Crystal Image through
Imaging Innovation

PIXELPLUS

SURROUND VIEW MONITORING SYSTEM

PI5008K BSP User Guide

Rev 0.3

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. BSP Guide

1.1. Introduction

This guide explains PI5008K operation before OS starting.

This guide also explains drivers of primary devices of PI5008K

1.2. System Boot Up

1.2.1. Boot mode

PI5008 supports various boot modes. Boot mode is determined by reading boot mode pin at system reset.

1) Boot mode 0 – SPI NOR Flash
   - Boot from SPI NOR Flash
   - There is a table in address 0 of flash memory. This table includes the location and size of every binary image. Bootloader will be loaded into SRAM of PI5008 based on the Flash table.

2) Boot mode 1 – SDCARD
   - Boot from SD Card. PI5008 Mask ROM supports FAT32 file system. The bootloader will be read from SD Card and loaded into SRAM

3) Boot mode 2 – SPI
   - The bootloader will be loaded via SPI using pre-defined protocol.
   - This mode is used to download the binary image into Flash memory of the target board.

4) Boot mode 4 – SPI NAND Flash
   - Boot from SPI NAND Flash
   - There is a table in address 0 of Flash memory. This table includes location and size of every the binary images. The bootloader will be loaded based on the Flash table.

1.2.2. Booting sequence

This guide explains two boot modes; SPI NOR Flash and SPI NAND Flash boot mode.

1) General Description

PI5008K booting sequence is divided into 2~3 stages, and there are 3 types of bootloaders as follows.

1.1) Bootloader 0

Bootloader 0 is stored on Mask Rom of PI5008K. After reset system, Bootloader 0 runs and load bootloader 1 to SRAM of PI5008K according to boot mode.

1.2) Bootloader 1

Bootloader 1 performs two different functions according to boot mode

SPI NOR Flash

Bootloader 1 initializes DRAM and load the main binary from flash memory

SPI NAND Flash

Bootloader 1 initializes DRAM and loads Bootloader 2 from non FTL area of flash memory.

1.3) Bootloader 2

Bootloader 2 exists only for SPI NAND Flash boot mode.

Bootloader 2 load the main binary from FTL area of flash memory to DRAM

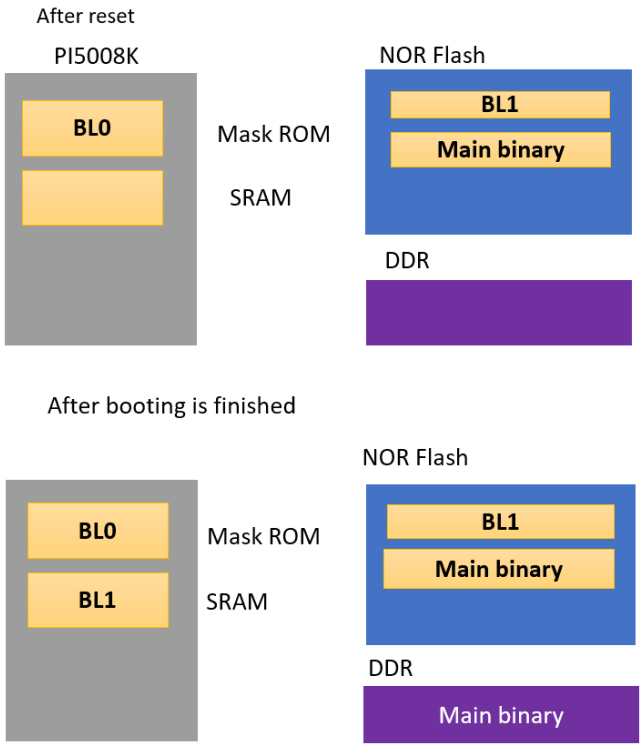


Figure 1 Boot From NOR Flash Memory

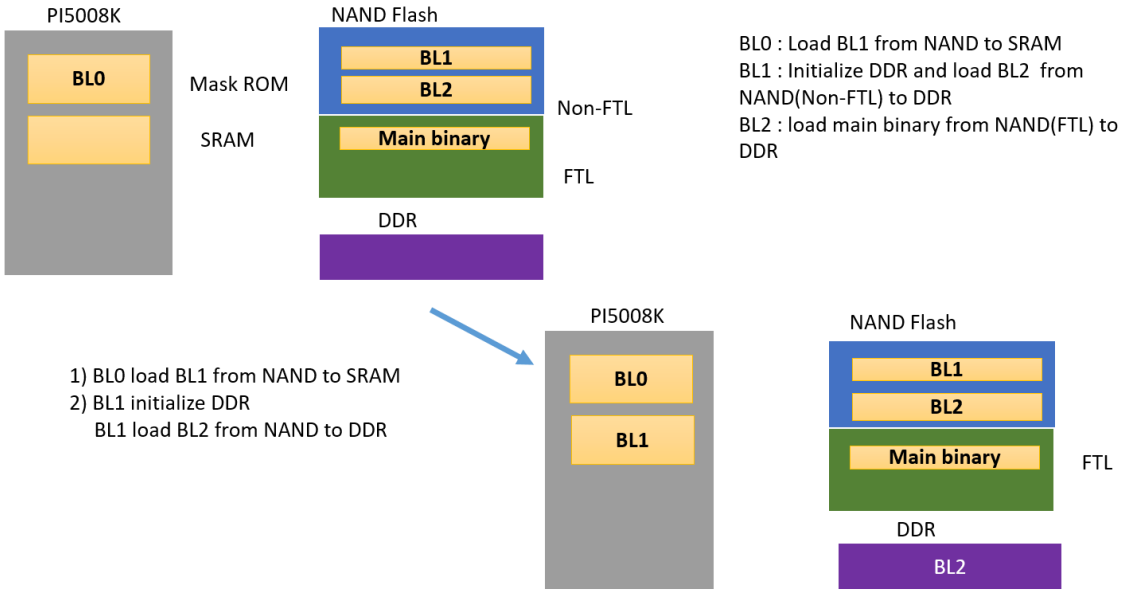


Figure 2 NAND Boot Process_1

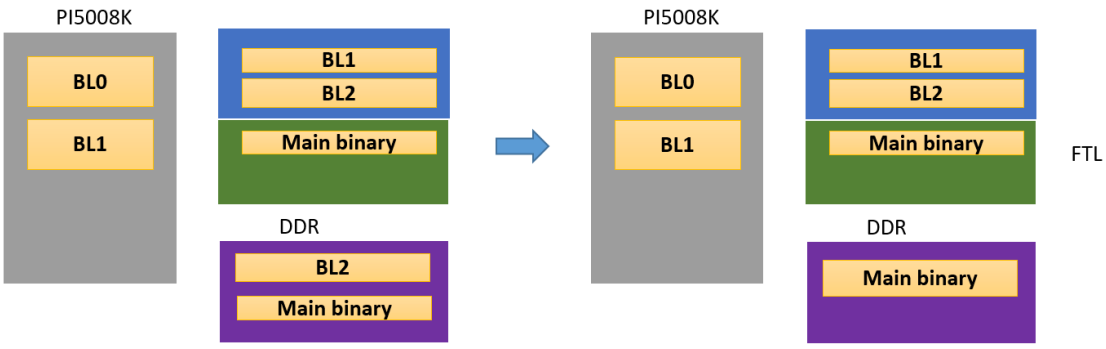


Figure 3 NAND Boot Process_2

1.3. System Initialization

Below diagram shows initialization sequence of PI5008K until OS scheduler is started.

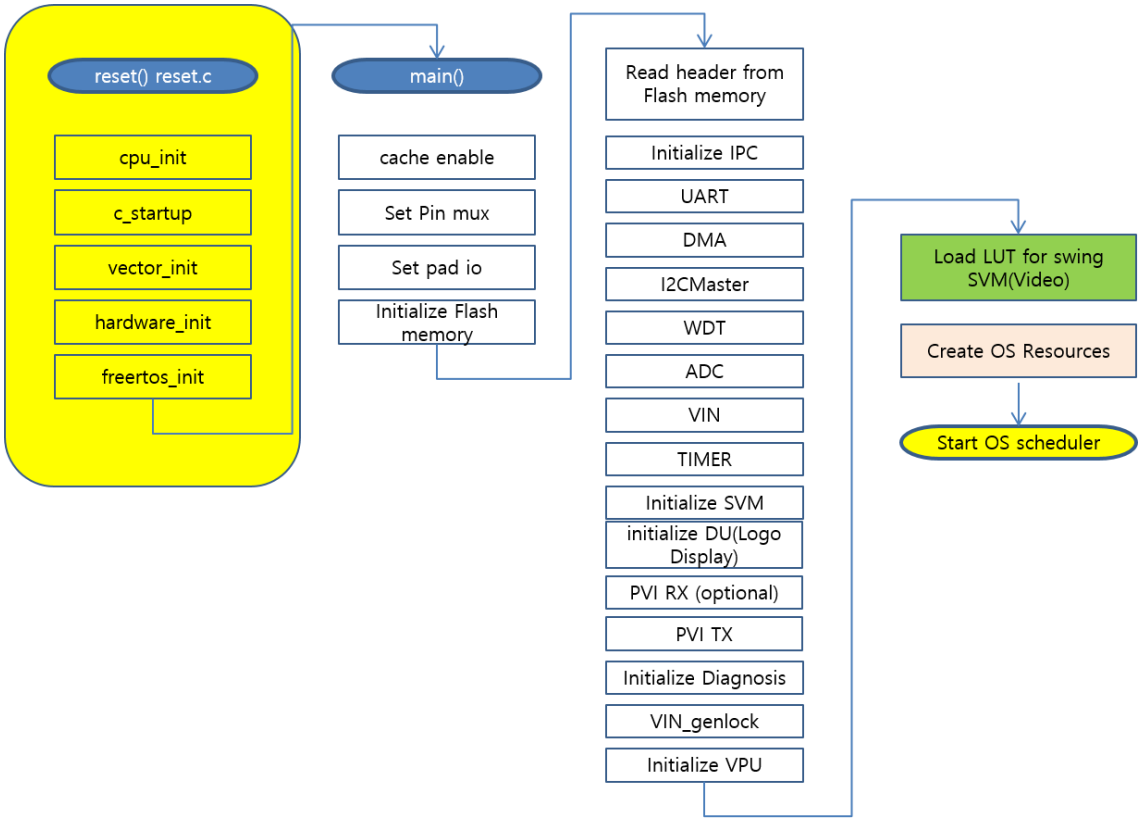


Figure 4 System Initialization Flow Chart

1.3.1. main() function

```
main()
{
  cache_enable(); //cache enable

  init_pinmux(); //pi5008k pin mux

  init_pad_io(); //pi5008k io pin setting

  SYS_PROC_initialize(); // add default irqs & devices //test code?

  AppUARTCon_Initialize(eBAUDRATE_115200, eDATA_BIT_8, eSTOP_BIT_1,
  ePARITY_NONE);
   //uart ch0 controller init

Flash Init
{
 ~~~
}

PPAPI_SVMMEM_Initialize(eVIEWMODE_360_SWING_0, eVIEWMODE_360_SWING_80);
//SVM memory init

init_dram_reserved();
//dram reserved init

PPAPI_IPC_Initialize();
//ipc init

SYSAPI_UART_initialize(eUART_CH_1, eBAUDRATE_115200, eDATA_BIT_8, eSTOP_BIT_1,
ePARITY_NONE);
// uart ch1 controller init

DMA_initialize();
//dma init
```

```
PPDRV_WDT_Initialize(eWDT_TIMEOUT_POW2_19); //wdt init

PPDRV_ADC_Initialize(); //adc init
VINAPI_initialize();//video input initialize

SUB_INTR_Initialize();//irq set

PPAPI_SVM_Initialize();//svm init
PPAPI_SVM_CraeteAllView();// svm create view

PPAPI_DISPLAY_LoadHeader();// load flash data
PPAPI_DISPlAY_Initialize();// display init

#if (VIDEO_IN_TYPE == VIDEO_IN_TYPE_PVI)
PPAPI_PVIRX_Initialize();// pvi rx init
#endif //(VIDEO_IN_TYPE == VIDEO_IN_TYPE_PVI)

PPAPI_PVITX_Initialize();//pvi tx init

PPAPI_DIAG_Initialize();//diagnosis init

PPAPI_VPU_Initialize();//vpu init

PPAPI_FATFS_Initialize();//fatfs init

#ifdef CACHE_VIEW_USE
        PPAPP_CACHE_LoadData(eVIEWMODE_360_SWING_0,
eVIEWMODE_360_SWING_80, eVIEWMODE_360_SWING_0+10); //load 360 swing data from
flash
#endif

AppTask_Init();// creat queue, create event group, create thread

OSAL_start_os();// start task scheduler
}
```

1.4. Bootloader feature

There are several options in Makefile.defs file to configure bootloader features.

1.4.1. DDR Type

User setting	Description
ENABLE_DDRx_INIT = DDR2	DDR2
ENABLE_DDRx_INIT = DDR3	DDR3

1.4.2. DDR Speed

User setting	Description
DDR_SPEED = 400	DDR speed 400MHz (800MT/s)
DDR_SPEED = 500	DDR speed 500MHz (1000MT/s)

1.4.3. DDR Bus Width

User setting	Description
DDR_BUS_BIT = 16	DDR bus width 16 bit
DDR_BUS_BIT = 32	DDR bus width 32 bit

1.4.4. DDR Total Size

User setting	Description
DDR_MEM_SIZE = 64	DDR total size 64MByte
DDR_MEM_SIZE = 128	DDR total size 128MByte
DDR_MEM_SIZE = 256	DDR total size 256MByte
DDR_MEM_SIZE = 512	DDR total size 512MByte
DDR_MEM_SIZE = 1024	DDR total size 1024MByte

1.4.5. Boot Method Selection

Depending on the boot method enable option, the corresponding code is included to output binary. If more than one boot method is enabled, the bootloader binary may exceed valid ram size. We recommend to enable only one boot method. If all boot method is disabled, output binary file name will be PI5008K_Init_ddr3.bin (or PI5008K_Init_ddr2.bin). This file cannot be used for booting, but it can be used to initialize clock and ddr when you run firmware code in dram with J-Tag debugger.

User setting	Description
ENABLE_FLASH_BOOT = y	y : Flash boot(NOR, NAND) is available n : Flash boot is not available.
ENABLE_SDCARD_BOOT = n	y : SDCard boot is available n : flash boot is not available.
ENABLE_EXT_BOOT = n	y : External SPI boot is available n : External SPI boot is not available.

1.4.6. Video Resolution

Video clock can be set according to the input and out resolution.

User setting	Description
DDR_RESOL_INOUT_OPER = HD720pToHD720p	Video input 720p, output 720p
DDR_RESOL_INOUT_OPER = FHD1080pToFHD1080p	Video input 1080p, output 1080p Video input 960p, output 960p
DDR_RESOL_INOUT_OPER = SD720hToSD720h	Video input SD720, output SD720
DDR_RESOL_INOUT_OPER = HD960pToHD720p	Video input 960p, output 720p

1.4.7. SPI Flash Controller Timing

User setting	Description
QSPI_DIV	Flash controller clock divider (0 ~ 15) 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

QSPI_DELAY	Flash controller round trip delay(0 ~ 15) This feature is used to adjust the read timing after each SPI clock. Delay will be set in the unit of SPI core clock.

1.5. Clock Control

PI5008 includes 3 PLLs to generate operating clock; FPLL x 1, MPLL x 2. FPLL generates clock for CPU, bus and SVM operation. MPLL generates clock for video in/out and video processing such as VPU/DU/QUAD. SCU(System Control Unit) block of PI5008K supplies various clocks generated by PLL to each function block.

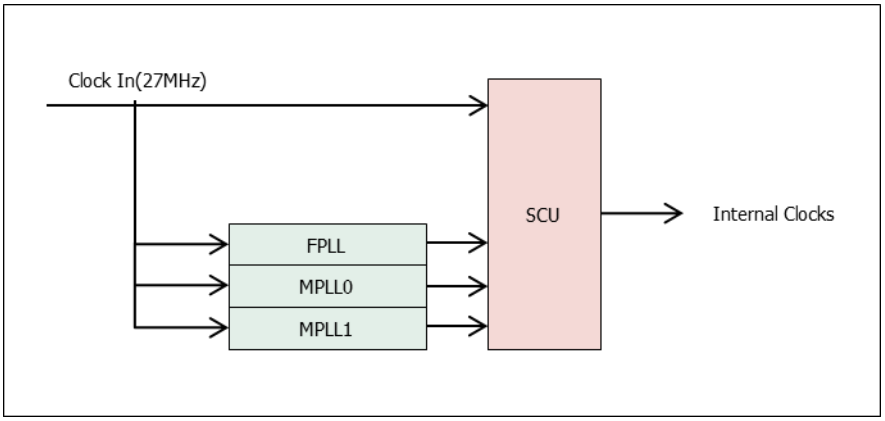


Figure 5 Clock Control

Function block	Freq. (MHz)
FPLL	999
MPLL0	297
MPLL1	216
CPU Core	249.75
CPU AXI	124.875
DDR	499.5
APB	62.4375
TIMER	6.75
UART	27

SVM	148.5
VIDEO IN	Depends on input
DU	Depends on output

Table 1 Clock Control

1.6. Firmware Download

There are two ways to download firmware to flash memory; JTAG and PC Tool

1.5.1 Download by JTAG

PI5008K SDK provides several files to download firmware using JTAG as follows,

Binary image file : flash_image.bin

Flash burning program : flash_burner.adx

Script file : run_flash

asicinit file : PI5008K_Init.bin

This execution file initialize DDR/PLL before download firmware

Procedure to download firmware is as follows

1) Copy files to same folder.

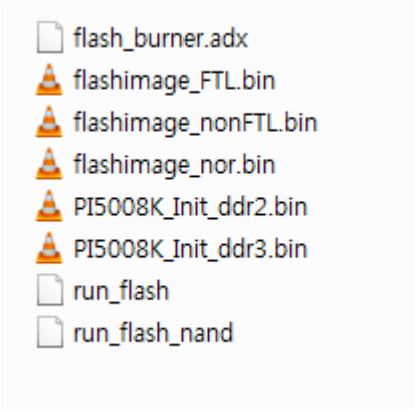


Figure 6 Files For Firmware Download

2) Edit run_flash script file

Below items need to be modified

Flash memory type, binary image file name, set file name for asicinit

A. NOR flash memory

```
#################################################################
###             burner program parameters                     ###
#################################################################
set pagination off
target remote:1111
# burner program version - do not modify
set $version=0x2
# burner program information addr - do not modify
set $info_addr=0x20000000

set $buffer_addr=0x20120000
set $buffer_size=0x1000000
set $read_buffer_addr=$buffer_addr + $buffer_size
set $verify_flag=1
set $flash_addr=0x0
set $spi_write_div=0
set $spi_read_div=0
set $spi_read_dly=3
set $break_addr=0x10124

# flash type
#   0: nor flash
#   1: nand
#   2: nand ftl
set $flash_type=0

asicinit "PI5008K_Init_ddr2.bin" $break_addr
flash_burn "flashimage_nor.bin"
```

B. NAND flash memory

In case of NAND flash memory, run_flash_nand script is used. Burning process is executed twice for NAND flash non-FTL area and FTL area.

```
#################################################################
###             burner program parameters                     ###
#################################################################
set pagination off
target remote:1111
# burner program version - do not modify
set $version=0x2
# burner program information addr - do not modify
set $info_addr=0x20000000

set $buffer_addr=0x20120000
set $buffer_size=0x1000000
set $read_buffer_addr=$buffer_addr + $buffer_size
set $verify_flag=1
set $flash_addr=0x0
set $spi_write_div=0
set $spi_read_div=0
set $spi_read_dly=3
set $break_addr=0x10124

# flash type
#   0: nor flash
#   1: nand
#   2: nand ftl
asicinit "PI5008K_Init_ddr3.bin" $break_addr

set $flash_type=1
flash_burn "flashimage_nonFTL.bin"

set $flash_type=2
flash_burn "flashimage_FTL.bin"
```

3) Run cygwin-andes.bat

4) Run flash burner

(nds32le-elf-gdb flash_burner.adx)

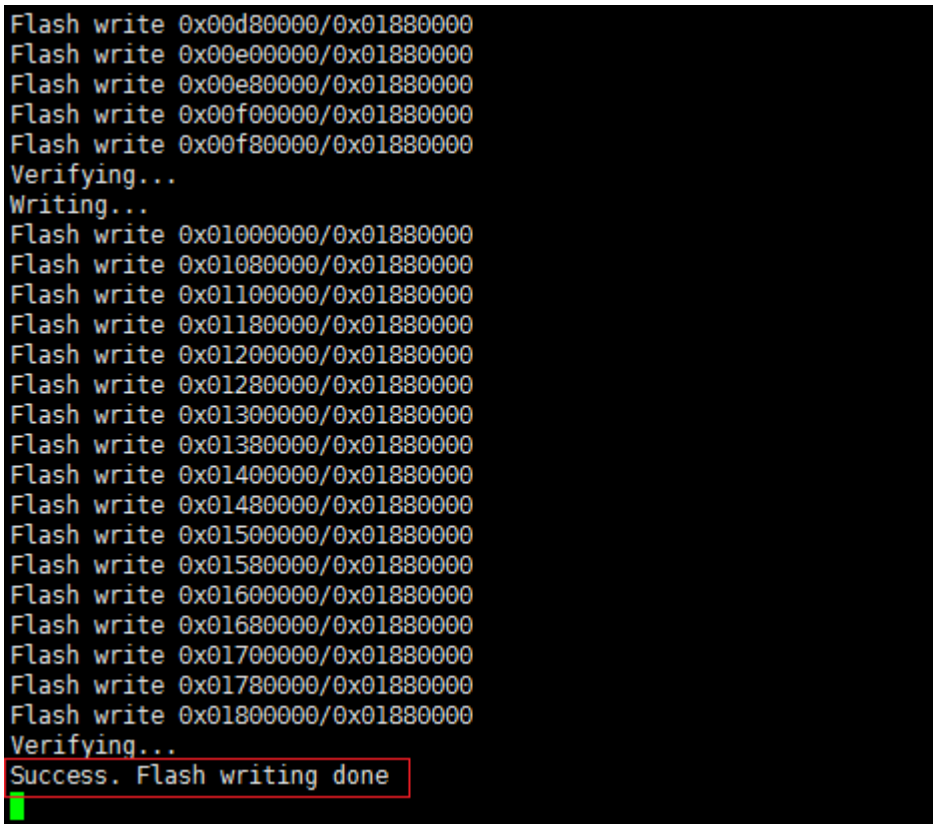
5) Run script

In case of NAND flash memory, execute run_flash_nand script.

```
smoh@SMOH /cygdrive/c/Users/smoh.PIXEL/Desktop/jtag_asic
$ nds32le-elf-gdb.exe flash_burner.adx
GNU gdb (2017-12-02_nds32le-elf) 7.7.0.20140207-cvs
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=nds32le-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
[info] Loading .Andesgdbinit.
[info] .Andesgdbinit loaded.
Reading symbols from flash_burner.adx...done.
(gdb) source run_flash
0x00000604 in ?? ()
"Target reset ..."
0x00000604 in ?? ()
edm_passcode =>Current privilege level: 0
nds32.cpu0: Data cache disabled
nds32.cpu0: Instruction cache disabled
Restoring binary file PI5008K_Init.bin into memory (0x10000 to 0x10af0)
$1 = (void (*)()) 0x10000
Breakpoint 1 at 0x10124

Breakpoint 1, 0x00010124 in ?? ()
nds32.cpu0: Data cache disabled
nds32.cpu0: Instruction cache disabled
Loading section .nds32_init, size 0x290 lma 0x23000000
Loading section .text, size 0x2d27c lma 0x23000290
Loading section .rodata, size 0x2560 lma 0x2302d50c
Loading section .data, size 0x240 lma 0x2302fa80
Start address 0x23005958, load size 195756
Transfer rate: 1131 KB/sec, 3838 bytes/write.
Breakpoint 2 at 0x2300885e: file ../source/applications/Apps/brun_flash.c, line 216.
"change memory access mode bus"nds32.cpu0: Data cache disabled
nds32.cpu0: Instruction cache disabled
Restoring binary file flash_image.bin into memory (0x20120000 to 0x21120000)
"change memory access mode cpu"
```

6. Check download process through serial terminal



1.5.2 Download by PC Tool

PC tool can be used to download 5008K firmware. PC is connected to PI5008 board via USB to SPI bridge board. To use PC tool, PI5008 has to be booted up in boot mode 2. For more details, please refer to PI5008K_Merge&DownloadToolUserGuide.

1.7. Basic Device Drivers

1.7.1. UART

1.7.1.1. PPDRV_UART_SWreset

Prototype	PP_VOID PPDRV_UART_SWreset (PP_VOID);
Description	Reset UART block
Argument	none
Return value	none
Example	

1.7.1.2. PPDRV_UART_Init

Prototype	PP_VOID PPDRV_UART_Init (PP_UART_CHANNEL_E IN ch, PP_UART_BAUDRATE_E IN baudrate, PP_UART_DATABIT_E IN databit, PP_UART_STOP_BIT_E IN stopbit, PP_UART_PARITY_E IN parity);
Description	Initialize UART block
Argument	ch : uart channel baudrate: uart baud rate databit: uart data bits stopbit: uart stop bits parity: ;uart parity bit
Return value	none
Example	

1.7.1.3. PPDRV_UART_InitInterrupt

Prototype	PP_VOID PPDRV_UART_InitInterrupt (PP_UART_CHANNEL_E IN ch, PP_UartFifoOpt_S IN opt);
Description	Initialize UART interrupt
Argument	ch : uart channel opt: interrupt option
Return value	none

Example	{
	PP_S32 ch = 0;
	PP_UartFifoOpt uartOpt;
	
	uartOpt.rxEnable = FLAG_SET; // rx interrupt enable
	uartOpt.txEnable = FLAG_CLEAR; // tx interrupt disable
	uartOpt.rxLevel = eRX_LEVEL_0; // rx fifo trigger level
	uartOpt.txLevel = eTX_LEVEL_0; // tx fifo trigger level
	
	PPDRV_UART_InitInterrupt(ch, uartOpt);
	}

1.7.1.4. PPDRV_UART_OutByte

Prototype	PP_VOID PPDRV_UART_OutByte (PP_U8 IN ch, PP_U8 IN c);
Description	Transmit 1 byte data
Argument	ch : uart channel c : Byte data to be transmitted
Return value	none
Example	

1.7.1.5. PPDRV_UART_InByte

Prototype	PP_U8 PPDRV_UART_InByte (PP_U8 IN ch);
Description	Receive 1 byte data
Argument	ch : uart channel
Return value	0 : There is no received data. Else: Number of received byte data
Example	

1.7.1.6. PPDRV_UART_GetRxReady

Prototype	PP_U8 PPDRV_UART_GetRxReady (PP_U8 IN ch);
Description	Check there is a received data
Argument	ch : uart channel
Return value	0 : No received data. 1 : There is(are) received data.
Example	

1.7.1.7. PPDRV_UART_GetTxEmpty

Prototype	PP_U8 PPDRV_UART_GetTxEmpty (PP_U8 IN ch);
Description	Check whether tx buffer is empty
Argument	ch : uart channel
Return value	0 : Tx buffer is not empty. 1 : Tx buffer is empty.
Example	

1.7.2. SPI

1.7.2.1. PPDRV_SPI_Initialize

Prototype	PP_RESULT_E PPDRV_SPI_Initialize(PP_S32 IN s32Ch, PP_U32 IN u32IsSlave, PP_U32 IN u32Freq, PP_U32 IN u32ClkMode, PP_U32 IN u32WordLen);
Description	Initialize SPI block
Argument	s32Ch : spi channel u32IsSlave : Select master or slave mode u32Freq : output clock frequency (Hz) u32ClkMode: spi clock & phase mode (0 ~ 3) u32WordLen: Size of data unit(bits)
Return value	eSUCCESS : Success else : Fail to initialize

Example	{
	PP_S32 s32Ch = 0;
	PP_U32 u32IsSlave = 0; // master
	PP_U32 u32Freq = 1000000; // device clock 1MHz
	PP_U32 u32ClkMode = 0; // spi mode 0
	PP_U32 u32WordLen = 8; // 8 bit data
	
	PPDRV_SPI_Initialize(s32Ch, u32IsSlave, u32Freq, u32ClkMode, u32WordLen);
	}

1.7.2.2. PPDRV_SPI_IRQEnable

Prototype	PP_VOID PPDRV_SPI_IRQEnable(PP_S32 s32Ch, PP_U32 u32Irq);
Description	Interrupt enable
Argument	s32Ch : spi channel u32Irq : Combination of interrupt flag which will be enabled
Return value	none
Example	

1.7.2.3. PPDRV_SPI_IRQClear

Prototype	PP_VOID PPDRV_SPI_IRQClear(PP_S32 s32Ch, PP_U32 u32Irq);
Description	Interrupt disable
Argument	s32Ch : spi channel u32Irq : Combination of interrupt flag which will be disabled
Return value	none
Example	

1.7.2.4. PPDRV_SPI_GetFreq

Prototype	PP_U32 PPDRV_SPI_GetFreq(PP_S32 s32Ch);
Description	Read clock frequency which is set.
Argument	s32Ch : spi channel
Return value	clock frequency (Hz) for device
Example	

1.7.2.5. PPDRV_SPI_SetISR

Prototype	PP_VOID PPDRV_SPI_SetISR(PP_S32 IN s32Ch, SPI_ISR_CALLBACK IN cbISR);
Description	Register callback function which is called by Interrupt service routine. Registered callback function will be executed whenever interrupt is happened.
Argument	s32Ch : spi channel cbISR : Pointer to Callback function
Return value	none

Example	static PP_VOID callback(PP_U32 u32Status)
	{
	...
	}
	
	{
	PP_S32 s32Ch = 0;

	PPDRV_SPI_SetISR(s32Ch, callback); }

1.7.2.6. PPDRV_SPI_Tx

Prototype	PP_RESULT_E PPDRV_SPI_Tx(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U32 IN u32Size);
Description	Transmit data
Argument	s32Ch : spi channel pu8DOut : Pointer to data buffer to be transmitted u32Size : Size of data to be transmitted (max 512)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.7. PPDRV_SPI_Rx

Prototype	PP_RESULT_E PPDRV_SPI_Rx(PP_S32 IN s32Ch, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size);
Description	Receive data
Argument	s32Ch : spi channel pu8DIn : Pointer to receive buffer u32Size : Size of data received (max 512)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.8. PPDRV_SPI_TxRx

Prototype	PP_RESULT_E PPDRV_SPI_TxRx(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size);
Description	Transmit and receive data simultaneously
Argument	s32Ch : spi channel pu8DOut : pointer to tx data buffer pu8DIn : pointer to rx data buffer

	u32Size : Size of data (max. 512)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.9. PPDRV_SPI_TxDMA

Prototype	PP_RESULT_E PPDRV_SPI_TxDMA(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U32 IN u32Size, PP_S32 IN s32DMACh, PP_U32 IN u32Timeout);
Description	Transmit data using DMA
Argument	s32Ch : spi channel pu8DOut : pointer to tx data buffer u32Size : Size of data (max. 512) s32DMACh : dma channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.10. PPDRV_SPI_RxDMA

Prototype	PP_RESULT_E PPDRV_SPI_RxDMA(PP_S32 IN s32Ch, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size, PP_S32 IN s32DMACh, PP_U32 IN u32Timeout);
Description	Receive data using DMA
Argument	s32Ch : spi channel pu8DIn : pointer to rx data buffer u32Size : Size of data (max. 512) s32DMACh : dma channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.11. PPDRV_SPI_TxRxDMA

Prototype	PP_RESULT_E PPDRV_SPI_TxRxDMA(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size, PP_S32 IN s32DMATxCh, PP_S32 IN s32DMARxCh, PP_U32 IN u32Timeout);
Description	Transmit and receive data using DMA
Argument	s32Ch : spi channel pu8DOut : pointer to tx data buffer pu8DIn : pointer to rx data buffer u32Size : Size of data (max. 512) s32DMATxCh : dma tx channel s32DMARxCh : dma rx channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.12. PPDRV_SPI_DMAWait

Prototype	PP_RESULT_E PPDRV_SPI_DMAWait(PP_S32 IN s32Ch, PP_S32 IN s32DMACh, PP_U32 IN u32Timeout);
Description	This function waits until DMA is finished. **[Caution]** If a SPI DMA related function is called with a timeout value 0, this wait function must be executed.
Argument	s32Ch : spi channel s32DMACh : dma channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.13. PPDRV_SPI_TxRx_DMAWait

Prototype	PP_RESULT_E PPDRV_SPI_TxRx_DMAWait(PP_S32 IN s32Ch,

	PP_S32 IN s32DMATxCh, PP_S32 IN s32DMARxCh, PP_U32 IN u32Timeout);
Description	This function is for "PPDRV_SPI_TxRxDMA" or "PPDRV_SPI_TxRxSlaveDMA". Other features are the same as "PPDRV_SPI_DMAWait" function
Argument	s32Ch : spi channel s32DMATxCh : dma tx channel s32DMARxCh : dma rx channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.14. PPDRV_SPI_GPIOCSEnable

Prototype	PP_VOID PPDRV_SPI_GPIOCSEnable(PP_S32 IN s32Ch, PP_U8 u8Enable);
Description	Control SPI chip select pin using GPIO. Basically chip select pin is controlled automatically by hardware. This function is used to control chip select signal directly..
Argument	s32Ch : spi channel u8Enable : enable flag
Return value	none

Example	{
	PP_U8 u8Cmd[4];
	PP_U8 u8Buf[4];
	
	PPDRV_SPI_Initialize(eSPI_CHANNEL_FLASH, 0, 1000000, 0, 8);
	PPDRV_SPI_GPIOCSEnable(eSPI_CHANNEL_FLASH, 1);
	
	u8Cmd[0] = 0x9f; // read id command
	u8Cmd[1] = 0;
	u8Cmd[2] = 0;
	u8Cmd[3] = 0;
	
	PPDRV_SPI_CSActivate(eSPI_CHANNEL_FLASH);

	PPDRV_SPI_TxRx(eSPI_CHANNEL_FLASH, u8Cmd, u8Buf, 4); PPDRV_SPI_CSDeActivate(eSPI_CHANNEL_FLASH); }

1.7.2.15. PPDRV_SPI_CSActivate

Prototype	PP_VOID PPDRV_SPI_CSActivate(PP_S32 IN s32Ch);
Description	Make SPI chip select signal low state
Argument	s32Ch : spi channel
Return value	None
Example	

1.7.2.16. PPDRV_SPI_CSDeActivate

Prototype	PP_VOID PPDRV_SPI_CSDeActivate(PP_S32 IN s32Ch);
Description	Make SPI chip select 신호 high state
Argument	s32Ch : spi channel
Return value	None
Example	

1.7.2.17. PPDRV_SPI_CSGetLevel

Prototype	PP_S32 PPDRV_SPI_CSGetLevel(PP_S32 IN s32Ch);
Description	Read current state of chip select signal.
Argument	s32Ch : spi channel
Return value	0: low 1: high
Example	

1.7.2.18. PPDRV_SPI_Wait

Prototype	PP_VOID PPDRV_SPI_Wait(PP_S32 IN s32Ch);
Description	This function waits while the spi state is in progress.
Argument	s32Ch : spi channel

Return value	none
Example	

1.7.2.19. PPDRV_SPI_TxSlave

Prototype	PP_RESULT_E PPDRV_SPI_TxSlave(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U32 IN u32Size);
Description	Transmit data in slave mode. SPI has to be initialized to slave mode in advance.
Argument	s32Ch : spi channel pu8DOut : Pointer to tx data u32Size : Size of data to be transmitted
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.20. PPDRV_SPI_RxSlave

Prototype	PP_RESULT_E PPDRV_SPI_RxSlave(PP_S32 IN s32Ch, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size);
Description	Receive data in slave mode. SPI has to be initialized to slave mode in advance.
Argument	s32Ch : spi channel pu8DIn : Pointer to rx data buffer u32Size : Size of data to be received
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.21. PPDRV_SPI_TxSlaveDMA

Prototype	PP_RESULT_E PPDRV_SPI_TxSlaveDMA(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U32 IN u32Size, PP_S32 IN s32DMACh, PP_U32 IN u32Timeout);
Description	Transmit data using DMA in slave mode. SPI has to be initialized to slave mode in advance.

Argument	s32Ch : spi channel pu8DOut : Pointer to tx data u32Size : Size of data to be transmitted s32DMACh : dma channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.22. PPDRV_SPI_RxSlaveDMA

Prototype	PP_RESULT_E PPDRV_SPI_RxSlaveDMA(PP_S32 IN s32Ch, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size, PP_S32 IN s32DMACh, PP_U32 IN u32Timeout);
Description	Receive data using DMA in slave mode. SPI has to be initialized to slave mode in advance.
Argument	s32Ch : spi channel pu8DIn : Pointer to rx data u32Size : Size of data to be transmitted s32DMACh : dma channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.23. PPDRV_SPI_TxRxSlaveDMA

Prototype	PP_RESULT_E PPDRV_SPI_TxRxSlaveDMA(PP_S32 IN s32Ch, PP_U8 IN *pu8DOut, PP_U8 OUT *pu8DIn, PP_U32 IN u32Size, PP_S32 IN s32DMATxCh, PP_S32 IN s32DMARxCh, PP_U32 IN u32Timeout);
Description	Transmit and receive data using DMA in slave mode. SPI has to be initialized to slave mode in advance.
Argument	s32Ch : spi channel pu8DOut : Pointer to tx data pu8DIn : Pointer to rx data

	u32Size : Size of data to be transmitted s32DMATxCh : dma tx channel s32DMARxCh : dma rx channel u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.24. PPDRV_SPI_SetTxSlaveIntr

Prototype	PP_RESULT_E PPDRV_SPI_SetTxSlaveIntr(PP_S32 IN s32Ch)
Description	Start to transmit data in interrupt enable mode. Interrupt handler function should be implemented to transmit data and interrupt handler function should be registered to spi driver using ″PPDRV_SPI_SetISR″ function before calling this function. SPI has to be initialized to slave mode in advance.
Argument	s32Ch : spi channel
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.25. PPDRV_SPI_SetRxSlaveIntr

Prototype	PP_RESULT_E PPDRV_SPI_SetRxSlaveIntr(PP_S32 IN s32Ch)
Description	Start to receive data in interrupt enable mode. Interrupt handler function should be implemented to receive data and interrupt handler function should be registered to spi driver using ″PPDRV_SPI_SetISR″ function before calling this function. SPI has to be initialized to slave mode in advance.
Argument	s32Ch : spi channel
Return value	eSUCCESS : Success else : Fail
Example	

1.7.2.26. PPDRV_SPI_SetSlaveReady

Prototype	PP_VOID PPDRV_SPI_SetSlaveReady(PP_S32 IN s32Ch);
Description	This function sets the slave ready bit in spi controller register. SPI master can read slave ready using status-checking command. When SPI transaction other than status-checking command ends, slave ready bit will be cleared to 0. For more details about status-checking command, please refer to spi communication protocol section in PI5008_SDK_UserGuide document.
Argument	s32Ch : spi channel
Return value	none
Example	

1.7.2.27. PPDRV_SPI_SetSlaveStatus

Prototype	PP_VOID PPDRV_SPI_SetSlaveStatus(PP_S32 IN s32Ch, PP_U16 IN u16Status);
Description	This function sets the slave status in spi controller register. SPI master can read slave status using status-checking command. For more details about status-checking command, please refer to spi communication protocol section in PI5008_SDK_UserGuide document.
Argument	s32Ch : spi channel u16Status : spi slave status
Return value	none
Example	

1.7.2.28. PPDRV_SPI_GetSlaveStatus

Prototype	PP_U16 PPDRV_SPI_GetSlaveStatus(PP_S32 IN s32Ch);
Description	This function returns the slave status which is set by "PPDRV_SPI_SetSlaveStatus" function.
Argument	s32Ch : spi channel
Return value	Spi slave status
Example	

1.7.3. I2C

1.7.3.1. PPDRV_I2C_Initialize

Prototype	PP_VOID PPDRV_I2C_Initialize(PP_VOID);
Description	Initialize I2c block
Argument	None
Return value	none
Example	

1.7.3.2. PPDRV_I2C_Setup

Prototype	PP_VOID PPDRV_I2C_Setup(PP_S32 IN s32Ch, PP_U32 IN u32DevAddr, PP_I2C_SPEED_E IN enSpeed, PP_I2C_MODE_E IN enMode, PP_I2C_ADDRBIT_E IN enAddrBit);
Description	Setting for I2c communication
Argument	s32Ch : i2c channel u32DevAddr : For master mode, slave device address. For slave mode, own address enSpeed : i2c speed (0: 100kb/s, 1: 400kb/s) enMode : master, slave mode (0: master, 1: slave) enAddrBit : i2c addressing mode (0: 7bit, 1: 10bit)
Return value	none

Example	{
	
	PP_S32 s32Ch = 0;
	PP_U32 u32DevAddr = 0x48;
	PP_I2C_SPEED_E enSpeed = eI2C_SPEED_NORMAL;
	// 100kb/s
	PP_I2C_MODE_E enMode = eI2C_MODE_MASTER; // master
	PP_I2C_ADDRBIT_E enAddr = eI2C_ADDRESS_7BIT; // 7bit address
	PP_U8 u8Buf[4];

	PP_U32 u32Timeout = 1000;
	// timeout 1000 ms
	PP_RESULT_E enRet;
	
	PPDRV_I2C_Initialize();
	PPDRV_I2C_Setup(s32Ch, u32DevAddr, enSpeed, enMode, enAddr);
	
	PPDRV_I2C_Enable(s32Ch, 1);
	enRet = PPDRV_I2C_Write(s32Ch, u8Buf, 4, u32Timeout);
	if(enRet == eSUCCESS){
	// success
	}else if(enREt == eERROR_TIMEOUT){
	// timeout
	}else{
	// fail
	
	}

1.7.3.3. PPDRV_I2C_Enable

Prototype	PP_VOID PPDRV_I2C_Enable(PP_S32 IN s32Ch, PP_U32 IN u32Enable);
Description	I2c enable
Argument	s32Ch : i2c channel u32Enable : enable (0: disable, 1: enable)
Return value	none
Example	

1.7.3.4. PPDRV_I2C_SetISR

Prototype	PP_VOID PPDRV_I2C_SetISR(PP_S32 IN s32Ch, I2C_ISR_CALLBACK IN cbISR);
Description	Register callback function which is called by Interrupt service routine. Registered callback function will be executed whenever interrupt is happened..

Argument	s32Ch : i2c channel cbISR : Pointer to callback function
Return value	none
Example	

1.7.3.5. PPDRV_I2C_Write

Prototype	PP_RESULT_E PPDRV_I2C_Write(PP_S32 IN s32Ch, PP_U8 IN *pu8Data, PP_U8 IN u8DataSize, PP_U32 IN u32Timeout);
Description	Transmit data
Argument	s32Ch : i2c channel pu8Data : Pointer to tx data buffer u8DataSize : Number of tx data u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.3.6. PPDRV_I2C_Read

Prototype	PP_RESULT_E PPDRV_I2C_Read(PP_S32 IN s32Ch, PP_U8 OUT *pu8Data, PP_U8 IN u8DataSize, PP_U32 IN u32Timeout);
Description	Receive Data
Argument	s32Ch : i2c channel pu8Data : Pointer to rx data buffer u8DataSize : Size of rx data size u32Timeout : timeout (ms)
Return value	eSUCCESS : Success else : Fail
Example	

1.7.3.7. PPDRV_I2C_GetDone

Prototype	PP_U32 PPDRV_I2C_GetDone(PP_S32 IN s32Ch, PP_RESULT_E OUT *penResult);
Description	Decide whether to finish i2c communication I2c. If rx/tx sets

	timeout value is 0, it is possible to check whether the communication is finished and the result of communication.
Argument	s32Ch : i2c channel enResult : Success(eSuccess) or fail(eERROR_FAILURE)
Return value	Return whether i2c communication is finished regardless of success or fail 0: Not yet finished. 1: Finished
Example	

1.7.3.8. PPDRV_I2C_SlaveSetCallback

Prototype	PP_VOID PPDRV_I2C_SlaveSetCallback(PP_S32 IN s32Ch, I2C_SLAVE_READ IN cbRecv, I2C_SLAVE_WRITE IN cbSend);
Description	Register callback function to be called when tx or rx is requested. Registered rx function is called to receive data. If tx is requested, send function is called.
Argument	s32Ch : i2c channel cbRecv : Pointer to receive callback function cbSend : Pointer to send callback function
Return value	none

Example	PP_VOID i2c_slave_onreceive(PP_S32 s32Ch, PP_U8 *pu8Buf, PP_U32 u32Size)
	{
	printf("i2c slave received. size: %d\n", u32Size);
	}
	
	PP_U32 i2c_slave_onsend(PP_S32 s32Ch, PP_U8 *pu8Buf, PP_U32 u32MaxSize)
	{
	PP_S32 i;
	
	for(i=0;i<SendSize && i<u32MaxSize;i++){
	buf[i] = SendBuf[i];
	}

	return i; // return size
	}
	
	{
	PP_VOID PPDRV_I2C_SlaveSetCallback(0, i2c_slave_onreceive, i2c_slave_onsend);
	}

1.7.3.9. PPDRV_I2C_Reset

Prototype	PP_VOID PPDRV_I2C_Reset(PP_S32 IN s32Ch);
Description	I2c block reset. Stop current communication and clear fifo
Argument	s32Ch : i2c channel
Return value	none
Example	

1.7.4. I2S

PI5008 I2S block is used only for PCM data output. When data size in output buffer is smaller than under-run margin, interrupt is happened and data will be filled.

1.7.4.1. Data Type

2) PP_DAI_FORMAT_E

A. Syntax

```
typedef enum ppDAI_FORMAT_E
{
        eFORMAT_16BIT = 0,
        eFORMAT_32BIT = 2,
}PP_DAI_FORMAT_E;
```

B. Members

eFORMAT_16BIT	Audio data format 16 bit
eFORMAT_32BIT	Audio data format 32 bit

3) PP_DAI_MLF_E

A. [syntax]

```
typedef enum ppDAI_MLF_E
{
        eMLF_MSB_FIRST = 0,
        eMLF_LSB_FIRST,
}PP_DAI_MLF_E;
```

B. [member]

eMLF_MSB_FIRST	MSB first
eMLF_LSB_FIRST	LSB first

4) PP_DAI_BUFF_SIZE_E

A. [syntax]

```
typedef enum ppDAI_BUFF_SIZE_E
{
        eBUFF_SIZE_4K = 0,
        eBUFF_SIZE_8K,
        eBUFF_SIZE_16K,
        eBUFF_SIZE_32K,
}PP_DAI_BUFF_SIZE_E;
```

B. [member]

eBUFF_SIZE_4K	PCM buffer size 4 KB
eBUFF_SIZE_8K	PCM buffer size 8 KB
eBUFF_SIZE_16K	PCM buffer size 16 KB
eBUFF_SIZE_32K	PCM buffer size 32 KB

5) PP_DAI_RFS_E

A. [syntax]

```
typedef enum ppDAI_RFS_E
{
        eRFS_256FS = 0,
        eRFS_384FS,
        eRFS_512FS,
        eRFS_768FS,
```

```
}PP_DAI_RFS_E;
```

B. [member]

eRFS_256FS	FS = mclk / 256
eRFS_384FS	FS = mclk / 384
eRFS_512FS	FS = mclk / 512
eRFS_768FS	FS = mclk / 768

6) PP_I2S_INIT_S

A. [syntax]

```
typedef struct ppI2S_INIT_S{
        PP_DAI_FORMAT_E enDataFormat;
        PP_DAI_MLF_E enMLBFirst;
        PP_DAI_BUFF_SIZE_E enBufferSize;
        PP_DAI_RFS_E enSampleFreqRatio;
        PP_U32 u32UnderMargin;
        PP_U32 u32IsMaster;
}PP_I2S_INIT_S;
```

B. [member]

enDataFormat	Audio data format
enMLBFirst	MSB/LSB first
enBufferSize	PCM buffer size
enSampleFreqRatio	Sampling frequency
u32UnderMargin	Tx buffer under-run margin size (word align)
u32IsMaster	0: slave, 1: master

1.7.4.2. PPDRV_I2S_Initialize

Prototype	PP_RESULT_E PPDRV_I2S_Initialize(PP_I2S_INIT_S IN *pstInit);
Description	I2S initialize
Argument	pstInit : Pointer to initial variable(?)
Return value	eSUCCESS : Success else : Fail
Example	static PP_VOID audio_callback(PP_VOID) {

	if(PPDRV_I2S_GetWriteBuffAvailSize() >= write_unit){
	PPDRV_I2S_WriteBuf(&pAudioBuf[writep], write_unit);
	writep += write_unit;
	}
	}
	
	{
	PP_I2S_INIT_S stInit;
	
	stInit.enDataFormat = eFORMAT_16BIT;
	stInit.enMLBFirst = eMLF_MSB_FIRST;
	stInit.enBufferSize = eBUFF_SIZE_32K;
	stInit.enSampleFreqRatio = eRFS_512FS;
	// mclk: 4.096M, fs: 8K
	stInit.u32UnderMargin = (16*1024);
	// under-run margin = buffersize / 2
	stInit.u32IsMaster = 0;
	// slave
	
	PPDRV_I2S_Initialize(&stInit);
	PPDRV_I2S_SetISR(audio_callback);
	I2S_WriteBuf(pAudioBuf, stInit.u32UnderMargin);
	PPDRV_I2S_WriteEnable(1);
	}

1.7.4.3. PPDRV_I2S_SetISR

Prototype	PP_VOID PPDRV_I2S_SetISR(I2S_ISR_CALLBACK IN cbISR);
Description	Register callback function which is called by Interrupt service routine. Registered callback function will be executed whenever interrupt is happened.
Argument	cbISR : cbISR : Pointer to callback function
Return value	none
Example	

1.7.4.4. PPDRV_I2S_WriteEnable

Prototype	PP_VOID PPDRV_I2S_WriteEnable(PP_S32 IN s32Enable);
Description	Start I2S audio data output
Argument	s32Enable : enable(1) or disable(0)
Return value	None
Example	

1.7.4.5. PPDRV_I2S_ResetBufPtr

Prototype	PP_VOID PPDRV_I2S_ResetBufPtr(PP_VOID);
Description	Reset I2S write, read point of I2S block.
Argument	None
Return value	None
Example	

1.7.4.6. PPDRV_I2S_WriteBuf

Prototype	PP_VOID PPDRV_I2S_WriteBuf(PP_U8 IN *pu8Buf, PP_U32 IN u32Size);
Description	Fill pcm audio buffer with data to be transmitted.
Argument	pu8Buf : audio data buffer 의 pointer u32Size : audio data size
Return value	None
Example	

1.7.4.7. PPDRV_I2S_GetWriteBuffAvailSize

Prototype	PP_S32 PPDRV_I2S_GetWriteBuffAvailSize(PP_VOID);
Description	Get the size which can be written to pcm audio buffer.
Argument	None
Return value	Data size which can be written to pcm audio buffer
Example	

1.7.5. GPIO

1.7.5.1. Data Type

1) PP_GPIO_INTR_MODE_E

A. Syntax

```
typedef enum ppGPIO_INTR_MODE_E
{
    eHIGH_LEVEL = 2,
    eLOW_LEVEL,
    eNEGATIVE_EDGE = 5,
    ePOSITIVE_EDGE,
    eDUAL_EDGE,
}PP_GPIO_INTR_MODE_E;
```

B. Members

eHIGH_LEVEL	Interrupt trigs on high level
eLOW_LEVEL	Interrupt trigs on low level
eNEGATIVE_EDGE	Interrupt trigs on negative edge
ePOSITIVE_EDGE	Interrupt trigs on positive edge
eDUAL_EDGE	Interrupt trigs on both edge

1.7.5.2. PPDRV_GPIO_Initialize

Prototype	PP_VOID PPDRV_GPIO_Initialize(PP_VOID)
Description	Initialize GPIO block
Argument	none
Return value	none
Example	

1.7.5.3. PPDRV_GPIO_SetIntrMode

Prototype	PP_VOID PPDRV_GPIO_SetIntrMode(PP_S32 IN s32Dev, PP_S32 IN s32Ch, PP_GPIO_INTR_MODE_E IN enMode)
Description	Set interrupt mode of target GPIO channel
Argument	s32Dev : gpio device number s32Ch : gpio channel enMode : interrupt mode
Return value	none
Example	

1.7.5.4. PPDRV_GPIO_SetDir

Prototype	PP_VOID PPDRV_GPIO_SetDir(PP_S32 IN s32Dev, PP_S32 IN s32Ch, PP_GPIO_DIR_E IN enDir, PP_S32 s32OutVal)
Description	Set input/output direction of target channel
Argument	s32Dev : gpio device number s32Ch : gpio channel enDir : input/ouput direction s32OutVal : Initial value when output mode is set.
Return value	none
Example	

1.7.5.5. PPDRV_GPIO_GetDir

Prototype	PP_GPIO_DIR_E PPDRV_GPIO_GetDir(PP_S32 IN s32Dev, PP_S32 IN s32Ch)
Description	Check input/output mode of each GPIO channel
Argument	s32Dev : gpio device number s32Ch : gpio channel
Return value	Input/output mode
Example	

1.7.5.6. PPDRV_GPIO_SetIRQEnable

Prototype	PP_VOID PPDRV_GPIO_SetIRQEnable(PP_S32 IN s32Dev, PP_S32 IN s32Ch, PP_S32 IN s32Enable)
Description	Interrupt enable
Argument	s32Dev : gpio device number s32Ch : gpio channel s32Enable : enable flag
Return value	none
Example	

1.7.5.7. PPDRV_GPIO_GetIRQEnable

Prototype	PP_S32 PPDRV_GPIO_GetIRQEnable(PP_S32 IN s32Dev, PP_S32 IN s32Ch)
Description	Check interrupt enable state of target channel
Argument	s32Dev : gpio device number s32Ch : gpio channel
Return value	Interrupt enable state of target GPIO channel
Example	

1.7.5.8. PPDRV_GPIO_SetValue

Prototype	PP_RESULT_E PPDRV_GPIO_SetValue(PP_S32 IN s32Dev, PP_S32 IN s32Ch, PP_S32 IN s32Value)
Description	Set GPIO output value
Argument	s32Dev : gpio device number s32Ch : gpio channel s32Value : gpio output value
Return value	eSUCCESS: success else: fail
Example	

1.7.5.9. PPDRV_GPIO_GetOutValue

Prototype	PP_S32 PPDRV_GPIO_GetOutValue(PP_S32 IN s32Dev, PP_S32 IN s32Ch)
Description	Check current value of Gpio output
Argument	s32Dev : gpio device number s32Ch : gpio channel
Return value	Gpio output value
Example	

1.7.5.10. PPDRV_GPIO_GetValue

Prototype	PP_S32 PPDRV_GPIO_GetValue(PP_S32 IN s32Dev, PP_S32 IN s32Ch)
Description	Get GPIO input value
Argument	s32Dev : gpio device number s32Ch : gpio channel
Return value	Gpio input value
Example	

1.7.5.11. PPDRV_GPIO_SetISR

Prototype	PP_VOID PPDRV_GPIO_SetISR(PP_S32 IN s32Dev, PP_S32 IN s32Ch, GPIO_ISR_CALLBACK IN cbISR)
Description	Register callback function which is called by Interrupt service routine. Registered callback function will be executed whenever interrupt is happened
Argument	s32Dev : gpio device number s32Ch : gpio channel cbISR : Pointer to callback function
Return value	none
Example	

1.7.6. ADC

1.7.6.1. PPDRV_ADC_Initialize

Prototype	PP_VOID PPDRV_ADC_Initialize(PP_VOID)
Description	Initialize ADC block
Argument	none
Return value	none
Example	

1.7.6.2. PPDRV_ADC_Start

Prototype	PP_VOID PPDRV_ADC_Start(PP_U32 IN u32ChBits, PP_U8 IN u8Block)
Description	Start ADC conversion
Argument	u32ChBits : Bit combination of ADC channel to be converted u8Block : Decide whether to block until conversion is finished
Return value	none

Example	{
	
	PP_U32 u32ChBit;
	PP_U32 u32Value[2];
	PP_U32 i;
	
	PPDRV_ADC_Initialize();
	u32ChBit = ((1<<1) | (1<<0)); // ch0, ch1
	
	while(1){
	PPDRV_ADC_Start(u32ChBit, 1);
	
	for(i=0;i<2;i++){
	u32Val[i] = PPDRV_ADC_GetData(i);
	LOG_DEBUG("ADC Ch:%d, Value: 0x%x\n", i, u32Val[i]);
	}

	OSAL_sleep(1000); } }

1.7.6.3. PPDRV_ADC_GetReady

Prototype	PP_U32 PPDRV_ADC_GetReady(PP_VOID)
Description	Check whether ADC conversion is finished
Argument	None
Return value	Shows whether conversion is finished or not for each channel. The state of each channel is represented in each bit of return value. (bit0: ch0, bit1: ch1) 1: ready 0: not ready
Example	

1.7.6.4. PPDRV_ADC_GetData

Prototype	PP_U32 PPDRV_ADC_GetData(PP_S32 IN s32Ch)
Description	Get the result of ADC conversion
Argument	s32Ch : ADC channel
Return value	Result of ADC conversion for target channel
Example	

1.7.7. WDT

Watchdog timer will be incremented from zero as soon as it is started. The counting value of Watchdog reaches a certain value, WDT will generate timeout interrupt and system will be reset after 2 msec. PPDRV_WDT_KeepAlive resets timer count value to 0 and re-start watchdog timer. While Watchdog is enabled, PPDRV_WDT_KeepAlive has to be called periodically before timeout is happened to prevent the system being reset

1.7.7.1. Data Type

1) PP_WDT_TIMEOUT_E

A. Syntax

```
typedef enum ppWDT_TIMEOUT_E
{
        eWDT_TIMEOUT_POW2_6 = 0,
        eWDT_TIMEOUT_POW2_8,
        eWDT_TIMEOUT_POW2_10,
        eWDT_TIMEOUT_POW2_11,
        eWDT_TIMEOUT_POW2_12,
        eWDT_TIMEOUT_POW2_13,
        eWDT_TIMEOUT_POW2_14,
        eWDT_TIMEOUT_POW2_15,
        eWDT_TIMEOUT_POW2_17,
        eWDT_TIMEOUT_POW2_19,
        eWDT_TIMEOUT_POW2_21,
        eWDT_TIMEOUT_POW2_23,
        eWDT_TIMEOUT_POW2_25,
        eWDT_TIMEOUT_POW2_27,
        eWDT_TIMEOUT_POW2_29,
        eWDT_TIMEOUT_POW2_31,
} PP_WDT_TIMEOUT_E;
```

B. Members

eWDT_TIMEOUT_POW2_6	Wdt clock period x 2^{6} (0.009 ms)
eWDT_TIMEOUT_POW2_8	Wdt clock period x 2^{8} (0.037 ms)
eWDT_TIMEOUT_POW2_10	Wdt clock period x 2^{10} (0.151 ms)
eWDT_TIMEOUT_POW2_11	Wdt clock period x 2^{11} (0.303 ms)
eWDT_TIMEOUT_POW2_12	Wdt clock period x 2^{12} (0.606 ms)
eWDT_TIMEOUT_POW2_13	Wdt clock period x 2^{13} (1.213 ms)
eWDT_TIMEOUT_POW2_14	Wdt clock period x 2^{14} (2.427 ms)
eWDT_TIMEOUT_POW2_15	Wdt clock period x 2^{15} (4.854 ms)
eWDT_TIMEOUT_POW2_17	Wdt clock period x 2^{17} (19.418 ms)
eWDT_TIMEOUT_POW2_19	Wdt clock period x 2^{19} (77.672 ms)
eWDT_TIMEOUT_POW2_21	Wdt clock period x 2^{21} (310.689 ms)

eWDT_TIMEOUT_POW2_23	Wdt clock period x 2^{23} (1,242.756 ms)
eWDT_TIMEOUT_POW2_25	Wdt clock period x 2^{25} (4,971.026 ms)
eWDT_TIMEOUT_POW2_27	Wdt clock period x 2^{27} (19,884.107 ms)
eWDT_TIMEOUT_POW2_29	Wdt clock period x 2^{29} (79,536.431 ms)
eWDT_TIMEOUT_POW2_31	Wdt clock period x 2^{31} (318,145.726 ms)

1.7.7.2. PPDRV_WDT_Initialize

Prototype	PP_VOID PPDRV_WDT_Initialize(PP_WDT_TIMEOUT_E IN enTimeOut);
Description	Initialize WDT block
Argument	enTimeOut : Timeout value
Return value	None
Example	None

1.7.7.3. PPDRV_WDT_SetISR

Prototype	PP_VOID PPDRV_WDT_SetISR(WDT_ISR_CALLBACK IN cbISR)
Description	Register callback function which is called by Interrupt service routine. Registered callback function will be executed whenever interrupt is happened
Argument	cbISR : Pointer to callback function
Return value	none
Example	

1.7.7.4. PPDRV_WDT_SetEnable

Prototype	PP_VOID PPDRV_WDT_SetEnable(PP_U8 IN u8Enable)
Description	WDT enable
Argument	u8Enable : enable flag
Return value	none
Example	

1.7.7.5. PPDRV_WDT_KeepAlive

Prototype	PP_VOID PPDRV_WDT_KeepAlive(PP_VOID)
Description	Restart WDT
Argument	None
Return value	none
Example	

1.7.8. DMA

1.7.8.1. PPDRV_DMA_Initialize

Prototype	PP_VOID PPDRV_DMA_Initialize(PP_VOID)
Description	Initialize DMA block
Argument	None
Return value	None
Example	None

1.7.8.2. PPDRV_DMA_M2M_Word

Prototype	PP_RESULT_E PPDRV_DMA_M2M_Word(PP_S32 IN s32Ch, PP_U32 IN *SrcAddr, PP_U32 IN *DstAddr, PP_U32 IN u32Size, PP_U32 IN u32Timeout)
Description	Memory to memory dma (word align)
Argument	s32Ch : DMA channel SrcAddr : Pointer to source memory address DstAddr : Pointer to destination memory address
Return value	eSUCCESS : success eERROR_TIMEOUT : timeout else : fail

Example	{
	PP_S32 s32DMACh = 0;
	PP_U32 u32DMASize = 0x100000;
	PP_U32 u32Timeout = 1000; //1000 ms
	PP_RESULT_E enRet;

	enRet = PPDRV_DMA_M2M_Word(s32DMACh, (uint32*)pSrcMem, (uint32*)pDstMem, u32DMASize, u32Timeout);
	
	if(enRet == eSUCCESS){
	// success
	}else if(enRet == eERROR_TIMEOUT){
	// timeout
	}else{
	// fail
	}
	
	}

1.7.8.3. PPDRV_DMA_GetDone

Prototype	PP_U32 PPDRV_DMA_GetDone(PP_S32 IN s32Ch, PP_RESULT_E OUT *penResult)
Description	Check whether DMA transfer of target channel is finished
Argument	s32Ch : DMA channel penResult : DMA result eSUCCESS : success eERROR_TIMEOUT : timeout else : fail
Return value	Return whether DMA is finished regardless of success or fail 0: Not yet finished. 1: Finished

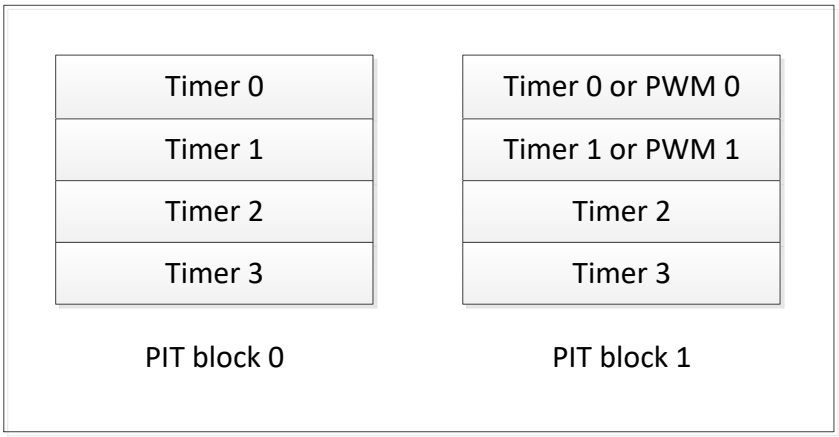
Example	{
	PP_S32 s32DMACh = 0;
	PP_U32 u32DMASize = 0x100000;
	PP_U32 u32Timeout = 0; //no wait
	PP_RESULT_E enRet;
	
	PPDRV_DMA_M2M_Word(s32DMACh, (uint32*)pSrcMem, (uint32*)pDstMem, u32DMASize, u32Timeout);
	
	while(!PPDRV_DMA_GetDone(s32DMACh, &enRet)){

```
            OSAL_sleep(1);
}
if(enRet == eSUCCESS){
// success
}else if(enRet == eERROR_TIMEOUT){
// timeout
}else{
// fail
}

}
```

1.7.9. Timer & PWM

PI5008 contains 2 Programmable Interval Timer(PIT) blocks, and each block is composed of 4 32bit timer or 16bit PWM. Only channel 0 and channel 1 of the PIT block1 can be set as PWM.



1.7.9.1. Data Type

1) PP_PIT_MODE_E

A. Syntax

```
typedef enum ppPIT_MODE_E
{
        ePIT_MODE_TIMER = 0,
        ePIT_MODE_PWM,
}PP_PIT_MODE_E;
```

B. Members

ePIT_MODE_TIMER	32bit timer mode
ePIT_MODE_PWM	16bit PWM mode

2) PP_TIMER_CFG_S

A. Syntax

```
typedef struct ppTIMER_CFG_S {
    PP_PIT_MODE_E enTimerMode[4];     // IN
    PP_U32 u32PWMInitVal;             // IN
    PP_U32 u32nTimerCh;               // OUT
    PP_U32 u32nPWMCh;                 // OUT
} PP_TIMER_CFG_S;
```

B. Members

enTimerMode	Timer or PWM mode
u32PWMInitVal	PWM initial value 0: The PWM output is LOW when the channel is disabled. The low-period PWM counter will be counted first before toggling the out to HIGH when the channel is enabled 1: The PWM output is HIGH when the channel is disabled. The high-period PWM counter will be counted first before toggling the out to LOW when the channel is enabled
u32nTimerCh	The number of timer channel (output value)
u32nPWMCh	The number of PWM channel (output value)

1.7.9.2. PPDRV_TIMER_Initialize

Prototype	PP_RESULT_E PPDRV_TIMER_Initialize(PP_U32 IN u32Dev, PP_TIMER_CFG_S IN *pstCfg)
Description	Initialize Timer block
Argument	u32Dev : Timer device number pstCfg : Timer setting value
Return value	eSUCCESS : success else : fail

Example	#define configTICK_RATE_HZ 1000
	
	// timer isr
	PP_VOID tickIsr(PP_VOID)
	{
	}
	
	{
	
	PP_TIMER_CFG_S stCfg
	PP_U32 u32Dev = 0;
	PP_S32 s32Ch = 0;
	PP_U32 u32TimerClk = 6750000; // Timer clock 6.75MHz
	
	stCfg.enTimerMode[0] = ePIT_MODE_TIMER; // timer ch0
	stCfg.enTimerMode[1] = ePIT_MODE_TIMER; // timer ch1
	stCfg.enTimerMode[2] = ePIT_MODE_TIMER; // timer ch2
	stCfg.enTimerMode[3] = ePIT_MODE_TIMER; // timer ch3
	
	if (PPDRV_TIMER_Initialize(u32Dev, &stCfg) != eSUCCESS)
	{
	OSAL_system_panic(eERROR_FAILURE);
	}
	
	PPDRV_TIMER_SetISR(u32Dev, s32Ch, tickIsr);
	PPDRV_TIMER_Stop(u32Dev, s32Ch);

	PPDRV_TIMER_SetPeriod(u32Dev, s32Ch, (u32TimerClk / configTICK_RATE_HZ)); // timer period 1ms
	PPDRV_TIMER_IRQControl(u32Dev, s32Ch, 1);
	PPDRV_TIMER_Start(u32Dev, s32Ch);
	
	}

1.7.9.3. PPDRV_TIMER_SetISR

Prototype	PP_VOID PPDRV_TIMER_SetISR(PP_U32 IN u32Dev, PP_S32 IN s32Ch, TIMER_ISR_CALLBACK IN cbISR)
Description	Register callback function which is called by Interrupt service routine. Registered callback function will be executed whenever interrupt is happened
Argument	u32Dev : Timer device number
Return value	none
Example	

1.7.9.4. PPDRV_TIMER_SetPeriod

Prototype	PP_VOID PPDRV_TIMER_SetPeriod(PP_U32 IN u32Dev, PP_S32 IN s32Ch, PP_U32 IN u32Period)
Description	Timer count down from the period value and generate an interrupt when the count reaches zero.
Argument	u32Dev : Timer device number s32Ch : Timer channel u32Period : Timer period count value
Return value	none
Example	

1.7.9.5. PPDRV_TIMER_IRQControl

Prototype	PP_VOID PPDRV_TIMER_IRQControl(PP_U32 IN u32Dev, PP_S32 IN s32Ch, PP_U32 IN u32Enable)
Description	Enable or disable timer interrupt
Argument	u32Dev : Timer device number s32Ch : Timer channel u32Enable : enable (1), disable (0)
Return value	none
Example	

1.7.9.6. PPDRV_TIMER_Start

Prototype	PP_VOID PPDRV_TIMER_Start(PP_U32 IN u32Dev, PP_S32 IN s32Ch)
Description	Start timer
Argument	u32Dev : Timer device number s32Ch : Timer channel
Return value	none
Example	

1.7.9.7. PPDRV_TIMER_Stop

Prototype	PP_VOID PPDRV_TIMER_Stop(PP_U32 IN u32Dev, PP_S32 IN s32Ch)
Description	Stop timer
Argument	u32Dev : Timer device number s32Ch : Timer channel
Return value	none
Example	

1.7.9.8. PPDRV_TIMER_Reset

Prototype	PP_VOID PPDRV_TIMER_Reset(PP_U32 IN u32Dev)
Description	Reset timer

Argument	u32Dev : Timer device number
Return value	none
Example	

1.7.9.9. PPDRV_TIMER_CountRead

Prototype	PP_U32 PPDRV_TIMER_CountRead(PP_U32 IN u32Dev, PP_S32 IN s32Ch)
Description	Read current count value
Argument	u32Dev : Timer device number s32Ch : Timer channel
Return value	Current count value
Example	

1.7.9.10. PPDRV_PWM_SetPeriod

Prototype	PP_VOID PPDRV_PWM_SetPeriod(PP_U32 IN u32Dev, PP_S32 IN s32Ch, PP_U16 IN u16HighPeriod, PP_U16 IN u16LowPeriod)
Description	Set PWM high period and low period
Argument	u32Dev : Timer device number s32Ch : Timer channel u16HighPeriod : high period count value u16LowPeriod : low period count value
Return value	none

Example	{
	
	PP_TIMER_CFG_S stCfg
	PP_U32 u32Dev = 1;
	PP_S32 s32Ch = 0;
	PP_U32 u32TimerClk = 6750000; // Timer clock 6.75MHz
	
	stCfg.enTimerMode[0] = ePIT_MODE_PWM; // PWM ch0
	stCfg.enTimerMode[1] = ePIT_MODE_PWM; // PWM ch1
	stCfg.enTimerMode[2] = ePIT_MODE_TIMER; // timer ch2
	stCfg.enTimerMode[3] = ePIT_MODE_TIMER; // timer ch3
	
	if (PPDRV_TIMER_Initialize(u32Dev, &stCfg) != eSUCCESS)
	{
	OSAL_system_panic(eERROR_FAILURE);
	}
	
	PPDRV_PWM_SetPeriod (u32Dev, u32Ch, (u32TimerClk /1000), (u32TimerClk /1000)); // high period: 1ms, low period: 1ms
	PPDRV_PWM_Start (u32Dev, u32Ch);
	
	}

1.7.9.11. PPDRV_PWM_Start

Prototype	PP_VOID PPDRV_PWM_Start(PP_U32 IN u32Dev, PP_S32 IN s32Ch)
Description	Start PWM
Argument	u32Dev : Timer device number s32Ch : Timer channel
Return value	none
Example	

1.7.9.12. PPDRV_PWM_Stop

Prototype	PP_VOID PPDRV_PWM_Stop(PP_U32 IN u32Dev, PP_S32 IN s32Ch)
Description	Stop PWM
Argument	u32Dev : Timer device number s32Ch : Timer channel
Return value	none
Example	

1.7.9.13. VPU

TBD

1.7.9.14. Display Driver

TBD

1.7.9.15. Video Input

TBD

1.7.9.16. Video Output

TBD

1.7.9.17. SVM

TBD

1.8. Storage Configuration

PI5008K SDK support two storage configurations; SPI NOR + SD, SPI NAND + SD

1) SPI NAND +SD/MMC Card

   PI5008 SDK includes FTL(Flash Translation Layer) to support NAND Flash. It also includes FAT32 File System to support SD/MMC card.

   NAND flash is used to hold main binary, bootloader and user-defined data

   MMC/SD can be used to store captured image or upgrade file.

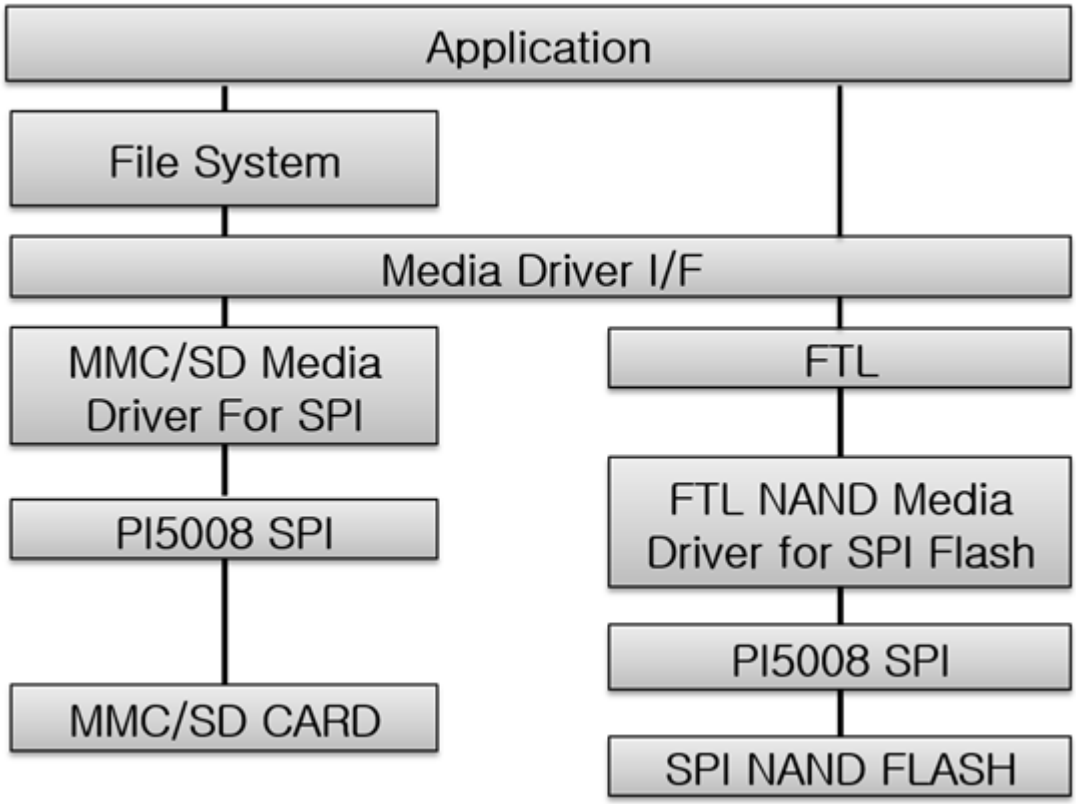


Figure 7 SPI NAND + SD Configuration

2) SPI NOR +SD/MMC Card

PI5008 SDK support SPI NOR + SD/MMC Card.

It supports NOR flash using its own driver.

NOR flash is used to hold main binary, bootloader and user-defined data

MMC/SD can be used to store captured image or upgrade file.

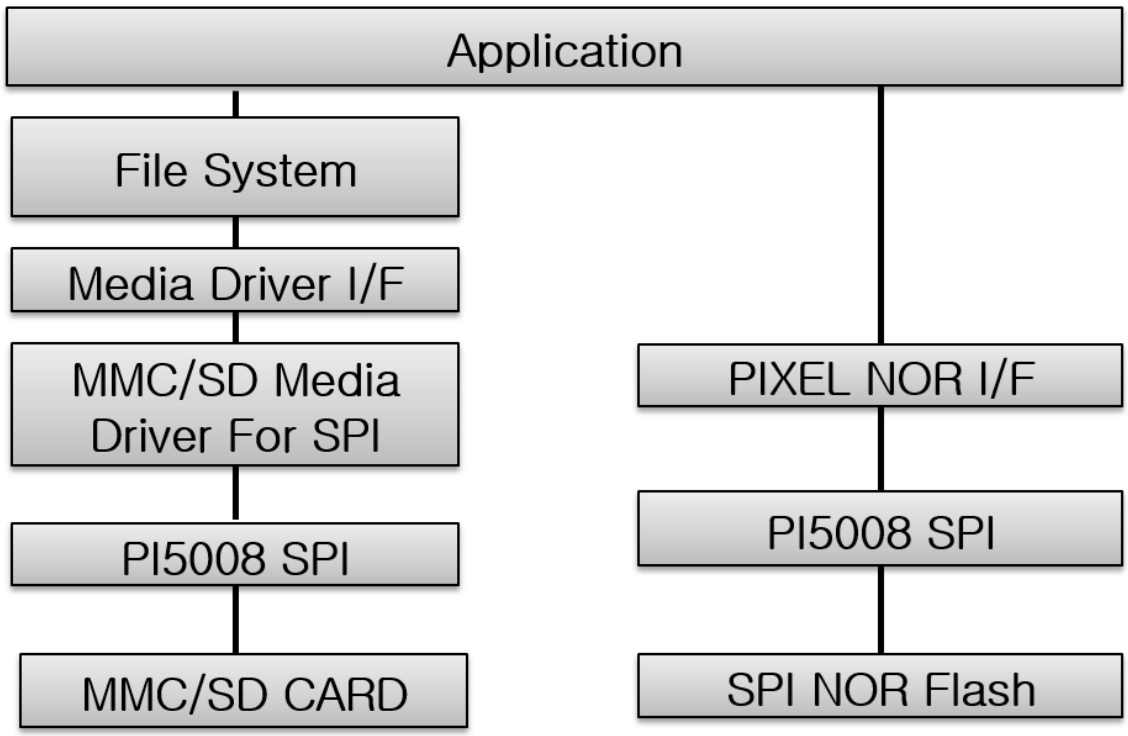


Figure 8 SPI NOR+ SD Configuration

2. Revision History

Version	Date	Description
V0.0	20180510	
V0.1	20180608	