart_temp,0,1024);
           }
          }
          else if(queueItem.u32Cmd == CMD_MONITOR_CAMERA_PLUG)
          {/*{{{*/
           if( (queueItem.u32Length > 0) && (queueItem.pData != NULL) )
           {
            uint32_t u32StatusPlug;
            u32StatusPlug = *(uint32_t *)queueItem.pData;
```

```
                AppTask_SendCmd(CMD_UI_CAMERA_PLUG, TASK_MONITOR, TASK_UI, 0, &u32StatusPlug,
                sizeof(u32StatusPlug), 1000);
              }
             }
            }
           }
          }
          else
          {
            //ignore
          }
         else
         {
          LOG_CRITICAL("Invalid handle.₩n");
         }

       }
      }
     }
    }
   return;
}
```

2) OS Resource

Event Group : gHandle[TASK_MONITOR].phEventGroup

Message Queue : gHandle[TASK_ MONITOR].phQueue

Queue Command:

```
typedef enum {
        CMD_MONITOR_ALIVE = MAX_CMD_COMMON, //fix value 0
        CMD_MONITOR_NUM1,    //not used
        CMD_GET_REM_FROM_UART_ISR,
        CMD_DBG_PRINT_MSG_ON,
        CMD_DBG_PRINT_MSG_OFF,
        CMD_MONITOR_CAMERA_PLUG,
        MAX_CMD_MONITOR
    }eCMD_SET_MONITOR;
```

#define MONITOR_TASK_WAKEUP_TIME_MS 20

This value will be used by SetTaskMonitorTimer();

3) UART Key list

PI5008K receives user input via UART.

User inputs from UART are as follows.

```
typedef enum {
    CMD_UI_ALIVE = MAX_CMD_COMMON, //fix value 0
    CMD_UI_FWDN,
    CMD_UI_KEY_UP,
    CMD_UI_KEY_UP_LONG,
    CMD_UI_KEY_DOWN,
    CMD_UI_KEY_DOWN_LONG,
    CMD_UI_KEY_LEFT,
    CMD_UI_KEY_LEFT_LONG,
    CMD_UI_KEY_RIGHT,
    CMD_UI_KEY_RIGHT_LONG,
    CMD_UI_KEY_CENTER,
    CMD_UI_KEY_CENTER_LONG,
    CMD_UI_KEY_MENU,
    CMD_UI_KEY_MENU_LONG,
    CMD_UI_INIT_SCENE,
    CMD_UI_NEXT_SCENE,
    CMD_UI_PREV_SCENE,
    CMD_UI_DIALOG,
    CMD_UI_TRIGGER_REVERSE,
    CMD_UI_TRIGGER_TURN,
    CMD_UI_TRIGGER_EMERGENCY,
    CMD_UI_TRIGGER_OFF,
    CMD_UI_CAMERA_PLUG,

    MAX_CMD_UI
}eCMD_SET_UI;
```

4) Sample Code

This code shows how SVM SDK handles user input. The sequence is as below.

A. If there is a user input, the UART interrupt handler will be executed. UART interrupt handler will send an event to MonitorTask by using AppTask_SendCmdFromISR.
B. MonitorTask will wake up and parse message queue to find out command type.
C. If message type is CMD_GET_REM_FROM_UART_ISR, Monitor task will send an event to UI Task using AppTask_SendCmd_To_UI_Remocon_Cmd.

UART Interrupt handler

UART Interrupt handler receive UART data and send it to Monitor Task.

```
#define BUF_SIZE 1024
PP_U8 uart_buf[BUF_SIZE] = {0};

PP_VOID uart_rx_rem_isr (PP_VOID)
{
        /*Save UART data */
        while( PPDRV_UART_GetRxReady(UART_CH) )
        {
          uart_buf[rx_idx++] = PPDRV_UART_GetRxData(UART_CH);
          if(rx_idx >= BUF_SIZE)
          rx_idx = 0;
        }
/*If uart data is (enter key==₩n), send data to Monitor Task */
        if(uart_buf[rx_idx-1]=='₩n') //end of remocon protocol
        {
            AppTask_SendCmdFromISR(CMD_GET_REM_FROM_UART_ISR, TASK_MONITOR, TASK_MONITOR, 0,
            uart_buf, rx_idx);
          memset(uart_buf,0,BUF_SIZE);
          rx_idx=0;
        }
}
```

4.2. UI Task

UI Tasks handles all UI functions. UI event such as user input and display update is

handled by UI Task.

4.2.1. Event definition

Name	Bit position	Description
EVENT_UI_INIT	Bit 0	Initial event when the system is started
EVENT_UI_MSG	Bit 1	UI message event

Table 30 Event definition

4.2.2. Queue Command

Name	Description
CMD_UI_ALIVE	UI alive command
CMD_UI_FWDN	Firmware update command
CMD_UI_KEY_UP	Remote control "up" key command
CMD_UI_KEY_UP_LONG	Remote control "up" long-key command
CMD_UI_KEY_DOWN	Remote control "down" key
CMD_UI_KEY_DOWN_LONG	Remote control "down" long-key command
CMD_UI_KEY_LEFT	Remote control "left" key
CMD_UI_KEY_LEFT_LONG	Remote control "left" long-key command
CMD_UI_KEY_RIGHT	Remote control "right" key
CMD_UI_KEY_RIGHT_LONG	Remote control "right" long-key command
CMD_UI_KEY_CENTER	Remote control "center" key
CMD_UI_KEY_CENTER_LONG	Remote control "center" long-key command
CMD_UI_KEY_MENU	Remote control "menu" key
CMD_UI_KEY_MENU_LONG	Remote control "menu" long-key command
CMD_UI_INIT_SCENE	Scene initialize command
CMD_UI_NEXT_SCENE	Next scene command
CMD_UI_PREV_SCENE	Previous scene command

CMD_UI_DIALOG	Dialog box command
CMD_UI_TRIGGER_REVERSE	Reverse trigger command
CMD_UI_TRIGGER_TURN	Turn trigger command
CMD_UI_TRIGGER_EMERGENCY	Emergency trigger command
CMD_UI_TRIGGER_DRIVE	Driver trigger command
CMD_UI_CAMERA_PLUG	Camera plug command
CMD_UI_CACHE_TASK_DONE	Cache task done command
CMD_UI_CAR_DOOR	Open car door command

Table 31 Queue Command

4.2.3. Task Code

```
for( ;; )
{
  eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
  eventResultBits = OSAL_EVENTGROUP_CMD_WaitBits(gHandle[TASK_UI].phEventGroup, eventWaitBits,
  pdTRUE, pdFALSE, timeOut);
  if(eventResultBits & (1<<EVENT_UI_MSG))
  {
    if( (queueCnt = OSAL_QUEUE_CMD_GetCount(gHandle[myHandleNum].phQueue)) > 0)
    {
      for(i = 0; i < queueCnt; i++)
      {
        if( OSAL_QUEUE_CMD_Receive(gHandle[myHandleNum].phQueue, &queueItem, 0) ==
        eSUCCESS )
        {
         if(queueItem.u32Cmd==CMD_UI_FWDN)
         {
              AppTask_SendCmd(CMD_FWDN_FLASH_UPDATE, TASK_UI, TASK_FWDN,
              (1<<QUEUE_CMDATTR_BIT_REQACK), (PP_VOID *)queueItem.pData,
              queueItem.u32Length, 3000)}
         else if(queueItem.u32Cmd==CMD_UI_CAMERA_PLUG)
         {
            PPAPI_SVM_SetInputReplacementColorOnOff(u32StatusPlug>>16, !(u32StatusPlug&1));
         }
         else
```

```
            {
                UI_Process();
            }
          }
        }
      }
    }
}
```

4.3. Display Task

Display Task will perform a display update which is requested by other tasks than UITask.

4.3.1. Event

Name	Bit position	Description
EVENT_DISPLAY_INIT	Bit 0	Initial event when the system is started
EVENT_DISPLAY_MSG	Bit 1	Display message event

Table 32 Event

4.3.2. Queue Command

Name	Description
CMD_DISPLAY_ALIVE	Display alive command
CMD_DISPLAY_POPUP_TIMER	Pop-up timer on & off command
CMD_DISPLAY_POPUP_ON	Pop-up on command
CMD_DISPLAY_POPUP_OFF	Pop-up off command
CMD_DISPLAY_PROGRESSBAR_TIMER_ON	Progress bar timer on command
CMD_DISPLAY_PROGRESSBAR_TIMER_OFF	Progress bar timer off command
CMD_DISPLAY_PROGRESSBAR_ON	Progress bar on command
CMD_DISPLAY_PROGRESSBAR_OFF	Progress bar off command
CMD_DISPLAY_PGL_ANGLE	PGL angle update and display command

Table 33 Queue Command

4.3.3. Task Code

```
for( ;; )
{
    eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
    eventResultBits = OSAL_EVENTGROUP_CMD_WaitBits(gHandle[myHandleNum].phEventGroup,
    eventWaitBits, pdTRUE, pdFALSE, timeOut);
    if(eventResultBits & (1<<EVENT_DISPLAY_MSG))
    {
     if( (queueCnt = OSAL_QUEUE_CMD_GetCount(gHandle[myHandleNum].phQueue)) > 0)
     {
        for(i = 0; i < queueCnt; i++)
        {
            if( OSAL_QUEUE_CMD_Receive(gHandle[TASK_DISPLAY].phQueue, &queueItem, 0) ==
            eSUCCESS )
           {
             if(queueItem.u32Cmd==CMD_DISPLAY_POPUP_TIMER)
             {
                    SetPopUpTimer();
             }
             else if(queueItem.u32Cmd==CMD_DISPLAY_POPUP_ON)
             {
                    PPAPI_DISPLAY_POPUP_On(queueItem.u16Attr);
             }
             else if(queueItem.u32Cmd==CMD_DISPLAY_POPUP_OFF)
             {
                    PPAPI_DISPLAY_POPUP_Off();
             }
             else
             {
                    ......
             }
           }
        }
     }
  }
```

}

4.4. DB(Dynamic Blending) Task

Dynamic blending will be done as follows.

1) **MonitorTask will send the event to DBTask to start/stop dynamic blending.**
2) DBTask sets VPU and waits for Semaphore(DB_SEMA)
3) VPU Interrupt handler set DB_SEMA
4) DBTask is unblocked and run object detection software algorithm.
5) DBTask set proper blend map.

Redo list (2) ~ (5) until stop event receives.

4.4.1. Kernel Resources

1) Event Group

Name	Bit position	Description
EVENT_DYNBLEND_INIT	Bit 0	Initialize Dynamic blending event.
EVENT_DYNBLEND_MSG	Bit 1	DB message event
EVENT_DYNBLEND_OPER	Bit 2	Dynamic blending operate event.

Table 34 Event definition

2) Binary Semaphore

Name	Description
lockFast	Fast brief process semaphore of VPU block.
lockHammingD	Hamming Distance process semaphore of VPU block.

Table 35 Binary semaphore

3) Message Queue

Name	Description
CMD_DYNBLEND_ALIVE	DB task alive command
CMD_DYNBLEND_OPER	DB start or stop command

Table 36 Queue Command

4.4.2. Task code

```
for(;;)
{
    pstParam->u32TaskLoopCnt++;

    if(gHandle[myHandleNum].phEventGroup)
    {
        eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
        eventResultBits =
        OSAL_EVENTGROUP_CMD_WaitBits(gHandle[myHandleNum].phEventGroup, eventWaitBits,
        pdTRUE, pdFALSE, timeOut);

        if ( eventResultBits == 0 )
        {
            continue;
        }
        else
        {
            // process event bit
            if(eventResultBits & (1<<EVENT_DYNBLEND_INIT))
            {/*{{{*/
                //Called once when start task.
                //No message Queue.

            }/*}}}*/
            if(eventResultBits & (1<<EVENT_DYNBLEND_MSG))
            {
              {
                    if(gHandle[myHandleNum].phQueue)
                    {
                        int queueCnt = 0;
                        stQueueItem queueItem;
                        if( (queueCnt =
```

```
OSAL_QUEUE_CMD_GetCount(gHandle[myHandleNum].phQueue)) > 0)
{
    for(i = 0; i < queueCnt; i++)
    {
        if( OSAL_QUEUE_CMD_Receive(gHandle[myHandleNum].phQueue,
        &queueItem, 0) == eSUCCESS )
        {

            if(queueItem.u32Cmd == CMD_DYNBLEND_ALIVE)
            {
            }
            else if(queueItem.u32Cmd == CMD_DYNBLEND_OPER)
            {
                uint32_t bEnable = *(uint32_t *)queueItem.pData;
                if(bEnable)
                {
                    bDBLoop = PP_TRUE;
                    PPAPI_SVM_SetDynamicBlendingOnOff(PP_TRUE);

                    // set fast config
                    {
                        PPAPI_VPU_FAST_GetConfig(&stVPUFBConfig);
                        //stVPUFBConfig.fld.fast_n = FAST_PX_NUM;
                        //stVPUFBConfig.fld.ch_sel = eVpuChannel;
                        //stVPUFBConfig.fld.use_5x3_nms = FALSE;
                        //stVPUFBConfig.fld.brief_enable = TRUE;
                        //stVPUFBConfig.fld.roi_enable = FALSE;
                        //stVPUFBConfig.fld.scl_enable                    =
                        bScaleEnable;
                        stVPUFBConfig.fld.dma2otf_enable = FALSE;
                        stVPUFBConfig.fld.otf2dma_enable = FALSE;
                        PPAPI_VPU_FAST_SetConfig(&stVPUFBConfig);
                    }
                    //-----------------------------------------------------
----------
```

```
                ObjectDetector_InitParam(odParam);

OSAL_EVENTGROUP_CMD_SetBits(gHandle[myHandleNum].phEv
entGroup, (1<<EVENT_DYNBLEND_OPER)); //self send event
            }
            else
            {
                bDBLoop = PP_FALSE;
                PPAPI_SVM_SetDynamicBlendingOnOff(PP_FALSE);
                PPAPI_VPU_FAST_Stop();
            }
        }

        if(queueItem.u16Attr)
        {
            if(queueItem.u16Attr                                &
              (1<<QUEUE_CMDATTR_BIT_REQACK))
            {
                if(queueItem.u32Cmd == CMD_DYNBLEND_ALIVE)
                {

                    AppTask_SendCmd(CMD_ACK,
                    TASK_DYNBLEND,   queueItem.u16Sender,   0,
                    &pstParam->u32TaskLoopCnt,
                    sizeof(pstParam->u32TaskLoopCnt), 1000);
                }
                else
                {
                    AppTask_SendCmd(CMD_ACK,
                    TASK_DYNBLEND,   queueItem.u16Sender,   0,
                    NULL, 0, 1000);
                }
            }
        }

        if( (queueItem.u32Length > 0) && (queueItem.pData !=
```

```
NULL) )
                                        {
                                            OSAL_free(queueItem.pData);
                                        }

                                    }
                                }
                            }
                            else
                            {
                                //ignore
                            }
                        }
                        else
                        {
                        }

                    }
                }
                if(eventResultBits & (1<<EVENT_DYNBLEND_OPER))
                {/*{{{*/
                    if(bDBLoop)
                    {
                        runMax = 1;
                        u32TimeOut = 1000;
                        PPAPI_VPU_FAST_Start(runMax, u32TimeOut); //runMax = runcount, -1 =
                        endless continue.
                        // When manual, 1frame + alpha.

                        PPAPI_VPU_GetStatus(&stVPUStatus);

                        if( (PPAPI_VPU_HAMMINGD_Start(&stVPUStatus, 1000)) == eSUCCESS)
                        //2~3msec
                        {

                            //LOG_DEBUG("Get HamminD result of VPU.₩n");
```

```
{/*{{{*/
    pVPU_MATCHING_RESULT_POS_T hw_HD;
    for (zoneNum = 0; zoneNum < eVPU_ZONE_MAX; zoneNum++)
    {
        outTotalNumMotion[zoneNum]                                  =
        stVPUStatus.u32HDMatchResultCount[zoneNum];
        hw_HD                                                       =
(pVPU_MATCHING_RESULT_POS_T)stVPUStatus.pVBufHDMatchAddr[zoneNum];
        for (i = 0; i < stVPUStatus.u32HDMatchResultCount[zoneNum];
            i++)
        {
            outMotionPair[zoneNum][i].first.x = hw_HD[i].x1;
            outMotionPair[zoneNum][i].first.y = hw_HD[i].y1;
            outMotionPair[zoneNum][i].second.x = hw_HD[i].x2;
            outMotionPair[zoneNum][i].second.y = hw_HD[i].y2;
        }
    }
}/*}}}*/

// process libOD function.
{/*{{{*/
    // After VPU operation, pass matching pair position and matching
pair number at each zone.
    // first - position for previous frame
    // second - position for current frame

    for (i = 0; i < TOTAL_ZONE_NUM; i++) {
        currNumMotion[i] = outTotalNumMotion[i];

        for (iter = 0; iter < currNumMotion[i]; iter++)
        {
            currMotionPair[i][iter].first.x = outMotionPair[i][iter].first.x;
            currMotionPair[i][iter].first.y = outMotionPair[i][iter].first.y;
            currMotionPair[i][iter].second.x                        =
            outMotionPair[i][iter].second.x;
            currMotionPair[i][iter].second.y                        =
```

```
                                    outMotionPair[i][iter].second.y;
                                }
                            }

                            ObjectDetector (currMotionPair, currNumMotion, prevMotionPair,
                            prevNumMotion, odParam, statusNear, statusAway);

                            odBlendControl (1, statusNear, statusAway, cntCornerNear,
                            cntCornerAway, blendCorner);

                            PPAPI_SVM_SetDynamicBlending(eSVMAPI_DB_FRONTLEFT,
                            blendCorner[0]);
                            PPAPI_SVM_SetDynamicBlending(eSVMAPI_DB_FRONTRIGHT,
                            blendCorner[1]);
                            PPAPI_SVM_SetDynamicBlending(eSVMAPI_DB_REARLEFT,
                            blendCorner[2]);
                            PPAPI_SVM_SetDynamicBlending(eSVMAPI_DB_REARRIGHT,
                            blendCorner[3]);
                        }/*}}}*/
                    }

                    //OSAL_sleep(80); //check fastbrief time <= 2frame

                    OSAL_EVENTGROUP_CMD_SetBits(gHandle[myHandleNum].phEventGroup,
                    (1<<EVENT_DYNBLEND_OPER)); //self send event
                }
                else
                {
                }
            }/*}}}*/
        }
    }
}
for (zoneNum = 0; zoneNum < eVPU_ZONE_MAX; zoneNum++)
{
    OSAL_free(currMotionPair[zoneNum]);
```

```
            OSAL_free(outMotionPair[zoneNum]);
        }

        return;
}
```

4.5. Calibration Task

CalibrationTask performs the following functions

1) **Off-line camera calibration**

   CalibrationTask captures image from the selected camera and performs camera calibration.

   The Captured image will be saved optionally.

2) **View Generation**

   CalibrationTask generates 3 LUTs for each view according to configuration data. The total number of views is fixed.

3) **Preview & Save Merged LUT and calibration data**

   CalibrationTask preview and saves Merged LUT and camera parameter at flash memory

4.5.1. On board calibration

After MonitorTask sends user key input to UITask, UITask sends an event to make calibrationTask to start offline calibration. If Calibration Task receives offline calibration start event command(CMD_OFFCALIB_START), it starts to get camera position and angle information. Message queue holds a current step of the offline calibration process. (Please refer to sample code of 3.5.4).

It is possible to select one or all camera target camera using UI. If all cameras are selected, calibration for 4 cameras will be done.

If test patterns are found during OFFCALIB_FIND_PATTERN step, position and angle information will be obtained successfully. This information will be used during view generation process. If patterns are not found, OFFCALIB_END step will be started. The calibration process is terminated and memory allocated for calibration is freed.

Kernel Resource>

Event Group : gHandle[TASK_CALIBRATION].phEventGroup

Message Queue : gHandle[TASK_CALIBRATION].phQueue

Queue Command : CMD_OFF_CALIB_START

4.5.2. View Generation

If CalibrationTask receives LUT Start event command (CMD_VIEWGEN_START), it sets GenerateMode to 1 and generates LUT for all view mode sequentially. After LUTs for all view modes are generated Preview image will be shown when each view is generated. LUT data will be saved at flash memory just after preview image is shown.

Kernel Resource>

Event Group : gHandle[TASK_CALIBRATION].phEventGroup

Message Queue : gHandle[TASK_CALIBRATION].phQueue

Queue Command : CMD_VIEWGEN_START

4.5.3. Preview & Merged LUT Data save

If CalibrationTask receives CMD_VIEWGEN_START Command with VIEWGEN_MERGE_SET Attribution, Generated LUTs are merged so that SVM block can use it. Surround view image generated by SVM block will be shown and merged LUTs are saved at Flash memory.

Event Group : gHandle[TASK_CALIBRATION].phEventGroup

Message Queue : gHandle[TASK_CALIBRATION].phQueue

Queue Command : CMD_VIEWGEN_START

Queue Attribute : VIEWGEN_MERGE_SET

4.5.4. Task Code

```
typedef enum ppOFFCALIB_PROCESS_STEP_E
{
        OFFCALIB_WAIT_CMD=0, // 0x00
```

```
        OFFCALIB_START, //0x01
        OFFCALIB_CAPTURE,//0x02
        OFFCALIB_GET_CNF,//0x03
        OFFCALIB_GET_WOLRD_POINT,//0x04
        OFFCALIB_FIND_PATTERN,//0x05
        OFFCALIB_GET_INTRINSIC_PARAM,//0x06
        OFFCALIB_GET_CAMERA_POSITION,//0x07
        OFFCALIB_GET_FEATURE_POINT,//0x08
        OFFCALIB_END,//0x09
} OFFCALIB_PROCESS_STEP_E;
/* Offline Calibration Process Step */

typedef enum ppVIEWGEN_PROCESS_STEP_E
{
        VIEWGEN_WAIT_CMD=0, // 0x00
        VIEWGEN_START ,//0x01
        VIEWGEN_PREPARE_MAKE_TOP_2D_LUT,// 0x02
        VIEWGEN_GET_TOP_2D_FB_LUT,// 0x03
        VIEWGEN_GET_TOP_2D_LR_LUT,// 0x04
        VIEWGEN_GET_TOP_2D_BLEND_LUT,// 0x05
        VIEWGEN_GENERATE_TOP_2D_STATIC_PGL,// 0x06
        VIEWGEN_GENERATE_TOP_2D_DYNAMIC_PGL,// 0x07

        VIEWGEN_PREPARE_MAKE_RS_UNDIS_LUT, // 0x08
        VIEWGEN_GET_RS_UNDIS_FB_LUT, // 0x09
        VIEWGEN_GET_RS_UNDIS_LR_LUT,// 0x0A
        VIEWGEN_GET_RS_UNDIS_BLEND_LUT,// 0x0B
        VIEWGEN_GENERATE_RS_UNDIS_STATIC_PGL, // 0x0C
        VIEWGEN_GENERATE_RS_UNDIS_DYNAMIC_PGL,// 0x0D

        VIEWGEN_PREPARE_MAKE_RS_3D_LUT, // 0x0E
        VIEWGEN_GET_RS_3D_FB_LUT, // 0x0F
        VIEWGEN_GET_RS_3D_LR_LUT,// 0x10
        VIEWGEN_GET_RS_3D_BLEND_LUT,// 0x11
        VIEWGEN_GENERATE_RS_3D_STATIC_PGL, // 0x12
        VIEWGEN_GENERATE_RS_3D_DYNAMIC_PGL,// 0x13
```

```
        VIEWGEN_PREPARE_MAKE_RS_REAR_LUT, // 0x14
        VIEWGEN_GET_RS_REAR_FB_LUT, // 0x15
        VIEWGEN_GET_RS_REAR_LR_LUT,// 0x16
        VIEWGEN_GET_RS_REAR_BLEND_LUT,// 0x17
        VIEWGEN_GENERATE_RS_REAR_STATIC_PGL, // 0x18
        VIEWGEN_GENERATE_RS_REAR_DYNAMIC_PGL,// 0x19


        VIEWGEN_MERGE_SET,// 0x1A
        VIEWGEN_END, // 0x1B
} VIEWGEN_PROCESS_STEP_E;
/* View Generation Process Step */

PP_VOID vTaskCalibration(PP_VOID *pvData)
{
  PP_S32 myHandleNum = TASK_CALIBRATION;
  PP_S32 timeOut = 1000; //msec
  EventBits_t eventWaitBits;
  EventBits_t eventResultBits;
  TaskParam_t *pstParam = (TaskParam_t *)pvData;

  PP_U32 i;
  for(;;) //infinite loop
  {
    pstParam->u32TaskLoopCnt++;
    if(gHandle[myHandleNum].phEventGroup)
    {
      eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
      eventResultBits    =    OSAL_EVENTGROUP_CMD_WaitBits(gHandle[myHandleNum].phEventGroup,
eventWaitBits, pdTRUE, pdFALSE, timeOut);
      if ( eventResultBits == 0 ) //if eventgroup command not exist,
      {
        continue;
      }
      else
```

```
    {
     if(eventResultBits & (1<<EVENT_CALIBRATION_MSG))
     {
      if(gHandle[myHandleNum].phQueue)
      {
       PP_S32 queueCnt = 0;
       stQueueItem queueItem;
       if( (queueCnt = OSAL_QUEUE_CMD_GetCount(gHandle[myHandleNum].phQueue)) > 0)
       {
        for(i = 0; i < queueCnt; i++)
        {
         if(  OSAL_QUEUE_CMD_Receive(gHandle[myHandleNum].phQueue,  &queueItem,  0)  ==
eSUCCESS )
         {
          if(queueItem.u32Cmd == CMD_OFF_CALIB_START)
          {
           PPAPI_Offcalib_Main(queueItem.u16Attr);
//offline calibration main function, queueItem.u16Attr== enum OFFCALIB_PROCESS_STEP_E
          }
          else if(queueItem.u32Cmd == CMD_VIEWGEN_START)
          {
           PPAPI_Viewgen_Main(queueItem.u16Attr);
//View Generation main function, queueItem.u16Attr== enum VIEWGEN_PROCESS_STEP_E
          }
         }
        }
       }
       else
       {
        //ignore
       }
      }
      else
      {

      }
```

```
    }
   }
  }
 }
}
```

4.6. Emergency Task

EmegencyTask handles all system errors. There are two ways to detect system error.

1) Interrupt of hardware block

If an error happens in a hardware block, the corresponding interrupt will occur. For example, SVM Interrupt will happen if there is continuous overflow/underflow error in SVM block.

2) Diagnosis function

PI5008 provides diagnosis function to monitor internal status. If an abnormal status happens, the corresponding bit of diagnosis register will be set and the diagnosis interrupt will occur. By using diagnosis function, various status can be detected.

- Camera plug-in/plug-out detection
- Video Freeze detection
- Invalid video input format detection

3) Task code

```
PP_VOID vTaskEmergency(PP_VOID *pvData)
{
   int myHandleNum = TASK_EMERGENCY;
   int timeOut = 1000; //msec
   EventBits_t eventWaitBits;
   EventBits_t eventResultBits;
   TaskParam_t *pstParam = (TaskParam_t *)pvData;

int i;

for(;;)
{
 pstParam->u32TaskLoopCnt++;
```

```
if(gHandle[myHandleNum].phEventGroup)
{
 eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
 eventResultBits = OSAL_EVENTGROUP_CMD_WaitBits(gHandle[myHandleNum].phEventGroup, eventWaitBits,
 pdTRUE, pdFALSE, timeOut);

 if ( eventResultBits == 0 )
 {
    continue;
 }
 else
 {
  // process event bit
  if(eventResultBits & (1<<EVENT_EMERGENCY_INIT))
  {
   //Called once when start task.
   //No message Queue.

  }
  if(eventResultBits & (1<<EVENT_EMERGENCY_MSG))
  {
          if(gHandle[myHandleNum].phQueue)
   {
    int queueCnt = 0;
    stQueueItem queueItem;
    if( (queueCnt = OSAL_QUEUE_CMD_GetCount(gHandle[myHandleNum].phQueue)) > 0)
    {
       for(i = 0; i < queueCnt; i++)
     {
      if( OSAL_QUEUE_CMD_Receive(gHandle[myHandleNum].phQueue, &queueItem, 0) == eSUCCESS )
      {
              queueItem.u32Cmd, queueItem.u16Sender, psTaskName[queueItem.u16Sender], queueItem.u16Attr,
queueItem.u32Length);
        if(queueItem.u32Cmd == CMD_EMERGENCY_ALIVE)
        {
```

```
            }

            if(queueItem.u16Attr)
            {
             if(queueItem.u16Attr & (1<<QUEUE_CMDATTR_BIT_REQACK))
             {
              if(queueItem.u32Cmd == CMD_EMERGENCY_ALIVE)
              {
               AppTask_SendCmd(CMD_ACK, TASK_EMERGENCY, queueItem.u16Sender, 0, &pstParam->u32TaskLoopCnt, sizeof(pstParam->u32TaskLoopCnt), 1000);
              }
              else
              {
               AppTask_SendCmd(CMD_ACK, TASK_EMERGENCY, queueItem.u16Sender, 0, NULL, 0, 1000);
              }
             }
            }
            if( (queueItem.u32Length > 0) && (queueItem.pData != NULL) )
            {
             OSAL_free(queueItem.pData);
            }
           }
          }
         }
         else
         {
          }
        }
        else
        {
       }
       }
      }
     }
    }
   return;
```

}

4.6.1. Exception handling

There are 2 ways to handle an exceptional case as follows.

1) Showing error message and system restart if the error happens continuously for a certain period of time.

Error types: TBD

4) Just showing a warning message to notify which error was happened for a certain period of time.

Error types: TBD

4.6.2. Interrupt handler

Each interrupt handler will set HW_ERROR_BIT of ErrEventGroup and fill proper error number into ERROR_Q.

4.7. FWDN Task

FWDNTask performs the system upgrade. If the user selects update menu, an event will be sent to FWDNTask from UITask. At once firmware upgrade is started, task switching will not be done until the upgrade is finished because FWDNTask has the highest priority. Before sending an event, UITask will display a warning message and check conditions such as new binary existence in the SD card. If conditions are satisfied, UITask sends the event to FWDNTask and firmware upgrade will be started.

4.7.1. Event

Name	Bit position	Description
EVENT_FWDN_INIT	Bit 0	Initial event when system is started
EVENT_FWDN_MSG	Bit 1	FWDN message event

Table 37 FWDN Task Event

4.7.2. Queue Command

Name	Description
CMD_FWDN_ALIVE	FWDN alive command
CMD_FWDN_FLASH_UPDATE	FWDN flash update command

Table 38 FWDN Task Queue Command

4.7.3. Task Code

```
for(;;)
{
        pstParam->u32TaskLoopCnt++;

        //LOG_DEBUG("WaitBits(%s)\n", __FUNCTION__);
        if(gHandle[myHandleNum].phEventGroup)
        {
                eventWaitBits = 0x00FFFFFF; //0x00FFFFFF all bit
                eventResultBits = OSAL_EVENTGROUP_CMD_WaitBits
                (gHandle[myHandleNum].phEventGroup, eventWaitBits, pdTRUE, pdFALSE,
                timeOut);

                if ( eventResultBits == 0 )
                {
                    //LOG_DEBUG("timeout\n");
                    continue;
                }
                else
                {
                        // process event bit
                        if(eventResultBits & (1<<EVENT_FWDN_INIT))
                        {/*{{{*/
                                //Called once when start task.
                                //No message Queue.

                        }/*}}}*/
                        if(eventResultBits & (1<<EVENT_FWDN_MSG))
```

```
{/*{{{*/

{/*{{{*/
if(gHandle[myHandleNum].phQueue)
{
int queueCnt = 0;
stQueueItem queueItem;
if( (queueCnt = OSAL_QUEUE_CMD_GetCount (gHandle[myHandleNum].phQueue)) > 0)
{
//LOG_DEBUG("get Queue cnt:%d\n", queueCnt);
for(i = 0; i < queueCnt; i++)
{
if( OSAL_QUEUE_CMD_Receive(gHandle[myHandleNum].phQueue, &queueItem, 0) == eSUCCESS )
{
LOG_DEBUG("%s:Rcv cmd:%08x, sender:%d(%s), attr:%x, length:%d\n", psTaskName[myHandleNum],
queueItem.u32Cmd, queueItem.u16Sender, psTaskName[queueItem.u16Sender], queueItem.u16Attr, queueItem.u32Length);
if(queueItem.u32Cmd == CMD_FWDN_ALIVE)
{
}
else if(queueItem.u32Cmd == CMD_FWDN_FLASH_UPDATE){
result = FlashUpdate(queueItem.pData, queueItem.u32Length);
}
```

```
if(queueItem.u16Attr)
                                                                {
if(queueItem.u16Attr & (1<<QUEUE_CMDATTR_BIT_REQACK))
                                                                    {
if(queueItem.u32Cmd == CMD_FWDN_ALIVE)
{
        AppTask_SendCmd(CMD_ACK, TASK_FWDN, queueItem.u16Sender, 0, &pstParam->u32TaskLoopCnt, sizeof(pstParam->u32TaskLoopCnt), 1000);
}
else
{
    if(result == eSUCCESS)
        AppTask_SendCmd(CMD_ACK, TASK_FWDN, queueItem.u16Sender, 0, NULL, 0, 1000);
    else
        AppTask_SendCmd(CMD_NACK, TASK_FWDN, queueItem.u16Sender, 0, NULL, 0, 1000);
}

AppTask_SendCmd(CMD_DISPLAY_POPUP_ON, TASK_FWDN, TASK_DISPLAY, ePopUp_ShutDown, NULL_PTR, 0, 1000);
```

```
                                                                    }
                                                                }

        if( (queueItem.u32Length > 0) && (queueItem.pData != NULL) )
                                                                {

        OSAL_free(queueItem.pData);
                                                                }
                                                            }
                                                        }
                                                    }
                                                    else
                                                    {
                                                    }
                                                }
                                                else
                                                {
                                                    LOG_CRITICAL("Invalid handle.\n");
                                                }

                                            }/*}}}*/
                                        }/*}}}*/
                                    }
                                }
                            }
```

4.8. Cache Task

Cache Task reads LUTs and 3D car images to show 360-degree surrounding view of a vehicle just after boot up. PI5008 begins to show surround view after reading a certain amount of LUTs and 3D car images. Cache Task reads the remaining LUTs and 3D car images while showing surround view.

1) Loading data

PP_BOOL PPAPP_CACHE_LoadData(PP_S32 s32StartView, PP_S32 s32EndView,

PP_S32 s32BlockingEndView)

s32StartView : The first index of view to be displayed for view rotation.

s32EndView : The last index of view to be displayed for view rotation.

s32BlockingEndView : The last index of view to be read before view rotation.

If we set indexes as below, this function will read LUT data from 0° to 10°. After reading LUT data of 10°, the function is returned and rotation will be stared. Cachetask will read the remaining data(11° ~ 359°) while a view is being rotated.

s32StartView : eVIEWMODE_360_SWING_0

s32EndView : eVIEWMODE_360_SWING_359

s32BlockingEndView : eVIEWMODE_360_SWING_10

.

2) Task code

```
for (;;)
{
    if ( genCache_Status == eCACHETASK_STATUS_DONE )
    {
        OSAL_sleep(300);
    }
    else if ( genCache_Status == eCACHETASK_STATUS_STOP )
    {
#ifdef USE_PP_GUI
        if ( genCache_StopStatus == eCACHETASK_STOP_STATUS_NOMAL )
            AppTask_SendCmd(CMD_UI_CACHE_TASK_DONE, TASK_CACHE, TASK_UI, 0, PP_NULL, 0, 1000);
#endif
            genCache_StopStatus = eCACHETASK_STOP_STATUS_NOMAL;
            genCache_Status = eCACHETASK_STATUS_DONE;
        }
        else if ( gs32Cache_EndView >= 0 && gs32Cache_CurView != gs32Cache_EndView )
        {
            PPAPP_CACHE_LoadView(gs32Cache_CurView+1, gs32Cache_EndView);
        }
        else
        {
            if ( genCache_RunStatus == eCACHETASK_RUN_STATUS_BOTH || genCache_RunStatus ==
eCACHETASK_RUN_STATUS_UIDATA )
```

```
        {
            PPAPP_CACHE_LoadImage(0, eView360_carImage_tire01_start-1, eRSC_MODE_CAR);
            PPAPP_CACHE_LoadImage(0, ePGL_RSCLIST_MAX-1, eRSC_MODE_PGL);
#if defined(USE_GUI_MENU)
            PPAPP_CACHE_LoadImage(0, eUI_RSCLIST_MAX-1, eRSC_MODE_UI);
#else
            PPAPP_CACHE_LoadImage(0, 0, eRSC_MODE_UI);
#endif
#if defined (USE_CAR_WHEEL)
            PPAPP_CACHE_LoadImage(eView360_carImage_wheel_start, eView360_carImage_wheel_end,
eRSC_MODE_CAR);
#endif
#if defined(USE_CAR_DOOR)
            PPAPP_CACHE_LoadImage(eTop2d_carImage_tire01_door_start, eTop2d_carImage_tire01_door_end,
eRSC_MODE_CAR);
#endif
        }

        genCache_Status = eCACHETASK_STATUS_STOP;
    }
}
```

5. UI Structure

Please refer to GUI guide

6. Appendix

6.1. PC Tools for Camera Calibration/View Generation

Various PC Tools provided with PI5008K SDK. These tools will be used to get camera position and angle data, LUT data, Parking Guideline etc.

1) OffLineCalibration Tool
   This tool is used to get camera extrinsic parameters: position and angle
2) ViewGeneration Tool
   This tool is used to get Look-up table, Parking Guideline etc for each view mode
3) PI5008MMap3dCarTool
   This tool is used to get car image for each view mode.

For more details, please refer to the manual of each tool

6.2. On-Board Calibration Time

1. Condition

1) View modes
   Top view 1ea, SD view 4ea, Cylindrical view 2ea
   3D view : 8 view points and swing between points
   Total number of LUT = 45ea (0º,8 º,16 º, ... , 352º)

2) Calibration pattern : Type 7(main) / 8(sub)
3) Memory : NAND Flash / 32bit DDR
   Algorithm runs at both cores – Main and Sub core
4) SDK version : 1.1.0

2. Calibration time

Camera calibration: 18.9 sec
Top view: 506ms
One SD view: 562ms
One 3D view: 824.25ms

System latency: 2.8 sec (task switching & IPC between main & sub core etc.)

Total time: 40.9 sec

Total = camera calibration (18.9s) + top view (506ms) + SD view 4ea (562 ms* 4ea) + 3D views (824.25ms * 20ea) + system delay (2.8s) = 40.939s

Notes) How to make 3D views

Main and sub core are used to make LUT of 3D Views.

Because core0 and core1 runs concurrently, the execution time of sub core(core1) is not included in total time. But there is a system latency due to IPC communication between CPU 0 and 1 because Flash writing is done by CPU 0.

7. Revision History

Version	Date	Description
v0.1	20180608	
V0.5	20181116	
V0.7	20181207	