

**ARM® CORTEX®-M4****32-BIT MICROCONTROLLER****NuMicro® Family****N9H26****Non-OS Library Reference Guid**

*The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.*

*Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.*

*All data and specifications are subject to change without notice.*

For additional information or questions, please contact: Nuvoton Technology Corporation.

[www.nuvoton.com](http://www.nuvoton.com)

# Table of Contents

<b>1. AAC Library .....</b>	<b>19</b>
1.1. AAC Library Overview .....	19
1.2. AAC API .....	19
DrvAAC_Open.....	19
DrvAAC_Close .....	20
DrvAAC_Decoder.....	20
DrvAAC_Encoder.....	21
1.3. Example code .....	21
<b>2. AES Library Introduction .....</b>	<b>22</b>
2.1. Feature .....	22
2.2. API Data Structure.....	22
KEYSIZE .....	22
2.3. API Function.....	22
AES_Initial.....	22
AES_Final .....	23
AES_Encrypt.....	23
AES_Encrypt_Async .....	24
AES_Decrypt.....	24
AES_Decrypt_Async.....	25
AES_Flush.....	26
AES_Check_Status.....	26
AES_Enable_Interrupt .....	26
AES_Disable_Interrupt .....	27
2.4. Error Code Table .....	27
<b>3. Audio ADC Library Overview .....</b>	<b>28</b>
3.1. Audio ADC Library APIs Specification .....	28
DrvAUR_Open.....	28
DrvAUR_Close .....	29
DrvAUR_InstallCallback .....	29
DrvAUR_EnableInt.....	30
DrvAUR_DisableInt.....	30
DrvAUR_AutoGainCtrl.....	30
DrvAUR_AutoClampingGain .....	32

DrvAUR_SetSampleRate .....	32
DrvAUR_AudioI2cRead .....	33
DrvAUR_AudioI2cWrite .....	34
DrvAUR_SetDigiMicGain .....	34
DrvAUR_StartRecord .....	35
DrvAUR_StopRecord.....	36
DrvAUR_SetDataOrder .....	36
<b>4. AVI Library Overview .....</b>	<b>38</b>
4.1. Video render .....	38
4.2. How to use AVI player library.....	38
4.3. AVI player user callback .....	39
4.4. AVI playback information .....	39
4.5. API Enumeration .....	39
4.6. API Structure .....	40
4.7. Functions .....	40
aviStopPlayFile .....	40
aviPlayFile.....	41
aviGetFileInfo .....	42
aviSetPlayVolume .....	43
aviSetRightChannelVolume .....	43
4.8. Error Code Table .....	44
4.9. MP3 Library Overview .....	46
How to use MP3 player library.....	46
MP3 player user callback .....	46
MP3 player information.....	46
4.10. API Enumeration .....	46
4.11. Structure.....	47
4.12. Functions .....	47
mflMediaPlayer .....	47
mflGetMovieInfo.....	48
mflPlayControl .....	49
<b>5. BLT Library Introduction.....</b>	<b>50</b>
5.1. Feature .....	50

5.2.	Transformation Matrix .....	50
	Scaling.....	51
	Rotation .....	51
	Translation.....	51
5.3.	Amendment to User Transformation Matrix .....	52
5.4.	Pixel Mapping.....	56
5.5.	Color Transformation .....	57
5.6.	Palette .....	57
	Palette entries .....	58
	Source image in palette index format .....	58
5.7.	API Data Structure.....	58
	E_BLT_INT_TYPE .....	58
	E_DRVBLT_FILLOP .....	58
	E_DRVBLT_REVEAL_ALPHA.....	59
	E_DRVBLT_TRANSFORM_FLAG .....	59
	E_DRVBLT_BMPPIXEL_FORMAT.....	59
	E_DRVBLT_DISPLAY_FORMAT .....	60
	E_DRVBLT_FILL_STYLE.....	60
	E_DRVBLT_PALETTE_ORDER.....	60
	S_DRVBLT_MATRIX.....	60
	S_DRVBLT_ARGB16.....	61
	S_DRVBLT_ARGB8.....	61
	S_DRVBLT_SRC_IMAGE .....	61
	S_DRVBLT_DEST_FB .....	61
5.8.	API Function.....	62
	bltOpen .....	62
	bltClose.....	62
	bltSetTransformMatrix .....	63
	bltGetTransformMatrix .....	63
	bltSetSrcFormat.....	63
	bltGetSrcFormat .....	64
	bltSetDisplayFormat .....	64
	bltGetDisplayFormat .....	64
	bltEnableInt .....	65
	bltDisableInt .....	65
	bltIsIntEnabled .....	65
	bltPollInt.....	66
	bltInstallCallback.....	66
	bltSetColorMultiplier .....	66
	bltGetColorMultiplier .....	67
	bltSetColorOffset.....	67
	bltGetColorOffset .....	67
	bltSetSrcImage .....	68
	bltSetDestFrameBuf .....	68
	bltSetARGBFillColor .....	68
	bltGetARGBFillColor .....	69

bltGetBusyStatus .....	69
bltSetFillAlpha .....	69
bltGetFillAlpha.....	70
bltSetTransformFlag.....	70
bltGetTransformFlag .....	71
bltSetPaletteEndian .....	71
bltGetPaletteEndian.....	71
bltSetColorPalette.....	72
bltSetFillOP .....	72
bltGetFillOP .....	72
bltSetFillStyle .....	73
bltGetFillStyle .....	73
bltSetRevealAlpha .....	73
bltGetRevealAlpha .....	74
bltTrigger.....	74
bltSetRGB565TransparentColor .....	74
bltGetRGB565TransparentColor .....	75
bltSetRGB565TransparentColorCtl.....	75
bltGetRGB565TransparentColorCtl.....	75
bltFlush.....	76
5.9. Error Code Table .....	76
<b>6. CRC Library Introduction .....</b>	<b>77</b>
6.1. Feature .....	77
6.2. API Data Structure.....	77
E_CRC_CHANNEL_INDEX .....	77
E_CRC_OPERATION.....	77
E_CRC_MODE.....	78
E_WRITE_LENGTH.....	78
E_DATA_1sCOM.....	78
E_DATA_REVERSE.....	78
E_TRANSFER_MODE .....	79
S_CRC_CHANNEL_INFO.....	79
S_CRC_DESCRPT_SETTING .....	79
6.3. API Function.....	79
CRC_Init .....	79
CRC_Exit .....	80
CRC_Request .....	80
CRC_Free .....	80
CRC_FindandRequest .....	81
CRC_Run .....	81
6.4. Error Code Table .....	82
<b>7. EDMA Library Overview .....</b>	<b>83</b>

7.1. Programming Guide.....	83
System Overview.....	83
Block Diagram .....	84
EDMA Control .....	84
7.2. APIs Specification Functions .....	86
EDMA_Init.....	86
EDMA_Exit.....	86
VDMA_FindandRequest .....	87
PDMA_FindandRequest.....	87
EDMA_SetupHandlers.....	87
EDMA_SetupSingle .....	88
EDMA_Free .....	89
EDMA_SetupSG .....	89
EDMA_FreeSG .....	90
EDMA_SetupCST .....	90
EDMA_ClearCST .....	91
EDMA_Trigger .....	91
EDMA_IsBusy .....	91
EDMA_SetAPB .....	92
EDMA_SetWrapINTType .....	92
EDMA_SetDirection .....	93
7.3. Error Code Table .....	94
<b>8. EMAC Library Introduction.....</b>	<b>95</b>
8.1. Feature .....	95
8.2. API Data Structure.....	95
S_FrameDescriptor.....	95
S_Etheader .....	95
S_MACFrame.....	96
S_MACTxStatus: reserve .....	96
S_MACRxStatus: reserve .....	96
NVT_EMAC_T .....	96
8.3. API Function.....	96
EMAC_Init.....	96
EMAC_SetRxCallBack.....	97
EMAC_SendPacket.....	97
EMAC_EnableWakeOnLan .....	98
EMAC_Exit.....	98
8.4. Error Code Table .....	98
<b>9. Font Library Overview .....</b>	<b>99</b>
9.1. Font Library API.....	99

InitFont .....	99
DemoFont_PaintA .....	100
UnInitFont .....	101
DemoFont_Rect .....	101
DemoFont_RectClear .....	102
Font_ClrFrameBuffer .....	103
DemoFont_Border .....	103
DemoFont_ChangeFontColor .....	104
DemoFont_GetFontColor .....	105

## 10. GNAND Library Overview ..... 106

10.1. GNAND Library Introduction .....	106
10.2. Programming Guide.....	106
System Overview.....	106
Initialize GNAND Library.....	107
GNAND work with Nuvoton FAT Library .....	107
NAND driver function set .....	107
10.3. API Function.....	110
GNAND_InitNAND.....	110
GNAND_MountNandDisk .....	112
GNAND_read.....	112
GNAND_write .....	113
GNAND_block_erase.....	114
GNAND_chip_erase.....	114
GNAND_UnMountNandDisk .....	115
10.4. Example code .....	116
10.5. Error Code Table .....	116

## 11. GPIO Library Overview ..... 117

11.1. API Functions .....	117
gpio_open.....	117
gpio_configure.....	117
gpio_readport .....	118
gpio_setportdir .....	118
gpio_setportval .....	119
gpio_setportpull.....	119
gpio_setdebounce .....	120
gpio_getdebounce.....	120
gpio_setsrcgrp .....	121
gpio_getsrcgrp .....	121
gpio_setintmode .....	122
gpio_getintmode .....	122
gpio_setlatchtrigger .....	123

gpio_getlatchtrigger.....	123
gpio_getlatchval .....	124
gpio_gettriggersrc.....	124
gpio_cleartriggersrc.....	125

## 12. H264 Codec Library Introduction ..... 126

12.1.Feature .....	126
12.2.Rate Control for Encoder.....	126
12.3.API Data Structure.....	127
IOCTL COMMAND .....	127
FAVC_ENC_PARAM .....	128
12.4.API Function.....	130
H264Dec_Open .....	130
H264Enc_Open .....	131
H264_ioctl.....	131
H264Enc_Close.....	131
H264Dec_Close .....	132
nv_malloc .....	132
nv_free.....	132
12.5.Retun Code Table .....	133
AVC_RET .....	133

## 13. I2C Library Overview ..... 134

13.1.I2C Library APIs Specification.....	134
i2cInit .....	134
i2cOpen .....	134
i2cClose.....	135
i2cRead.....	135
i2cRead_OV .....	136
i2cWrite.....	137
i2cIoctl.....	137
i2cExit .....	138
13.2.Error Code Table .....	138

## 14. I2S Library Description ..... 140

14.1.API Functions .....	140
DrvI2S_Open .....	140
DrvI2S_Close .....	140
DrvI2S_StartPlay.....	141
DrvI2S_StopPlay.....	141

DrvI2S_StartRecord .....	142
DrvI2S_StopRecord .....	142
DrvI2S_SetSampleRate .....	142

## 15. JPEG Library Overview ..... 144

15.1.JPEG Overview .....	144
15.2.Programming Guide.....	144
System Overview.....	144
JPEG Operation Control.....	145
JPEG Library Constant Definition.....	151
JPEG Library Property Definition.....	153
15.3.JPEG API.....	154
jpegOpen .....	154
jpegClose.....	154
jpegInit .....	155
jpegGetInfo.....	155
jpegWait .....	156
jpegIsReady.....	156
jpegSetQTAB .....	157
jpegIoctl.....	157
15.4.Example code .....	162

## 16. PWM Library Overview ..... 163

16.1.Programming Guide.....	163
System Overview.....	163
Block Diagram .....	164
PWM Timer Control .....	164
PWM Library Constant Definition .....	167
PWM Library Property Definition.....	168
16.2.PWM API .....	169
PWM_Open.....	169
PWM_Close .....	169
PWM_SetClockSetting.....	170
PWM_GetEngineClock .....	170
PWM_SetTimerClk.....	171
PWM_SetTimerIO .....	172
PWM_Enable .....	172
PWM_IsTimerEnabled .....	173
PWM_SetTimerCounter .....	173
PWM_GetTimerCounter .....	174
PWM_EnableDeadZone.....	174
PWM_EnableInt.....	175
PWM_DisableInt.....	176

## **17. RFC Library Introduction.....181**

17.1. Feature .....	181
17.2. API Data Structure.....	181
E_RF_PNCTR_MODE .....	181
17.3. API Function.....	182
RF_Open .....	182
RF_Close.....	182
RF_Enable_Int.....	183
RF_Disable_Int .....	183
RF_Set_Puncture.....	183
RF_Get_Puncture .....	184
RF_Encrypt .....	184
RF_Decrypt.....	184
17.4. Error Code Table .....	185

**18. Rotation Library Overview ..... 186**

18.1. Rotation Library APIs Specification.....	186
rotOpen.....	186
rotClose .....	186
rotInstallISR .....	187
rotImageConfig.....	188
rotGetPacketPixelWidth.....	189
rotTrigger .....	190
18.2. Error Code Table .....	190

## **19. RSC Library Introduction.....191**

19.1. Feature .....	191
19.2. Function .....	191
RS_Open .....	191
RS_Close .....	192
RS_Enable_Int.....	192
RS_Disable_Int .....	192

RS_Encrypt .....	193
RS_Decrypt .....	193
19.3.Error Code Table .....	194
<b>20. RTC Library Overview.....</b>	<b>195</b>
20.1.Programming Guide.....	195
System Overview.....	195
System Power Control Flow .....	195
RTC Library Constant Definition.....	199
RTC Library Time and Date Definition .....	201
20.2.RTC API.....	201
RTC_Init.....	201
RTC_Open .....	202
RTC_Close .....	202
RTC_Read .....	203
RTC_WriteEnable .....	203
RTC_DoFrequencyCompensation.....	204
RTC_Ioctl.....	204
20.3.Example code .....	207
20.4.Error Code Table .....	208
<b>21. RTC Power Control Library Overview .....</b>	<b>209</b>
21.1.Power Control Function Overview.....	209
Power On.....	209
Power Off .....	209
21.2.Preliminary Definition .....	209
Power Key (PWRKEY).....	209
Power Control Signal (PWCE).....	209
Power Key Status (PWR_KEY) .....	210
Software Status (SW_STATUS) .....	210
Hardware Force Power Off Enable (HW_PCLR_EN) .....	210
Hardware Force Power Off Period (PCLR_TIME) .....	210
Power on (PWR_ON).....	210
Power-down Mode .....	210
Power key duration (POWER_KEY_DURATION_LENGTH).....	210
Interrupt Enable & Status .....	211
21.2.1. System Power On Control Flow.....	211
Power On from Power Key.....	211
Power On from RTC Alarm .....	212
Power On Source Judgment.....	215
Power On Exception.....	215

21.2.2. System Power Off Control Flow .....	215
Software Power Off .....	215
Hardware Force Power Off.....	216
21.3. Power Control Flow Truth Table.....	216
21.4. Power off Flow Example .....	217
Normal Power Off Flow .....	217
Software Busy/Crash Power Off Flow.....	218
<b>22. SDIO Library Overview .....</b>	<b>219</b>
22.1. SDIO Library .....	219
22.2. Programming Guide.....	219
System Overview.....	219
22.3. SDIO API.....	220
sdioOpen.....	220
sdioClose .....	221
sdioIoctl.....	221
22.4. SDIO API.....	222
sdioSdOpen .....	222
sdioSdClose .....	223
sdioSdRead.....	223
sdioSdWrite .....	224
22.5. Example code .....	225
22.6. Error Code Table .....	225
<b>23. SIC Library Overview.....</b>	<b>226</b>
23.1. Storage Interface Controller Library .....	226
23.2. Programming Guide.....	226
System Overview.....	226
NAND Driver and GNAND Library .....	227
23.3. SIC API.....	227
sicOpen.....	227
sicClose .....	228
sicIoctl.....	228
23.4. SIC / SD API .....	229
sicSdOpen.....	229
sicSdClose .....	230
sicSdRead .....	230

sicSdWrite .....	231
23.5.SIC / NAND API .....	232
nandInit0 .....	232
nand_ioctl .....	233
nandpread0 .....	233
nandpwrite0 .....	234
nand_is_page_dirty0 .....	235
nand_is_valid_block0 .....	236
nand_block_erase0 .....	237
nand_chip_erase0 .....	237
23.6.Example code .....	238
23.7.Error Code Table .....	238
<b>24. SPI Library Description.....</b>	<b>240</b>
24.1.API Functions .....	240
spiOpen.....	240
spiClose .....	241
spiIoctl.....	241
spiEnable .....	242
spiDisable .....	242
spiRead.....	242
spiWrite .....	243
spiEnableInt.....	243
spiDisableInt.....	244
spiInstallCallBack.....	244
spiSetGo .....	245
spiSetByteEndin .....	245
<b>25. SPU Library Description .....</b>	<b>247</b>
25.1.API Functions .....	247
spuOpen.....	247
spuStartPlay.....	247
spuStopPlay.....	248
spuClose .....	248
spuIoctl.....	249
spuDacOn.....	249
spuDacOff .....	250
spuEqOpen .....	250
spuEqClose .....	251
25.2.System Library Overview .....	251
25.3.Timer API Functions .....	251
sysSetTimerEvent.....	252

sysClearTimerEvent .....	253
sysClearWatchDogTimerCount .....	253
sysClearWatchDogTimerInterruptStatus .....	254
sysDelay .....	254
sysDisableWatchDogTimer .....	255
sysDisableWatchDogTimerReset .....	255
sysEnableWatchDogTimer .....	255
sysEnableWatchDogTimerReset .....	256
sysGetCurrentTime .....	256
sysGetTicks .....	257
sysInstallWatchDogTimerISR .....	257
sysResetTicks .....	258
sysSetLocalTime .....	258
sysSetTimerReferenceClock .....	259
sysSetWatchDogTimerInterval .....	260
sysStartTimer .....	261
sysStopTimer .....	261
sysUpdateTickCount .....	262
<b>25.4. UART Function .....</b>	<b>262</b>
sysGetChar .....	264
sysInitializeUART .....	264
sysPrintf .....	265
sysprintf .....	266
sysPutChar .....	266
sysUartInstallcallback .....	267
sysUartEnableInt .....	268
sysUartTransfer .....	268
<b>25.5. AIC Functions .....</b>	<b>269</b>
sysDisableInterrupt .....	271
sysEnableInterrupt .....	272
sysGetIBitState .....	272
sysGetInterruptEnableStatus .....	273
sysGetInterruptHighEnableStatus .....	273
sysInstallExceptionHandler .....	274
sysInstallFiqHandler .....	274
sysInstallIrqHandler .....	275
sysInstallIISR .....	275
sysSetAIC2SWMode .....	276
sysSetGlobalInterrupt .....	276
sysSetInterruptPriorityLevel .....	277
sysSetInterruptType .....	277
sysSetLocalInterrupt .....	278
<b>25.6. Cache Function .....</b>	<b>279</b>
sysDisableCache .....	279
sysEnableCache .....	279
sysFlushCache .....	280
sysGetCacheState .....	280
sysGetSdramSizebyMB .....	280
sysInvalidCache .....	281

sysSetCachePages.....	281
25.7. Clock Control function .....	282
sysGetExternalClock .....	282
sysSetSystemClock.....	283
sysSetDramClock .....	284
sysSetCPUClock.....	285
sysSetAPBClock.....	285
sysGetPLLOutputHz .....	286
sysGetSystemClock .....	287
sysGetDramClock.....	287
sysGetCPUClock.....	288
sysGetHCLK1Clock.....	288
sysGetAPBClock .....	289
sysSetPllClock .....	289
sysCheckPllConstraint.....	290
sysSetSystemDivider .....	290
25.8. Power management Function .....	291
sysPowerDown .....	292
25.9. Error Code Table .....	292
<b>26. Touch ADC Library Overview.....</b>	<b>293</b>
26.1. Touch ADC Library APIs Specification.....	293
DrvADC_Open.....	293
DrvADC_Close .....	293
DrvADC_InstallCallback .....	294
DrvADC_PenDetection.....	295
DrvADC_KeyDetection .....	297
DrvADC_VoltageDetection .....	298
26.2. Error Code Table .....	299
<b>27. UDC Library .....</b>	<b>300</b>
27.1. Programming Guide.....	300
System Overview.....	300
UDC Library Property Definition.....	301
Programming Flow .....	303
27.2. USB Device (UDC) API.....	304
udcOpen .....	304
udcClose .....	304
udcInit.....	304
udcDeinit .....	305
udcIsAttached.....	305
udcIsAttachedToHost .....	306

27.3. Mass Storage Class (MSCD) API.....	307
mscdInit .....	307
mscdDeinit.....	307
mscdFlashInit .....	307
mscdFlashInitNAND .....	308
mscdFlashInitExtend .....	309
mscdSdEnable .....	311
mscdNandEnable .....	311
mscdSdUserWriteProtectPin .....	312
mscdSdUserCardDetectPin .....	313
mscdMassEvent .....	313
27.4. USB Video Class (UVCD) API.....	314
uvcdInit.....	314
uvcdSendImage .....	316
uvcdIsReady .....	316
27.5. Example code .....	317
<b>28. USB Core Library Overview .....</b>	<b>318</b>
28.1. USB Core Library Overview .....	318
28.2. Data Structures .....	319
USB_DEV_T .....	319
28.3. Descriptor Structures .....	321
DEV_REQ_T .....	327
USB_DEV_ID_T .....	328
USB_DRIVER_T .....	331
URB_T .....	332
28.4. Data Transfer .....	332
28.5. Pipe Control .....	333
Transfer Type .....	334
Maximun Packet Size .....	334
Direction.....	334
Device Number.....	335
Endpoint Number .....	335
Data Toggle.....	335
Speed.....	335
Pipe Creation.....	335
28.6. Control Transfer .....	337
28.7. Bulk Transfer .....	340
28.8. Interrupt Transfer.....	343

28.9. USB Core Library API.....	345
USB_PortInit.....	345
USB_PortDisable .....	346
InitUsbSystem .....	346
DeInitUsbSystem.....	347
UMAS_InitUmasDriver .....	348
USB_RegisterDriver.....	349
USB_DeregisterDriver .....	350
USB_AllocateUrb .....	350
USB_FreeUrb.....	351
USB_SubmitUrb .....	351
USB_UnlinkUrb.....	352
USB_SendBulkMessage.....	353
USB_free.....	355
<b>29. VIDEOIN Library Overview.....</b>	<b>357</b>
29.1.Features.....	357
29.2. VIDEOIN Library Description .....	358
29.3. VIDEOLIN Library API .....	358
register_vin_device.....	358
Init .....	359
Open .....	361
Close.....	362
SetPipeEnable.....	363
InstallCallback.....	364
EnableInt .....	366
DisableInt .....	366
SetInputType .....	367
SetSensorPolarity .....	368
SetDateFormatAndOrder.....	369
SetCropWinSize .....	370
SetCropWinStartAddr .....	371
PreviewPipeSize .....	372
EncodePipeSize .....	373
SetStride .....	374
GetStride.....	374
SetPlanarFormat .....	375
SetBaseStartAddress.....	376
SetOperationMode.....	378
GetOperationMode .....	379
SetColorEffect .....	379
SetColorEffectParameter .....	380
SetMotionDet .....	380
SetMotionDetEx .....	381
SetStandardCCIR656 .....	382
SetShadowRegister .....	383
29.4.Error Code Table .....	384

<b>30. VPE Library Overview .....</b>	<b>385</b>
30.1.Features.....	385
30.2.VPE Library Description .....	386
30.3.VPE Library API.....	386
vpeOpen .....	386
vpeClose .....	387
vpeInstallCallback .....	387
vpeEnableInt.....	389
vpeDisableInt.....	389
vpeIoctl.....	390
30.4.Error Code Table .....	398
<b>31. VPOST Overview.....</b>	<b>399</b>
31.1.VPOST Overview.....	399
How to build the VPOST library.....	399
31.2.API Enumeration .....	400
31.3.API Structure .....	403
31.4.API Functions .....	405
vpostGetFrameBuffer .....	405
vpostSetFrameBuffer.....	405
vpostLCMInit .....	406
vpostLCMDeinit.....	406
31.5.Error Code Table .....	407
<b>32. Revision History.....</b>	<b>408</b>

# 1. AAC Library

## 1.1. AAC Library Overview

This library is designed to make user application to use N9H26 AAC IMDCT/MDCT more easily.  
The AAC library has the following features:

- AAC IMDCT for decoder.
- AAC MDCT for encoder.

SDK Non-OS provide one library and one sample code to test AAC IMDCT/MDCT function. User could use them to verify hardware IP.

## 1.2. AAC API

### *DrvAAC\_Open*

#### Synopsis

```
VOID DrvAAC_Open(VOID)
```

#### Description

This function enables the AAC engine clock.

#### Parameter

None

#### Return Value

None

#### Example

```
DrvAAC_Open();
```

## ***DrvAAC\_Close***

### **Synopsis**

```
VOID DrvAAC_Close (VOID)
```

### **Description**

This function disables the AAC engine clock.

### **Parameter**

None

### **Return Value**

None

### **Example**

```
DrvAAC_Close ();
```

## ***DrvAAC\_Decoder***

### **Synopsis**

```
INT32  
DrvAAC_Decoder(  
    INT32 i32Size,  
    INT32 *pi32inbuf,  
    INT32 *pi32outbuf  
)
```

### **Description**

Set the parameters for AAC IMDCT of decoder, it will return the size of output buffer and the output buffer for the result of IMDCT.

### **Parameter**

i32Size	2048 or 256
pi32inbuf	The input encoded data.
Pi32outbuf	The output data by running AAC IMDCT of decoder.

### **Return Value**

The size of output buffer. Its bytes is the size x 4.

### **Example**

```
DrvAAC_Open();
```

```
DrvAAC_Decoder(128*2, pi32inptr,pi32resultptr);  
DrvAAC_Close();
```

### ***DrvAAC\_Encoder***

#### **Synopsis**

```
INT32  
DrvAAC_Encoder(  
    INT32 *pi32inbuf,  
    INT32 *pi32outbuf,  
    INT32 i32Size  
)
```

#### **Description**

Set the parameters for AAC MDCT of encoder, it will return the size of output buffer and the output buffer for the result of IMDCT.

#### **Parameter**

pi32inbuf	The input encoded data.
Pi32outbuf	The output data by running AAC MDCT of encoder.
i32Size	2048 or 256

#### **Return Value**

The size of output buffer , its bytes is the size x 4.

#### **Example**

```
DrvAAC_Open();  
DrvAAC_Encoder(pi32inptr,pi32resultptr,256);  
DrvAAC_Close();
```

---

## **1.3. Example code**

This demo code includes sample code and library code. Please refer to AAC sample codes of SDK Non-OS.

## 2. AES Library Introduction

The AES accelerator is a fully compliant implementation of the AES algorithm. Such accelerator supports both encryption and decryption. The AES accelerator can be used in different data security applications, such as secure communications, which need to provide cryptographic protection.

---

### 2.1. Feature

- Supports both encryption and decryption.
- Supports only CBC (Cipher Block Chaining) mode.
- All three kinds of key lengths, 128, 192, and 256 bits, are supported.

---

### 2.2. API Data Structure

#### ***KEYSIZE***

Key size..

Name	Value	Description
<b><i>KEY_128</i></b>	0	128-bit key size
<b><i>KEY_192</i></b>	1	192-bit key size
<b><i>KEY_256</i></b>	2	256-bit key size

---

### 2.3. API Function

#### ***AES\_Initial***

##### Synopsis

```
VOID AES_Initial(VOID);
```

**Description**

Initialize AES engine and install interrupt service routine.

**Parameter**

None

**Return Value**

None

***AES\_Final*****Synopsis**

```
VOID AES_Final(VOID);
```

**Description**

Tear down AES engine.

**Parameter**

None

**Return Value**

None

***AES\_Encrypt*****Synopsis**

```
int AES_Encrypt(UINT8 *input_buf, UINT8 *output_buf, UINT32 input_len, UINT8 *iv,  
UINT8 *key, KEYSIZE key_size);
```

**Description**

Start to encrypt in AES CBC mode and wait for its finish.

**Parameter**

input_buf	4-byte aligned address of input buffer
output_buf	4-byte aligned address of output buffer. If NULL, output_buf = input_buf
input_len	Length of input buffer in bytes
iv	16-byte initialization vector
key	16-, 24-, or 32-byte key buffer
key_size	key size as defined in <b>KEYSIZE</b>

**Return Value**

Success

AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in <a href="#">KEYSIZE</a>
AES_ERR_IV	NULL initialization vector

## ***AES\_Encrypt\_Async***

### Synopsis

```
int AES_Encrypt_Async(UINT8 *input_buf, UINT8 *output_buf, UINT32 input_len, UINT8
*iv, UINT8 *key, KEYSIZE key_size);
```

### Description

Start to encrypt in AES CBC mode but doesn't wait for its finish.

### Parameter

input_buf	4-byte aligned address of input buffer
output_buf	4-byte aligned address of output buffer. If NULL,
output_buf = input_buf	
input_len	Length of input buffer in bytes
iv	16-byte initialization vector
key	16-, 24-, or 32-byte key buffer
key_size	key size as defined in <a href="#">KEYSIZE</a>

### Return Value

Success

AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in <a href="#">KEYSIZE</a>
AES_ERR_IV	NULL initialization vector
AES_ERR_RUNNING	Operation is on-going. Wait by <a href="#">AES_Flush</a> or
Poll by <a href="#">AES_Check_Status</a>	

## ***AES\_Decrypt***

### Synopsis

```
int AES_Decrypt(UINT8 *input_buf, UINT8 *output_buf, UINT32 input_len, UINT8 *iv,
UINT8 *key, KEYSIZE key_size);
```

### Description

Start to decrypt in AES CBC mode and wait for its finish.

**Parameter**

input_buf	4-byte aligned address of input buffer
output_buf	4-byte aligned address of output buffer. If NULL,
output_buf = input_buf	
input_len	Length of input buffer in bytes
iv	16-byte initialization vector
key	16-, 24-, or 32-byte key buffer
key_size	key size as defined in <a href="#">KEYSIZE</a>

**Return Value**

Success	
AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in <a href="#">KEYSIZE</a>
AES_ERR_IV	NULL initialization vector

***AES\_Decrypt\_Async*****Synopsis**

```
int AES_Decrypt_Async(UINT8 *input_buf, UINT8 *output_buf, UINT32 input_len, UINT8
*iv, UINT8 *key, KEYSIZE key_size);
```

**Description**

Start to decrypt in AES CBC mode but doesn't wait for its finish.

**Parameter**

input_buf	4-byte aligned address of input buffer
output_buf	4-byte aligned address of output buffer. If NULL,
output_buf = input_buf	
input_len	Length of input buffer in bytes
iv	16-byte initialization vector
key	16-, 24-, or 32-byte key buffer
key_size	key size as defined in <a href="#">KEYSIZE</a>

**Return Value**

Success	
AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in <a href="#">KEYSIZE</a>

AES_ERR_IV	NULL initialization vector
AES_ERR_RUNNING Poll by <a href="#">AES_Check_Status</a>	Operation is on-going. Wait by <a href="#">AES_Flush</a> or

## ***AES\_Flush***

### **Synopsis**

```
int AES_Flush(VOID);
```

### **Description**

Wait for operation done

### **Parameter**

None

### **Return Value**

Success	Operation done
AES_ERR_BUS_ERROR	Encounter bus error

## ***AES\_Check\_Status***

### **Synopsis**

```
int AES_Check_Status(VOID);
```

### **Description**

Check operation status

### **Parameter**

None

### **Return Value**

Success	Operation done
AES_ERR_BUS_ERROR	Encounter bus error
AES_ERR_BUSY	Operation busy

## ***AES\_Enable\_Interrupt***

### **Synopsis**

```
VOID AES_Enable_Interrupt(VOID);
```

### **Description**

Enable the only interrupt souce

**Parameter**

None

**Return Value**

None

***AES\_Disable\_Interrupt*****Synopsis**

VOID AES\_Disable\_Interrupt(VOID);

**Description**

Disable the only interrupt source

**Parameter**

None

**Return Value**

None

---

## 2.4. Error Code Table

Code Name	Value	Description
Successful	0	Success
AES_ERR_FAIL	AES_ERR_ID   0x01	Internal
AES_ERR_DATA_LEN	AES_ERR_ID   0x02	Not 16-byte aligned
AES_ERR_DATA_BUF	AES_ERR_ID   0x03	NULL buffer
AES_ERR_CIPHER_KEY	AES_ERR_ID   0x04	NULL key or invalid key size
AES_ERR_IV	AES_ERR_ID   0x05	NULL initialization vector
AES_ERR_MODE	AES_ERR_ID   0x06	Internal
AES_ERR_BUS_ERROR	AES_ERR_ID   0x07	Encounter bus error
AES_ERR_RUNNING	AES_ERR_ID   0x08	Operation is on-going. Need to flush.
AES_ERR_BUSY	AES_ERR_ID   0x09	Operation is busy.
AES_ERR_CMPDAT	AES_ERR_ID   0x0A	Internal

## 3. Audio ADC Library Overview

The N9H26 Audio ADC library provides a set of APIs to record audio data from input device. With these APIs, user can set sampling rate. Pre-gain and post gain control if AGC disable, Output target level if AGC enable and so on.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

---

### 3.1. Audio ADC Library APIs Specification

#### *DrvAUR\_Open*

##### Synopsis

```
INT32 DrvAUR_Open(E_AUR_MIC_SEL eMIC, BOOL bIsCoworkEDMA)
```

##### Description

This function is used to open the Audio ADC library.

##### Parameter

eIntType                    Input device type. Please refer [Table 3-1:Input Device](#)

bIsCoworkEDMA            Coporate with EDMA driver to receiver audio data

TRUE: Enable corporation with EDMA.

FALSE: Disable corporation with EDMA

The parameter should be always equal to TRUE.

*Table 3-1:Input Device*

Field name	Value	Description
eAUR_MONO_LINE_IN	0	Mono Line In
eAUR_MONO_MIC_IN	1	Mono MIC In
eAUR_MONO_DIGITAL_MIC_IN	2	Mono Digital MIC In
eAUR_STEREO_DIGITAL_MIC_IN	3	Stereo Digital MIC In

##### Return Value

Successful

#### Example

```
/* Input device is Mono MIC and corporate with EDMA */
DrvAUR_Open(eAUR_MONO_MIC_IN, TRUE);
```

### ***DrvAUR\_Close***

#### Synopsis

```
INT32 DrvAUR_Close(void)
```

#### Description

Close the Audio ADC library.

#### Parameter

None

#### Return Value

Successful

#### Example

```
/* Close Audio ADC library*/
DrvAUR_Close();
```

### ***DrvAUR\_InstallCallback***

#### Synopsis

```
INT32 DrvAUR_InstallCallback ( PFN_AUR_CALLBACK pfnCallback,
                                PFN_AUR_CALLBACK* pfnOldCallback);
```

#### Description

This function is used to install callback function that is used to notice the upper layer for specified audio sample is done. The function will be useless if corporation with EDMA.

The specified audio sample is set in

#### Parameter

pfnCallback      The callback function want to register  
pfnOldCallback   old callback function

#### Return Value

-1 or Successful.

**Example*****DrvAUR\_EnableInt*****Synopsis**

```
void DrvAUR_EnableInt(void);
```

**Description**

This function was used to enable interrupt if converse audio sample done. The function will be useless if corporation with EDMA.

**Parameter**

None

**Return Value**

-1 or Successful.

**Example*****DrvAUR\_DisableInt*****Synopsis**

```
void DrvAUR_DisableInt(void);
```

**Description**

This function was used to disable interrupt if converse audio sample done. The function will be useless if corporation with EDMA.

**Parameter**

None

**Return Value**

-1 or Successful.

**Example**

```
DrvAUR_EnableInt(); /* Enable interrupt if specified audio length done */
```

***DrvAUR\_EnableInt******DrvAUR\_AutoGainCtrl*****Synopsis**

```
INT32 DrvAUR_AutoGainCtrl(BOOL bIsEnable,
```

```
BOOL bIsChangeStep,
E_AUR_AGC_LEVEL eLevel);
```

### Description

This function is used to enable or disable auto gain control-AGC function. And set output target level.

### Parameter

bIsEnable	Enable AGC or not.
bIsChangeStep	To trace the output target level, AGC algorithm change gain for each step
eLevel	Output target level. Please refer <a href="#">Table 3-2:Output Target Level</a>

*Table 3-2:Output Target Level*

Field name	Value	Description
eAUR_OTL_N3	0	-3 db
eAUR_OTL_N4P6	1	-4.6 db
eAUR_OTL_N6P2	2	-6.2 db
eAUR_OTL_N7P8	3	-7.8 db
eAUR_OTL_N9P4	4	-9.4 db
eAUR_OTL_N11	5	-11 db
eAUR_OTL_N12P6	6	-12.6 db
eAUR_OTL_N14P2	7	-14.2 db
eAUR_OTL_N15P8	8	-15.8 db
eAUR_OTL_N17P4	9	-17.4 db
eAUR_OTL_N19	10	-19 db
eAUR_OTL_N20P6	11	-20.6 db
eAUR_OTL_N22P2	12	-22.2 db
eAUR_OTL_N23P8	13	-23.8 db
eAUR_OTL_N25P4	14	-25.4 db
	15	Mute

### Return Value

Successful

### Example

***DrvAUR\_AutoClampingGain*****Synopsis**

```
INT32 DrvAUR_AutoClampingGain ( UINT32 u32MaxGain, UINT32 u32MinGain)
```

**Description**

This function was used to clamp the maximum and minimum gain if enable AGC function. It will be useless if disable AGC.

**Parameter**

- |            |   |
|------------|---|
| u32MaxGain | Maximun gain to clamp AGC. The value is from 0 ~15. |
| u32MinGain | Minimum gain to clamp AGC. The value is from 0 ~15. |

**Return Value**

Successful

**Example*****DrvAUR\_SetSampleRate*****Synopsis**

```
INT32 DrvAUR_SetSampleRate(E_AUR_SPS eSampleRate)
```

**Description**

This function is used to set sampling rate.

**Parameter**

- |             |  |
|-------------|--|
| eSampleRate | Sampling rate from 8K to 192K. Please refer <a href="#">Table 3-3: Sampling Rate</a> |
|-------------|--|

*Table 3-3: Sampling Rate*

Field name	Value	Description
eAUR_SPS_48000	48000	48K sampling rate
eAUR_SPS_44100	44100	44.1K sampling rate
eAUR_SPS_32000	32000	32K sampling rate
eAUR_SPS_24000	24000	24K sampling rate
eAUR_SPS_22050	22050	22K sampling rate
eAUR_SPS_16000	16000	16K sampling rate
eAUR_SPS_12000	12000	12K sampling rate
eAUR_SPS_11025	11025	11.025K sampling rate
eAUR_SPS_8000	8000	8K sampling rate

eAUR_SPS_96000	96000	96K sampling rate
eAUR_SPS_192000	192000	192K sampling rate

**Return Value**

Successful

**Example*****DrvAUR\_AudioI2cRead*****Synopsis**

INT32 DrvAUR\_AudioI2cRead(UINT32 u32Addr, UINT8\* p8Data)

**Description**

This function is used to read back the internal register of sigma-delta ADC. Programmer can use the API to adjust pre-gain and post-gain if AGC is disable.

**Parameter**

u32Addr	Register address. Please refer <a href="#">Table 3-4: Sigma-Delta Register Address</a> .
p8Data	Register content after read back.

*Table 3-4: Sigma-Delta Register Address*

Field name	Value	Description
eAUR_ADC_H20	0x20	Please refer IP programming guide
eAUR_ADC_H21	0x21	Please refer IP programming guide
eAUR_ADC_H22	0x22	Please refer IP programming guide
eAUR_ADC_H23	0x23	Please refer IP programming guide
eAUR_ADC_H24	0x24	Please refer IP programming guide
eAUR_ADC_H25	0x25	Please refer IP programming guide
eAUR_ADC_H26	0x26	Please refer IP programming guide
eAUR_ADC_H29	0x29	Please refer IP programming guide

**Return Value**

Successful

**Example**

```
DrvAUR_AudioI2cWrite(0x22, 0x1E); /* Adjust Pre-gain */
DrvAUR_AudioI2cWrite(0x23, 0x0E); /* Adjust Post-gain*/
```

***DrvAUR\_AudioI2cWrite*****Synopsis**

```
INT32 DrvAUR_AudioI2cWrite(UINT32 u32Addr, UINT32 u32Data);
```

**Description**

This function is used to program the internal register of sigma-delta ADC. Programmer can use the API to adjust pre-gain and post-gain if AGC is disable.

**Parameter**

u32Addr	Register address. Please refer <a href="#">Table 3-4: Sigma-Delta Register Address</a> .
u32Data	The content want to program sigma-delta

**Return Value**

Successful

**Example**

```
DrvAUR_AudioI2cWrite(0x22, 0x1E); /* Adjust Pre-gain */
DrvAUR_AudioI2cWrite(0x23, 0x0E); /* Adjust Post-gain */
```

***DrvAUR\_SetDigiMicGain*****Synopsis**

```
VOID DrvAUR_SetDigiMicGain(BOOL bIsEnable,
                           E_AUR_DIGI_MIC_GAIN eDigiGain)
```

**Description**

This function is used to set digital gain if input device is digital MIC. It is only for input device is Mono Digital MIC In or Stereo Digital MIC In. Please refer [Table 3-1:Input Device](#)

**Parameter**

bIsEnable	Enable digital gain for digital MIC.
eDigiGain	Digital gain. Please refer <a href="#">Table 3-5:Digital Gain</a>

*Table 3-5:Digital Gain*

Field name	Value	Description
eAUR_DIGI_MIC_GAIN_P0	0	+0db
eAUR_DIGI_MIC_GAIN_P1P6	1	+1.6db
eAUR_DIGI_MIC_GAIN_P3P2	2	+3.2db
eAUR_DIGI_MIC_GAIN_P4P8	3	+4.8db
eAUR_DIGI_MIC_GAIN_P6P4	4	+6.4db

eAUR_DIGI_MIC_GAIN_P8	5	+8db
eAUR_DIGI_MIC_GAIN_P9P6	6	+9.6db
eAUR_DIGI_MIC_GAIN_P11P2	7	+11.2db
eAUR_DIGI_MIC_GAIN_P12P8	8	+12.8db
eAUR_DIGI_MIC_GAIN_P14P4	9	+14.4db
eAUR_DIGI_MIC_GAIN_P16	10	+16db
eAUR_DIGI_MIC_GAIN_P17P6	11	+17.6db
eAUR_DIGI_MIC_GAIN_P19P2	12	+19.2db
eAUR_DIGI_MIC_GAIN_P20P8	13	+20.8db
eAUR_DIGI_MIC_GAIN_P22P4	14	+22.4db
eAUR_DIGI_MIC_GAIN_P24	15	+24db

**Return Value**

Successful

**Example**

```
DrvAUR_SetDigiMicGain(TRUE, eAUR_DIGI_MIC_GAIN_P19P2);
```

***DrvAUR\_StartRecord*****Synopsis**

```
VOID DrvAUR_StartRecord(E_AUR_MODE eMode);
```

**Description**

Start up sigma-delta ADC to converse audio data.

**Parameter**

eMode            Only eAUR\_MODE\_1 can be set if corporate with EDMA

Please refer [Table 3-6:Interface Between Audio ADC and EDMA](#)

*Table 3-6:Interface Between Audio ADC and EDMA*

Field name	Value	Description
eAUR_MODE_0	0	1 sample
eAUR_MODE_1	1	2 Samples
eAUR_MODE_2	2	4 Samples
eAUR_MODE_3	3	8 Samples

**Return Value**

Successful

**Example**

```
DrvAUR_StartRecord(eAUR_MODE_1);
```

***DrvAUR\_StopRecord*****Synopsis**

```
VOID DrvAUR_StopRecord(void);
```

**Description**

Stop record

**Parameter**

None

**Return Value**

Successful

**Example**

```
DrvAUR_StopRecord();
```

***DrvAUR\_SetDataOrder*****Synopsis**

```
VOID DrvAUR_SetDataOrder(E_AUR_ORDER eOrder)
```

**Description**

This function is used to set the PCM data order for each audio sample

**Parameter**

eOrder           PCM data format. Please refer [Table 3-7: Supported PCM Data Format](#)

*Table 3-7: Supported PCM Data Format*

Field name	Value	Description
eAUR_ORDER_MONO_32BITS	0	Mono little endian 32 bits signed PCM
eAUR_ORDER_MONO_16BITS	1	Mono little endian 16 bits signed PCM
eAUR_ORDER_STEREO_16BITS	2	Stereo little endian 16 bits signed PCM

eAUR\_ORDER\_MONO\_24BITS

3

(Non-standard 24 bits PCM)

**Return Value**

Successful

**Example**

```
DrvAUR_Open(eMicType, TRUE);

DrvAUR_SetDataOrder(eAUR_ORDER_MONO_16BITS);

if(eMicType == eAUR_MONO_MIC_IN){

    DrvAUR_AudioI2cWrite(0x22, 0x1E); /* Adjust Pre-gain */

    DrvAUR_AudioI2cWrite(0x23, 0x0E); /* Adjust Post-gain*/

    DrvAUR_DisableInt();

    DrvAUR_SetSampleRate(aArraySampleRate[i32Idx]);

    DrvAUR_AutoGainTiming(1,1,1);

    DrvAUR_AutoGainCtrl(TRUE, TRUE, eAUR_OTL_N12P6);

} else if((eMicType == eAUR_MONO_DIGITAL_MIC_IN) ||
          (eMicType == eAUR_STEREO_DIGITAL_MIC_IN)) {

    DrvAUR_SetDigiMicGain(TRUE, eAUR_DIGI_MIC_GAIN_P19P2);

    DrvAUR_DisableInt();

    DrvAUR_SetSampleRate(eAUR_DIGI_MIC_GAIN_P19P2);

}
```

# 4. AVI Library Overview

## 4.1. Video render

FA9x/VA9x can support JPEG decoder to output decoded packet data in DIRECT\_RGB555, DIRECT\_RGB565, DIRECT\_RGB888 or DIRECT\_YUV422 format. User application must initialize VPOST as corresponding format specified in AVI function call `aviPlayFile(...)`. AVI player library will configure JPEG output format as specified format and use DMA to copy the decoded data to VPOST frame buffer in Vsync period to avoid the tearing issue.  
In this way, three frame buffers are required. One is allocated in VPOST initialized function and two buffers are allocated in AVI library.

## 4.2. How to use AVI player library

The AVI player library has managed the file access, JPEG decode and audio decode. User only gives the AVI file name and render method to play the movie. The AVI player required user to prepare the following things before playing an AVI movie:

- Initialize system with cache on
- Initialize file system and storage interface (ex. SD card)
- Initialize timer 0
- Initialize VPOST

The VPOST frame buffer format should be consistent with the AVI playback render mode:

- Direct RGB555 – VPOST should select ***DRVVPPOST\_FRAME\_RGB555***
- Direct RGB565 – VPOST should select ***DRVVPPOST\_FRAME\_RGB565***
- Direct RGB888 – VPOST should select ***DRVVPPOST\_FRAME\_RGBx888 or DRVVPPOST\_FRAME\_RGB88x***
- Direct YUV422 – VPOST should select ***DRVVPPOST\_FRAME\_CBYCRY or DRVVPPOST\_FRAME\_YCBYCR or DRVVPPOST\_FRAME\_CRYCBY or DRVVPPOST\_FRAME\_YCRYCB***

Currently, if the decoded Video size is less then the panel size, it will be located at the center of panel. Moreover, decoded image scales by 1/2 in horizontal and vertical direction if the decoded video width is larger than the panel width.

The AVI playback function does not support (x, y) coordinate that are the second and third argument of `aviPlayFile()` used to specify the render location on LCD now.

## 4.3. AVI player user callback

While playing an AVI movie, user application may want to draw information on screen or manage user inputs. AVI library provides a callback function to allow user application to grab pieces of CPU time. The callback function pointer was passed to AVI player as the last argument of *aviPlayFile()*.

Depends on the loading of playing an AVI movie, the user callback will be called several times in each one second. User application should finish the execution of callback function as soon as possible. Otherwise, the AVI playback can be broken because of not enough CPU time.

---

## 4.4. AVI playback information

While playing an AVI movie, user application can get AVI file information and playback progress information from AVI player. The AVI information will be passed to user application as a parameter of callback function. All information is packed in the AVI\_INFO\_T structure.

## 4.5. API Enumeration

Name	Value	Description
<b>JV_MODE_E</b>		
DIRECT_RGB555	0x0	Direct RGB555 output format
DIRECT_RGB565	0x1	Direct RGB565 output format
DIRECT_RGB888	0x2	Direct RGB888 output format
DIRECT_YUV422	0x3	Direct YUV422 output format
<b>AU_TYPE_E</b>		
AU_CODEC_UNKNOWN	0x0	Unknown audio format
AU_CODEC_PCM	0x1	PCM audio format
AU_CODEC_IMA_ADPCM	0x2	ADPCM audio format
AU_CODEC_MP3	0x3	MP3 audio format

## 4.6. API Structure

*Figure 4-1 AVI\_INFO\_T structure*

Field	Type	Description
uMovieLength	UINT32	The total length of input AVI movie (in 0.01 second unit)
uPlayCurTimePos	UINT32	The current playback position. (in 0.01 second unit)
eAuCodec	AU_TYPE_E	Audio format type
nAuPlayChnNum	INT	Audio channel number. (1: mono, 2: stereo, 0: video-only)
nAuPlaySRate	INT	audio sampling rate
uVideoFrameRate	UINT32	Video frame rate.
usImageWidth	UINT16	Video image width
usImageHeight	UINT16	Video image height
uVidTotalFrames	UINT32	total number of video frames
uVidFramesPlayed	UINT32	Indicate how many video frames have been played
uVidFramesSkipped	UINT32	The number of frames was skipped. Video frames may be skipped due to A/V sync

## 4.7. Functions

### ***aviStopPlayFile***

#### Synopsis

```
int
aviStopPlayFile(void);
```

#### Description

Stop current AVI file playback.

#### Parameter

None

**Return Value**

Successful: Success

ERRCODE: Error

**Example**

None.

***aviPlayFile*****Synopsis**

```
int  
aviPlayFile(  
    char *suFileName,  
    int x,  
    int y,  
    JV_MODE_E mode,  
    AVI_CB *cb  
) ;
```

**Description**

Play an AVI file.

**Parameter****suFileName [in]**

The full path file name of input AVI file.

**x [in]**

The left-up corner x-coordinate of AVI video render area. Not used now.

**y [in]**

The left-up corner y-coordinate of AVI video render area. Not used now.

**mode [in]**

Video render mode.

**cb [in]**

User application callback function.

**Return Value**

Successful: Success

ERRCODE: Error

### Example

```
/*-----*/
/* Direct RGB565 AVI playback !! */
/*-----*/

lcdformatex.ucVASrcFormat = DRVVPPOST_FRAME_RGB565;

vpostLCMInit(&lcdformatex, (UINT32 *)_VpostFrameBuffer);

fsAsciiToUnicode("c:\\Flip-20fps_640x480.avi", suFileName, TRUE);

aviPlayFile(suFileName, 0, 0, DIRECT_RGB565, avi_play_control);
```

## **aviGetFileInfo**

### Synopsis

```
int
aviGetFileInfo (
    char *suFileName,
    AVI_INFO_T *ptAviInfo
);
```

### Description

Get the AVI file information.

### Parameter

#### **suFileName [in]**

The full path file name of input AVI file.

#### **ptAviInfo [in]**

Return AVI parsing information.

### Return Value

Successful: Success

ERRCODE: Error

### Example

```
fsAsciiToUnicode("c:\\Flip-20fps.avi", suFileName, TRUE);
```

```
aviPlayFile(suFileName, &sAVIInfo);
```

## ***aviSetPlayVolume***

### **Synopsis**

```
int  
aviSetPlayVolume (  
    int vol  
) ;
```

### **Description**

Set the Left channel and Right channel playback audio volume.

### **Parameter**

#### **vol [in]**

The audio volume

### **Return Value**

Successful: Success

ERRCODE: Error

### **Example**

```
aviSetPlayVolume(suFileName, 0x1F);
```

## ***aviSetRightChannelVolume***

### **Synopsis**

```
int  
aviSetRightChannelVolume (  
    int vol  
) ;
```

### **Description**

Set the Right channel audio playback volume only.

### **Parameter**

#### **vol [in]**

The audio volume

#### Return Value

Successful: Success

ERRCODE: Error

#### Example

```
// Set Right Channel as Mute
aviSetPlayRightChannelVolume(suFileName, 0x0);
```

---

## 4.8. Error Code Table

Code Name	Value	Description
MFL_ERR_NO_MEMORY	0xFFFF8000	no memory
MFL_ERR_HARDWARE	0xFFFF8002	hardware general error
MFL_ERR_NO_CALLBACK	0xFFFF8004	must provide callback function
MFL_ERR_AU_UNSUPPORT	0xFFFF8006	not supported audio type
MFL_ERR_VID_UNSUPPORT	0xFFFF8008	not supported video type
MFL_ERR_OP_UNSUPPORT	0xFFFF800C	unsupported operation
MFL_ERR_PREV_UNSUPPORT	0xFFFF800E	preview of this media type was not supported or not enabled
MFL_ERR_FUN_USAGE	0xFFFF8010	incorrect function call parameter
MFL_ERR_RESOURCE_MEM	0xFFFF8012	memory is not enough to play/record a media file
MFL_ERR_FILE_OPEN	0xFFFF8020	cannot open file
MFL_ERR_FILE_TEMP	0xFFFF8022	temporary file access failure
MFL_ERR_STREAM_IO	0xFFFF8024	stream access error
MFL_ERR_STREAM_INIT	0xFFFF8026	stream was not opened
MFL_ERR_STREAM_EOF	0xFFFF8028	encounter EOF of file
MFL_ERR_STREAM_SEEK	0xFFFF802A	stream seek error
MFL_ERR_STREAM_TYPE	0xFFFF802C	incorrect stream type
MFL_ERR_STREAM_METHOD	0xFFFF8030	missing stream method
MFL_ERR_STREAM_MEMOUT	0xFFFF8032	recorded data has been over the application provided memory buffer
MFL_INVALID_BITSTREAM	0xFFFF8034	invalid audio/video bitstream forma
MFL_ERR_AVI_FILE	0xFFFF8080	Invalid AVI file format
MFL_ERR_AVI_VID_CODEC	0xFFFF8081	AVI unsupported video codec type
MFL_ERR_AVI_AU_CODEC	0xFFFF8082	AVI unsupported audio codec type

MFL_ERR_AVI_CANNOT_SEEK	0xFFFF8083	The AVI file is not fast-seekable
MFL_ERR_AVI_SIZE	0xFFFF8080	Exceed estimated size
MFL_ERR_MP3_FORMAT	0xFFFF80D0	incorrect MP3 frame format
MFL_ERR_MP3_DECODE	0xFFFF80D2	MP3 decode error
MFL_ERR_HW_NOT_READY	0xFFFF8100	the picture is the same as the last one
MFL_ERR_SHORT_BUFF	0xFFFF8104	buffer size is not enough
MFL_ERR_VID_DEC_ERR	0xFFFF8106	video decode error
MFL_ERR_VID_DEC_BUSY	0xFFFF8108	video decoder is busy
MFL_ERR_VID_ENC_ERR	0xFFFF810A	video encode error
MFL_ERR_UNKNOWN_MEDIA	0xFFFF81E2	unknow media type

---

## 4.9. MP3 Library Overview

Support MP3 sampling rate 8000 Hz, 11025 Hz, 16000 Hz, 22050 Hz, 32000 Hz, 44100 Hz and 48000 Hz.

### ***How to use MP3 player library***

- Init cache.
- Init UART.
- Init timer.
- Init filesystem.
- Init storage device.
- Init audio device.
- Start play MP3 file.

### ***MP3 player user callback***

”ap\_time”, the member of structure MV\_CFG\_T can execute user defined API. Any time information or control can be handled in it.

### ***MP3 player information***

Structure MV\_INFO\_T will give you the time information, include current time and total time.

---

## 4.10. API Enumeration

Name	Value	Description
MEDIA_TYPE_E		
MFL_MEDIA_MP3	0x5	MP3 audio format
STRM_TYPE_E		
MFL_STREAM_FILE	0x1	MP3 file
PLAY_CTRL_E		
PLAY_CTRL_STOP	0x5	Stop playback

---

## 4.11. Structure

*Table 4-1 :MV\_CFG\_T structure*

Field	Type	Description
eInMediaType	MEDIA_TYPE_E	PLAY - indicate the type of media to be played
eInStrmType	STRM_TYPE_E	PLAY - indicate the input stream method
szIMFAscii	CHAR *	PLAY - if in stream type is MFL_STREAM_FILE
suInMetaFile	CHAR *	PLAY - if in stream type is MFL_STREAM_FILE
szITFAscii	CHAR *	PLAY - if in stream type is MFL_STREAM_FILE
nAudioPlayVolume	INT	PLAY - volume of playback, 0~31, 31 is max.
uStartPlaytimePos	INT	PLAY - On MP3 playback start, just jump to a specific time offset then start playback. The time position unit is 1/100 seconds.
nAuABRScanFrameCnt	INT	PLAY - on playback, ask MFL scan how many leading frames to evaluate average bit rate. -1 means scan the whole file
ap_time	callback	callback
suInMediaFile	CHAR *	PLAY - if in stream type is MFL_STREAM_FILE

*Table 4-2: MV\_INFO\_T structure*

Field	Type	Description
uAuTotalFrames	UINT32	For playback, it's the total number of audio frames. For recording, it's the currently recorded frame number.
uPlayCurTimePos	UINT32	for playback, the play time position, in 1/100 seconds
uMovieLength	UINT32	in 1/100 seconds

---

## 4.12. Functions

### *mflMediaPlayer*

#### Synopsis

```
INT mflMediaPlayer(MV_CFG_T *ptMvCfg)
```

**Description**

Start play MP3 file.

**Parameter**

**ptMvCfg [in]**

The MV\_CFG\_T structure

**Return Value**

Successful: Success

ERRCODE: Error

**Example**

```
mflMediaPlayer(&_tMvCfg)
```

```
mflGetMovieInfo
```

***mflGetMovieInfo*****Synopsis**

```
INT mflGetMovieInfo(MV_CFG_T *ptMvCfg, MV_INFO_T **ptMvInfo)
```

**Description**

Get MP3 time information.

**Parameter**

**ptMvCfg [in]**

The MV\_CFG\_T structure

**ptMvInfo [in/out]**

The MV\_INFO\_T structure

**Return Value**

Successful: Success

ERRCODE: Error

**Example**

```
mflGetMovieInfo(ptMvCfg, &ptMvInfo)
```

mflPlayControl

### ***mflPlayControl***

#### **Synopsis**

```
INT mflPlayControl(MV_CFG_T *ptMvCfg, PLAY_CTRL_E ePlayCtrl, INT nParam)
```

#### **Description**

Control operation while playing MP3 file.

#### **Parameter**

##### **ptMvCfg [in]**

The MV\_CFG\_T structure

##### **ePlayCtrl [in]**

The PLAY\_CRTL\_E enumeration

##### **nParam [in]**

Reserved

#### **Return Value**

Successful: Success

ERRCODE: Error

#### **Example**

```
mflPlayControl(&_tMvCfg, PLAY_CTRL_STOP, 0)
```

# 5. BLT Library Introduction

This document is written for user applications which want to make use of BLT through provided API.

## 5.1. Feature

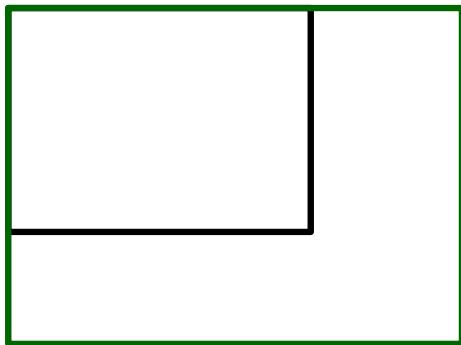
- Fill operation.
  - Fill color with alpha channel
- Blit operation
  - Transformation effects (Scaling, Rotation, Shearing, etc.) through 2x2 inverse transformation matrix.
  - Bitmap smoothing in bi-linear algorithm.
  - Tiling mode (for inversely mapped source pixels lying outside the boundaries of the source image)
    - ◆ No drawing
    - ◆ Clip to edge (closest edge pixel of the source image)
    - ◆ Repeat (source image repeated indefinitely in all directions)
  - Color transformation as defined in Adobe Flash
  - RGB565 color key
- Source format for Blit operation
  - ARGB8888
  - RGB565
  - Palette index with color ARGB8888
    - ◆ 1-bit, 2-bit, 4-bit, and 8-bit palette index
    - ◆ Endianness of palette index
- Destination format for Fill/Blit operation
  - ARGB8888
  - RGB555
  - RGB565

## 5.2. Transformation Matrix

In blit operation, transformation effects, such as scaling, rotation, translation, etc. can be achieved through a transformation matrix. These transformations can be combined into one and just one blit operation is needed to finish all the transformations. In the following, common transformations are listed, and user application can combine them to achieve wanted result.

### Scaling

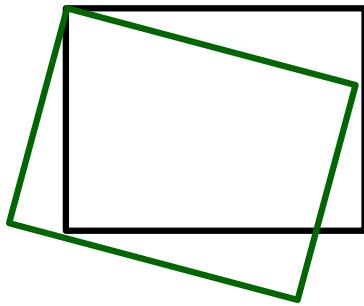
Resize the image by multiplying the location of each pixel by  $s_x$  on the x axis and  $s_y$  on the y axis.



$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$$

### Rotation

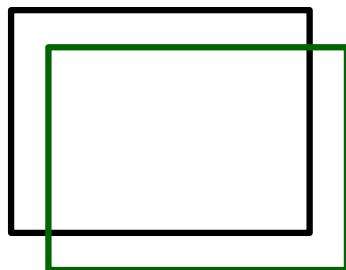
Rotate the image by an angle  $\theta$  clockwise.



$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$$

### Translation

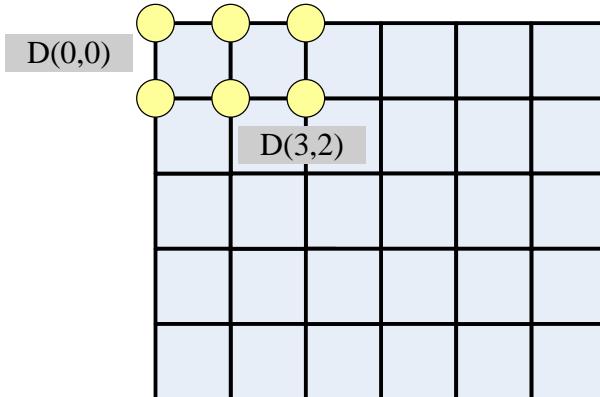
Translate the image by  $t_x$  along the x axis and  $t_y$  along the y axis.



$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$$

### 5.3. Amendment to User Transformation Matrix

On mapping back from destination CS<sup>1</sup> to source CS, a mapping point of destination pixel must be taken into consideration. Mapping point can be top-left ([Top-left point as mapping point of destination pixel](#)) or center ([Center point as mapping point of destination pixel](#)). In the blit implementation, top-left point is chosen and we may encounter an error due to the choice of mapping point. To help explain the issue, an example is given: blit with rotate 180°.



*Figure 5-1: Top-left point as mapping point of destination pixel*

<sup>1</sup> Coordinate system.

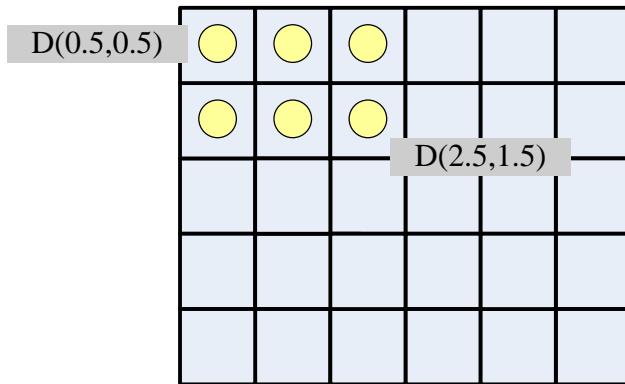


Figure 5-2: Center point as mapping point of destination pixel

1. We want to blit **Source image** and get **Final** result, This blit operation involves rotation and translation applied to the source image.

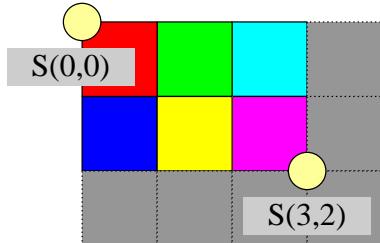


Figure 5-3: Source image

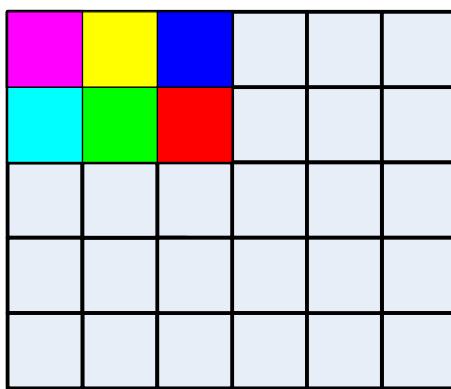


Figure 5-4: Final

2. First, just copy without any transformation effect and get **No transform** result.

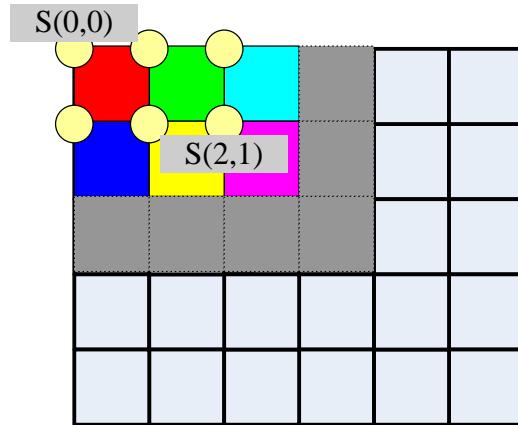


Figure 5-5: No transform

3. Rotate 180° clockwise and get [Rotate 180°](#) result.

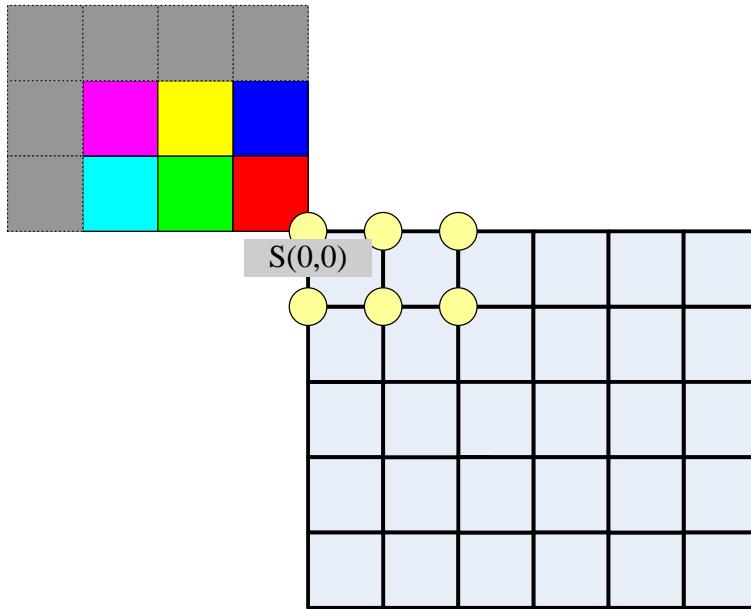


Figure 5-6: Rotate 180°

4. Translate 3 along x-axis and 2 along y-axis and get [Rotate 180° + Translate \(3, 2\)](#) result. But actually, we will get incorrect [Rotate 180° + Translate \(3, 2\) \(2\)](#) result. It is because in the hardware implementation, the top-left point of a destination pixel is picked as the mapping point. Take D(0, 0) as an example. It will map to S(3, 2) instead of S(2, 1) which is actually what we want.

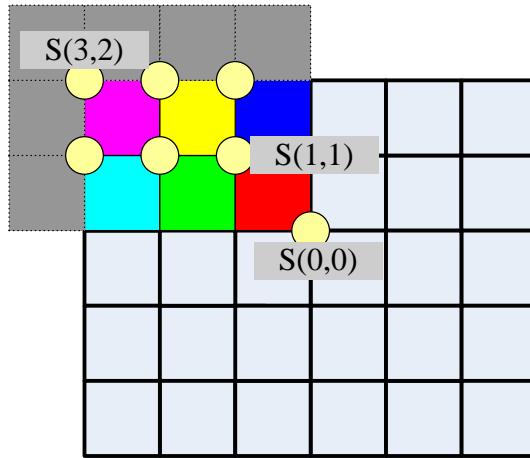


Figure 5-7: Rotate 180° + Translate (3, 2)

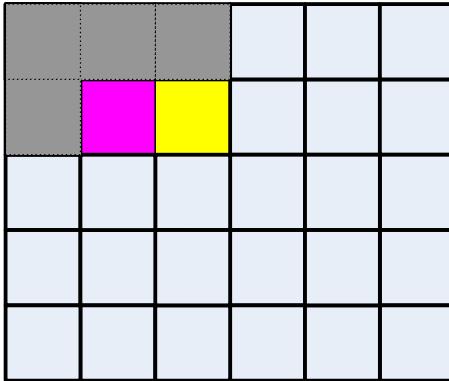


Figure 5-8: Rotate 180° + Translate (3, 2) (2)

5. To fix the issue, translate (-0.5, -0.5) to the end of all above transformations, and get [Rotate 180° + Translate \(3, 2\) + Translate \(-0.5, -0.5\)](#) result. And we finally get wanted [Final](#) result. In this case, D(0, 0) maps to S(2.5, 1.5), and so the source (2, 1) pixel ([magenta](#)) is blitted on the destination (0, 0) pixel.

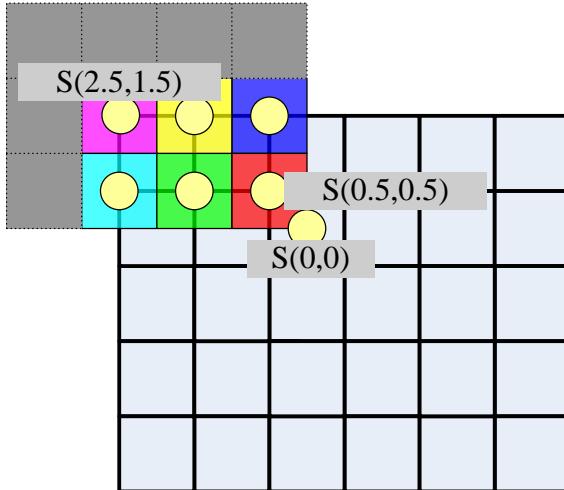


Figure 5-9: Rotate 180° + Translate (3, 2) + Translate (-0.5, -0.5)

## 5.4. Pixel Mapping

To use blit operation, think of pixel mapping in the inverse direction, that is, from destination CS to source CS. Fields associated with transformation matrix include:

- Elements a, b, c, and d in [S\\_DRVBLT\\_MATRIX](#).
- i32XOffset and i32YOffset in [S\\_DRVBLT\\_SRC\\_IMAGE](#).

Equations below give how these fields are associated with transformation matrix.

$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} s & t & t_x \\ u & v & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} s & t & t_x \\ u & v & t_y \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$$

$$\begin{vmatrix} a & c & i32XOffset \\ b & d & i32YOffset \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} s & t & t_x \\ u & v & t_y \\ 0 & 0 & 1 \end{vmatrix}^{-1}$$

When a point is mapped from destination CS to source CS, there are several cases to consider. Below gives an example to help explain:

$$\begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix}$$

1. In M0, D(0, 0) (origin pixel of destination CS) is inversely mapped to S(1, 1), which needn't be the origin pixel of the source image. D(0, 0) is filled with **Red** color.
2. In M1, D(1, 1) is inversely mapped to S(2, 2), which lies inside the source image. D(1, 1) is filled with **Green** color.
3. In M2, D(4, 2) is inversely mapped to S(5, 3), which lies outside the source image. Dependent on tiling mode specified in [E\\_DRVBLT\\_FILL\\_STYLE](#), there are 3 different rendering results:
  - 甲、No drawing: D(4, 2) is not drawn.
  - 乙、Clip to edge: D(4, 2) is inversely mapped to S(3, 3). D(4, 2) is filled with **Blue** color.
  - 丙、Repeat: Think of whole source CS as filled with source images and D(4, 2) is inversely mapped to S(5, 3), and then wraps to S(1, 3). D(4, 2) is filled with **Yellow** color.

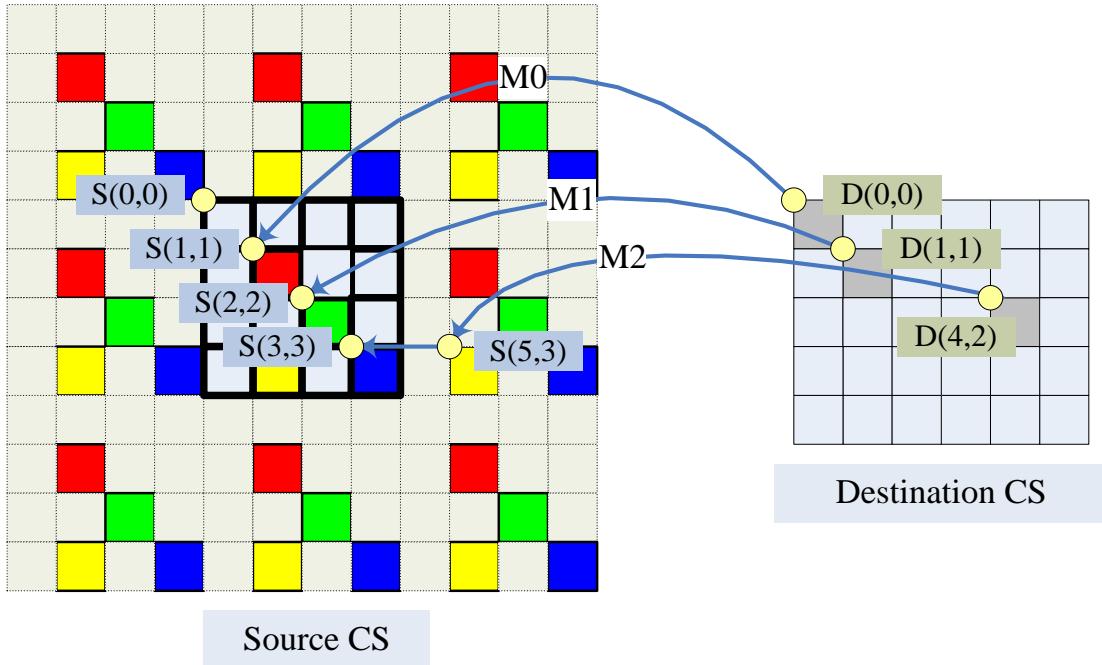


Figure 5-10: Mapping from destination CS to source CS

## 5.5. Color Transformation

In Blit operation, user application can decide to apply color transformation or not, which is defined by Adobe Flash and has the following formula. Besides, user application can further decide to apply the alpha channel only.

```

New alpha value = (old alpha value * alphaMultiplier) + alphaOffset
New red value = (old red value * redMultiplier) + redOffset
New green value = (old green value * greenMultiplier) + greenOffset
New blue value = (old blue value * blueMultiplier) + blueOffset

```

## 5.6. Palette

To use BLT palette, user must choose index size first. There are 4 index sizes (SFMT) supported:

- 1-bit index
- 2-bit index
- 4-bit index
- 8-bit index

After determination of index size, user then must set up two parts: palette entries and source image in palette index format, both of which will depend on index size.

### **Palette entries**

Palette ranges from BLT\_BA+0x400. Its format is ARGB8888, premultiplied-alpha by default and user can change it by setting up the field SET2DA.S\_ALPHA.

For n-bit index size where n can only be 1, 2, 4, or 8, user must prepare 2 to the power of n (2, 4, 16, or 256 respectively) palette entries. Take n=2 for example, user must fill in 4 palette entries in the range, BLT\_BA+0x400~BLT\_BA+0x400+3 words.

### **Source image in palette index format**

To specify source image in palette index format is the same as in other formats: pixel format (SFMT), start address (SADDR), width (SWIDTH), height (SEIGHT), stride (SSTRIDE). Note stride must be word-aligned. If palette index is not 8-bit, index order in one byte image data must be taken into consideration.

For example,

One byte in image data=b7b6b5b4b3b2b1b0

Index size=2-bit

Index order=big-endian (SET2DA.L\_ENDIAN=0) →

b7b6=1<sup>st</sup> pixel, b5b4=2<sup>nd</sup> pixel, b3b2=3<sup>rd</sup> pixel, b1b0=4<sup>th</sup> pixel

Index order=little-endian (SET2DA.L\_ENDIAN=1) →

b7b6=4<sup>th</sup> pixel, b5b4=3<sup>rd</sup> pixel, b3b2=2<sup>nd</sup> pixel, b1b0=1<sup>st</sup> pixel

## **5.7. API Data Structure**

### **E\_BLT\_INT\_TYPE**

Interrupt type.

Name	Value	Description
<i>BLT_INT_CMPLT</i>	1	Fill/Blit operation completed

### **E\_DRVBLT\_FILLOP**

Fill or Blit operation.

Name	Value	Description
<i>eDRVBLT_DISABLE</i>	0	Blit operation
<i>eDRVBLT_ENABLE</i>	1	Fill operation

***E\_DRVBLT\_REVEAL\_ALPHA***

Premultiplied alpha or not for source format of ARGB8888

Name	Value	Description
<i>eDRVBLT_EFFECTIVE</i>	0	Premultiplied alpha
<i>eDRVBLT_NO_EFFECTIVE</i>	1	Non-premultiplied alpha

***E\_DRVBLT\_TRANSFORM\_FLAG***

Transform flags for Blit operation.

Color transformation formula applied when eDRVBLT\_HASCOLORTRANSFORM specified:

New alpha value = (old alpha value \* alphaMultiplier) + alphaOffset

New red value = (old red value \* redMultiplier) + redOffset

New green value = (old green value \* greenMultiplier) + greenOffset

New blue value = (old blue value \* blueMultiplier) + blueOffset

Alpha-only color transformation formula applied when both eDRVBLT\_HASCOLORTRANSFORM and eDRVBLT\_HASALPHAONLY specified:

New alpha value = (old alpha value \* alphaMultiplier) + alphaOffset

Name	Value	Description
<i>eDRVBLT_NONTRANSPARENCYE</i>	0	No per-pixel transparency in the source.
<i>eDRVBLT_HASTRANSPARENCY</i>	1	Has per-pixel transparency in the source.
<i>eDRVBLT_HASCOLORTRANSFORM</i>	2	Apply color transformation formula.
<i>eDRVBLT_HASALPHAONLY</i>	4	If color transformation enabled, just apply the alpha-only formula.

***E\_DRVBLT\_BMPPIXEL\_FORMAT***

Source format for Blit operation.

If eDRVBLT\_SRC\_ARGB8888/palette index, source/palette color can be RGB888 or ARGB8888 dependent on [\*E\\_DRVBLT\\_TRANSFORM\\_FLAG\*](#).

Name	Value	Description
<i>eDRVBLT_SRC_ARGB8888</i>	1	RGB888/ARGB8888
<i>eDRVBLT_SRC_RGB565</i>	2	RGB565
<i>eDRVBLT_SRC_1BPP</i>	4	1-bit palette index
<i>eDRVBLT_SRC_2BPP</i>	8	2-bit palette index
<i>eDRVBLT_SRC_4BPP</i>	16	4-bit palette index

<i>eDRVBLT_SRC_8BPP</i>	32	8-bit palette index
-------------------------	----	---------------------

***E\_DRVBLT\_DISPLAY\_FORMAT***

Destination format for Fill/Blit operation.

Name	Value	Description
<i>eDRVBLT_DEST_ARGB8888</i>	1	ARGB8888
<i>eDRVBLT_DEST_RGB565</i>	2	RGB565
<i>eDRVBLT_DEST_RGB555</i>	4	RGB555

***E\_DRVBLT\_FILL\_STYLE***

Other flags for Blit operation.

*eDRVBLT\_CLIP\_TO\_EDGE*/*eDRVBLT\_NONE\_FILL* specify how to behave when reverse mapping doesn't fall in the range of source bitmap.

Name	Value	Description
<i>eDRVBLT_CLIP_TO_EDGE</i>	1	The bitmap should be clipped to its edges, otherwise a repeating texture.
<i>eDRVBLT_NOTSMOOTH</i>	2	The bitmap should not be smoothed
<i>eDRVBLT_NONE_FILL</i>	4	Neither clip to edge nor repeating texture

***E\_DRVBLT\_PALETTE\_ORDER***

Palette index in big-endian or little-endian.

Name	Value	Description
<i>eDRVBLT_BIG_ENDIAN</i>	0	Palette index in big endian
<i>eDRVBLT_LITTLE_ENDIAN</i>	1	Palette index in little endian

***S\_DRVBLT\_MATRIX***

Transformation matrix used in inverse mapping..

Name	Type	Description
<i>a</i>	INT32	
<i>b</i>	INT32	
<i>c</i>	INT32	

<i>d</i>	INT32	
----------	-------	--

**S\_DRVBLT\_ARGB16**

Multiplier/offset of A, R, G, and B channels used in color transformation.

Name	Type	Description
<i>i16Blue</i>	INT16	Color multiplier/offset of blue channel
<i>i16Green</i>	INT16	Color multiplier/offset of green channel
<i>i16Red</i>	INT16	Color multiplier/offset of red channel
<i>i16Alpha</i>	INT16	Color multiplier/offset of alpha channel

**S\_DRVBLT\_ARGB8**

ARGB8888 color

Name	Type	Description
<i>u8Blue</i>	UINT8	Value of blue channel
<i>u8Green</i>	UINT8	Value of green channel
<i>u8Red</i>	UINT8	Value of red channel
<i>u8Alpha</i>	UINT8	Value of alpha channel

**S\_DRVBLT\_SRC\_IMAGE**

Source image.

Name	Type	Description
<i>u32SrcImageAddr</i>	UINT32	Source image start address
<i>i32Stride</i>	INT32	Source image's stride in bytes
<i>i32XOffset</i>	INT32	X offset into the source to start rendering from
<i>i32YOffset</i>	INT32	Y offset into the source to start rendering from
<i>i16Width</i>	INT16	Source image's width in pixels
<i>i16Height</i>	INT16	Source image's height in pixels

**S\_DRVBLT\_DEST\_FB**

Destination buffer.

Name	Type	Description
<i>u32FrameBufAddr</i>	UINT32	Destination buffer address to start rendering to
<i>i32XOffset</i>	INT32	No use
<i>i32YOffset</i>	INT32	No use
<i>i32Stride</i>	INT32	Destination buffer's stride in bytes
<i>i16Width</i>	INT16	Destination buffer's width in pixels
<i>i16Height</i>	INT16	Destination buffer's height in pixels

---

## 5.8. API Function

### ***bltOpen***

#### **Synopsis**

```
ERRCODE bltOpen(void);
```

#### **Description**

Initialize BLT and install interrupt service routine.

#### **Parameter**

None

#### **Return Value**

E_SUCCESS	Success
-----------	---------

### ***bltClose***

#### **Synopsis**

```
void bltClose(void);
```

#### **Description**

Tear down BLT.

#### **Parameter**

None

#### **Return Value**

None

***bltSetTransformMatrix*****Synopsis**

```
void bltSetTransformMatrix(S_DRVBLT_MATRIX sMatrix);
```

**Description**

Set up inverse transformation matrix.

**Parameter**

sMatrix	Transformation matrix as defined in <a href="#">S_DRVBLT_MATRIX</a> .
---------	--

**Return Value**

None

***bltGetTransformMatrix*****Synopsis**

```
void bltGetTransformMatrix(S_DRVBLT_MATRIX *psMatrix);
```

**Description**

Retrieve inverse transformation matrix which has set up.

**Parameter**

psMatrix	User-prepared buffer to save read-back transformation matrix as defined in <a href="#">S_DRVBLT_MATRIX</a> .
----------	---

**Return Value**

None

***bltSetSrcFormat*****Synopsis**

```
ERRCODE bltSetSrcFormat (E_DRVBLT_BMPPIXEL_FORMAT eSrcFmt);
```

**Description**

Set up source format.

**Parameter**

eSrcFmt	Source format as defined in <a href="#">E_DRVBLT_BMPPIXEL_FORMAT</a> .
---------	--

**Return Value**

E_SUCCESS	Success
-----------	---------

ERR_BLT_INVALID_SRCFMT	Invalid source format
------------------------	-----------------------

***bltGetSrcFormat*****Synopsis**

```
E_DRVBLT_BMPPIXEL_FORMAT bltGetSrcFormat(void);
```

**Description**

Retrieve source format which has set up.

**Parameter**

None

**Return Value**

Source format as defined in [E\\_DRVBLT\\_BMPPIXEL\\_FORMAT](#).

***bltSetDisplayFormat*****Synopsis**

```
ERRCODE bltSetDisplayFormat(E_DRVBLT_DISPLAY_FORMAT eDisplayFmt);
```

**Description**

Set up destination format.

**Parameter**

eDisplayFmt	Destination format defined in <a href="#">E_DRVBLT_DISPLAY_FORMAT</a> .
-------------	--

**Return Value**

E_SUCCESS	Success
ERR_BLT_INVALID_DSTFMT	Invalid destination format

***bltGetDisplayFormat*****Synopsis**

```
E_DRVBLT_DISPLAY_FORMAT bltGetDisplayFormat(void);
```

**Description**

Retrieve destination format which has set up.

**Parameter**

None

**Return Value**

Destination format as defined in [E\\_DRVBLT\\_DISPLAY\\_FORMAT](#).

***bltEnableInt*****Synopsis**

```
void bltEnableInt(E_BLT_INT_TYPE eIntType);
```

**Description**

Enable specified interrupt type.

**Parameter**

eIntType	Interrupt type as defined in <a href="#">E_BLT_INT_TYPE</a> .
----------	---

**Return Value**

None

***bltDisableInt*****Synopsis**

```
void bltDisableInt(E_BLT_INT_TYPE eIntType);
```

**Description**

Disable specified interrupt type.

**Parameter**

eIntType	Interrupt type as defined in <a href="#">E_BLT_INT_TYPE</a> .
----------	---

**Return Value**

None

***bltIsIntEnabled*****Synopsis**

```
BOOL bltIsIntEnabled (E_BLT_INT_TYPE eIntType);
```

**Description**

Query if the specified interrupt type is enabled.

**Parameter**

eIntType	Interrupt type as defined in <a href="#">E_BLT_INT_TYPE</a> .
----------	---

**Return Value**

TRUE	Specified interrupt enabled
FALSE	Specified interrupt disabled

***bltPollInt*****Synopsis**

```
BOOL bltPollInt(E_BLT_INT_TYPE eIntType);
```

**Description**

Query interrupt status of the specified interrupt type.

**Parameter**

eIntType	Interrupt type as defined in <a href="#">E_BLT_INT_TYPE</a> .
----------	---

**Return Value**

TRUE	Specified interrupt type active.
FALSE	Specified interrupt type inactive.

***bltInstallCallback*****Synopsis**

```
void bltInstallCallback (E_BLT_INT_TYPE eIntType, PFN_BLT_CALLBACK pfnCallback,  
PFN_BLT_CALLBACK* pfnOldCallback);
```

**Description**

Install callback function invocated on interrupt generated.

**Parameter**

eIntType	Interrupt type as defined in <a href="#">E_BLT_INT_TYPE</a> .
pfnCallback	New callback function to install. NULL to uninstall.
pfnOldCallback	User-prepared buffer to save previously installed callback function.

**Return Value**

None

***bltSetColorMultiplier*****Synopsis**

```
void bltSetColorMultiplier(S_DRVBLT_ARGB16 sARGB16);
```

**Description**

Set up color multipliers of A, R, G, and B channels for color transformation.

**Parameter**

sARGB16  
**S\_DRVBLT\_ARGB16.**

Color multipliers of A, R, G, and B channels as defined in

## Return Value

None

### ***bItGetColorMultiplier***

## Synopsis

```
void bltGetColorMultiplier(S_DRVBLT_ARGB16* psARGB16);
```

## Description

Retrieve color multipliers of A, R, G, and B channels which has set up.

## Parameter

**psARGB16** User-prepared buffer to save color multipliers of A, R, G, and B channels as defined in **S\_DRVBLT\_ARGB16**.

## Return Value

None

### ***bltSetColorOffset***

## Synopsis

```
void bltSetColorOffset(S_DRVBLT_ARGB16 sARGB16);
```

## Description

Set up color offsets of A, R, G, and B channels for color transformation.

## Parameter

**sARGB16** Color offsets of A, R, G, and B channels as defined in  
**S\_DRVBLT\_ARGB16.**

## Return Value

None

## ***bltGetColorOffset***

### Synopsis

```
void bltGetColorOffset(S_DRVBLT_ARGB16* psARGB16);
```

## Description

Retrieve color offsets of A, R, G, and B channels which has set up.

**Parameter**

psARGB16 User-prepared buffer to save color offsets of A, R, G, and B channels as defined in [S\\_DRVBLT\\_ARGB16](#).

**Return Value**

None

***bltSetSrcImage*****Synopsis**

```
void bltSetSrcImage(S_DRVBLT_SRC_IMAGE sSrcImage);
```

**Description**

Set up source image..

**Parameter**

sSrcImage Source image as defined in [S\\_DRVBLT\\_SRC\\_IMAGE](#).

**Return Value**

None

***bltSetDestFrameBuf*****Synopsis**

```
void bltSetDestFrameBuf(S_DRVBLT_DEST_FB sFrameBuf);
```

**Description**

Set up destination buffer..

**Parameter**

sFrameBuf Destination buffer as defined in [S\\_DRVBLT\\_DEST\\_FB](#).

**Return Value**

None

***bltSetARGBFillColor*****Synopsis**

```
void bltSetARGBFillColor(S_DRVBLT_ARGB8 sARGB8);
```

**Description**

Set up fill color for Fill operation, which can be ARGB8888 or RGB888 dependent on [bltSetFillAlpha](#).

**Parameter**

sARGB8 Fill color as defined in [S\\_DRVBLT\\_ARGB8](#).

**Return Value**

None

**Note**

If ARGB8888, it must be in non-premultiplied alpha format.

***bltGetARGBFillColor*****Synopsis**

```
void bltGetARGBFillColor(S_DRVBLT_ARGB8* psARGB8 );
```

**Description**

Retrieve ARGB8888 color for Fill operation which has set up.

**Parameter**

psARGB8 User-prepared buffer to save read-back ARGB8888 color  
for Fill operation.

**Return Value**

None

***bltGetBusyStatus*****Synopsis**

```
BOOL bltGetBusyStatus(void);
```

**Description**

Query if Fill/Blit operation is busy.

**Parameter**

None

**Return Value**

TRUE	Busy
FALSE	Free

***bltSetFillAlpha*****Synopsis**

```
void bltSetFillAlpha(BOOL bEnable);
```

**Description**

Set up whether or not fill color's alpha channel is in effect.

**Parameter**

bEnable

TRUE	Fill color is ARGB8888
------	------------------------

FALSE	Fill color is RGB888
-------	----------------------

**Return Value**

None

***bltGetFillAlpha*****Synopsis**

```
BOOL bltGetFillAlpha(void);
```

**Description**

Retrieve whether or not fill color's alpha channel is in effect which has set up.

**Parameter**

None

**Return Value**

TRUE	Fill color is ARGB8888.
------	-------------------------

FALSE	Fill color is RGB888
-------	----------------------

***bltSetTransformFlag*****Synopsis**

```
void bltSetTransformFlag(UINT32 u32TransFlag);
```

**Description**

Set up transform flag.

**Parameter**

U32TransFlag	Transform flag as defined in <a href="#">E_DRVBLT_TRANSFORM_FLAG</a> .
--------------	---

**Return Value**

None

## ***bItGetTransformFlag***

## Synopsis

UINT32 bltGetTransformFlag(void);

## Description

Retrieve transform flag which has set up.

## Parameter

None.

## Return Value

Transform flag as defined in EDRVBLT TRANSFORM FLAG.

### ***bltSetPaletteEndian***

## Synopsis

```
void bltSetPaletteEndian(E_DRVBLT_PALETTE_ORDER eEndian);
```

## Description

Set up endianness of palette index..

## Parameter

eEndian Endianness of palette index as defined in  
**E DRVBLT PALETTE ORDER.**

## Return Value

None

### ***bltGetPaletteEndian***

## Synopsis

E\_DRVBLT\_PALETTE\_ORDER bltGetPaletteEndian(void);

## Description

Retrieve endianness of palette index which has set up.

## Parameter

None

## Return Value

Endianness of palette index as defined in [E\\_DRVBLT\\_PALETTE\\_ORDER](#).

***bltSetColorPalette*****Synopsis**

```
void bltSetColorPalette(UINT32 u32PaletteInx, UINT32 u32Num, S_DRVBLT_ARGB8  
*psARGB);
```

**Description**

Set up palette's colors.

**Parameter**

u32PaletteInx	Index of palette to start to set up
u32Num	Number of colors to set up
psARGB	ARGB8888 colors

**Return Value**

None

***bltSetFillOP*****Synopsis**

```
void bltSetFillOP(E_DRVBLT_FILLOP eOP);
```

**Description**

Set up operation to be Fill or Blit.

**Parameter**

eOP	Operation as defined in <a href="#">E_DRVBLT_FILLOP</a> .
-----	---

**Return Value**

None

***bltGetFillOP*****Synopsis**

```
BOOL bltGetFillOP(void);
```

**Description**

Retrieve operation which has set up..

**Parameter**

None

**Return Value**

TRUE	Fill operation.
FALSE	Blit operation

### ***bltSetFillStyle***

#### **Synopsis**

```
void bltSetFillStyle(E_DRVBLT_FILL_STYLE eStyle);
```

#### **Description**

Set up other flags for Blit operation.

#### **Parameter**

eStyle	Other flags as defined in <a href="#">E_DRVBLT_FILL_STYLE</a> .
--------	---

#### **Return Value**

None

### ***bltGetFillStyle***

#### **Synopsis**

```
E_DRVBLT_FILL_STYLE bltGetFillStyle(void);
```

#### **Description**

Retrieve other flags for Blit operatioin which has set up.

#### **Parameter**

None

#### **Return Value**

Other flags as defined in [E\\_DRVBLT\\_FILL\\_STYLE](#).

### ***bltSetRevealAlpha***

#### **Synopsis**

```
void bltSetRevealAlpha(E_DRVBLT_REVEAL_ALPHA eAlpha);
```

#### **Description**

Set up premultiplied alpha or not for source format of ARGB8888

#### **Parameter**

eAlpha	Premultiplied alpha or not as specified in <a href="#">E_DRVBLT_REVEAL_ALPHA</a>
--------	---

#### **Return Value**

None

### ***bltGetRevealAlpha***

#### **Synopsis**

```
BOOL bltGetRevealAlpha(void);
```

#### **Description**

Retrieve premultiplied alpha or not for source format of ARGB8888.

#### **Parameter**

None

#### **Return Value**

Premultiplied alpha or not as specified in [E\\_DRVBLT\\_REVEAL\\_ALPHA](#)

### ***bltTrigger***

#### **Synopsis**

```
void bltTrigger(void);
```

#### **Description**

Start Fill/Blit operation..

#### **Parameter**

None

#### **Return Value**

None

### ***bltSetRGB565TransparentColor***

#### **Synopsis**

```
void bltSetRGB565TransparentColor(UINT16 u16RGB565);
```

#### **Description**

Set up transparent color for source format of RGB565 for color key enabled

#### **Parameter**

u16RGB565	RGB565 to be transparent color
-----------	--------------------------------

#### **Return Value**

None

***bltGetRGB565TransparentColor*****Synopsis**

```
UINT16 bltGetRGB565TransparentColor(void);
```

**Description**

Retrieve transparent color which has set up..

**Parameter**

None

**Return Value**

RGB565 to be transparent color

***bltSetRGB565TransparentColorCtl*****Synopsis**

```
void bltSetRGB565TransparentColorCtl(BOOL bEnable);
```

**Description**

Enable color key or not.

**Parameter**

bEnable

TRUE	Enable color key
FALSE	Disable color key

**Return Value**

None

***bltGetRGB565TransparentColorCtl*****Synopsis**

```
BOOL bltGetRGB565TransparentColorCtl(void);
```

**Description**

Retrieve color key enabled or not.

**Parameter**

None

**Return Value**

TRUE	Color key enabled
------	-------------------

FALSE

Color key disabled

***bltFlush*****Synopsis**

```
void bltFlush(void);
```

**Description**

Wait for Fill/Blit operation to complete.

**Parameter**

None

**Return Value**

None

---

## 5.9. Error Code Table

Code Name	Value	Description
Successful	0	Success
ERR_BLT_INVALID_INT	BLT_ERR_ID   0x01	Invalid interrupt type
ERR_BLT_INVALID_SRCFMT	BLT_ERR_ID   0x02	Invalid source format
ERR_BLT_INVALID_DSTFMT	BLT_ERR_ID   0x01	Invalid destination format

# 6. CRC Library Introduction

The cyclic redundancy check (CRC) generator can perform CRC calculation with programmable polynomial settings. It supports CPU PIO mode directly and can use the VDMA function to get the data.

## 6.1. Feature

- Supports four common polynomials CRC-CCITT, CRC-8, CRC-16, and CRC-32
  - CRC-CCITT:  $X^{16} + X^{12} + X^5 + 1$
  - CRC-8:  $X^8 + X^2 + X + 1$
  - CRC-16:  $X^{16} + X^{15} + X^2 + 1$
  - CRC-32:  $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$
- Programmable seed value
- Supports programmable order reverse setting for input data and CRC checksum
- Supports programmable 1's complement setting for input data and CRC checksum.
- Supports 8/16/32-bit of data width in CPU PIO mode
  - 8-bit write mode: 1-AHB clock cycle operation
  - 16-bit write mode: 2-AHB clock cycle operation
  - 32-bit write mode: 4-AHB clock cycle operation
- Two CRC channels

## 6.2. API Data Structure

### *E\_CRC\_CHANNEL\_INDEX*

CRC channel index.

Name	Value	Description
<i>E_CHANNEL_0</i>	0	CRC channel 0
<i>E_CHANNEL_1</i>	1	CRC channel 1

### *E\_CRC\_OPERATION*

CRC channel operation.

Name	Value	Description
<i>E_CH_DISABLE</i>	0	CRC channel disable
<i>E_CH_ENABLE</i>	1	CRC channel enable

### ***E\_CRC\_MODE***

CRC polynomials.

Name	Value	Description
<i>E_CRC_CCITT</i>	0	CRC-CCITT polynomial
<i>E_CRC8</i>	1	CRC-8 polynomial
<i>E_CRC16</i>	2	CRC-16 polynomial
<i>E_CRC32</i>	3	CRC-32 polynomial

### ***E\_WRITE\_LENGTH***

CRC data width in CPU PIO mode, VDMA mode only supports 32-bit write mode.

Name	Value	Description
<i>E_LENGTH_BYTE</i>	0	8-bit write mode
<i>E_LENGTH_HALF_WORD</i>	1	16-bit write mode
<i>E_LENGTH_WORD</i>	2	32-bit write mode

### ***E\_DATA\_1sCOM***

1's complement setting for input data and CRC checksum.

Name	Value	Description
<i>E_IsCOM_OFF</i>	0	1's complement disable
<i>E_IsCOM_ON</i>	1	1's complement enable

### ***E\_DATA\_REVERSE***

Order reverse setting for input data and CRC checksum.

Name	Value	Description
<i>E_REVERSE_OFF</i>	0	Order reverse disable
<i>E_REVERSE_ON</i>	1	Order reverse enable

***E\_TRANSFER\_MODE***

CRC CPU PIO or VDMA mode.

Name	Value	Description
<i>E_CRC_CPU_PIO</i>	0	CRC CPU PIO mode
<i>E_CRC_VDMA</i>	1	CRC VDMA mode

***S\_CRC\_CHANNEL\_INFO***

CRC channel information.

Name	Type	Description
<i>bInRequest</i>	BOOL	CRC channel is in request or not
<i>bInUse</i>	BOOL	CRC channel is in use or not

***S\_CRC\_DESCRIPTOR\_SETTING***

CRC channel description of a calculation.

Name	Type	Description
<i>ePolyMode</i>	<i>E_CRC_MODE</i>	CRC polynomials
<i>eWriteLength</i>	<i>E_WRITE_LENGTH</i>	Data width of write modes
<i>eChecksumCom</i>	<i>E_DATA_1sCOM</i>	1's Complement setting for checksum
<i>eWdataCom</i>	<i>E_DATA_1sCOM</i>	1's Complement setting for input data
<i>eChecksumRvs</i>	<i>E_DATA_REVERSE</i>	order reverse setting for checksum
<i>eWdataRvs</i>	<i>E_DATA_REVERSE</i>	order reverse setting for input data
<i>eTransferMode</i>	<i>E_TRANSFER_MODE</i>	CRC run in CPU PIO or VDMA mode
<i>uSeed</i>	UINT32	CRC seed value

---

## 6.3. API Function

***CRC\_Init*****Synopsis**

```
INT32 CRC_Init(void);
```

**Description**

Initialize the software resource of CRC driver, call EDMA\_Init to initialize VDMA and enable interrupt.

**Parameter**

None

**Return Value**

Successful	Always returns Successful
------------	---------------------------

***CRC\_Exit*****Synopsis**

```
void CRC_Exit(void);
```

**Description**

Clear CRC initial flag.

**Parameter**

None

**Return Value**

None

***CRC\_Request*****Synopsis**

```
INT32 CRC_Request(INT32 channel);
```

**Description**

Specify a channel for request.

**Parameter**

channel	CRC channel number
---------	--------------------

**Return Value**

Successful	Specified channel is requested
CRC_ERR_INVAL	Specified channel number is invalid
CRC_ERR_BUSY	Specified channel is busy

***CRC\_Free*****Synopsis**

```
void CRC_Free(INT32 channel);
```

**Description**

Release a previously acquired channel.

**Parameter**

channel	CRC channel number
---------	--------------------

**Return Value**

None

***CRC\_FindandRequest*****Synopsis**

```
INT32 CRC_FindandRequest(void);
```

**Description**

Try to find a free channel and request it.

**Parameter**

None

**Return Value**

Successful	Allocated channel is returned
CRC_ERR_NODEV	No free channel is found

***CRC\_Run*****Synopsis**

```
UINT32 CRC_Run(INT32 channel, UINT8 *pDataBuf, UINT32 uDataLen,  
S_CRC_DESCRIPTOR_SETTING *psCRCDescript);
```

**Description**

Start to run a CRC calculation and wait for its finish.

**Parameter**

channel	CRC channel number
pDataBuf	Input buffer address
uDataLen	Length of input buffer in bytes
psCRCDescript	Pointer to the channel description of this CRC calculation

**Return Value**

Successful	CRC checksum is returned
------------	--------------------------

---

CRC_ERR_STATUS	Channel is not in request
CRC_ERR_BUSY	Channel is in use and cannot run a calculation

---

## 6.4. Error Code Table

Code Name	Value	Description
Successful	0	Success
CRC_ERR_INVAL	CRC_ERR_ID   0x01	Channel number is invalid
CRC_ERR_NODEV	CRC_ERR_ID   0x02	No free channel is found
CRC_ERR_STATUS	CRC_ERR_ID   0x03	Channel status is wrong
CRC_ERR_BUSY	CRC_ERR_ID   0x04	Channel is busy

# 7. EDMA Library Overview

This library is designed to make user application to set N9H26 EDMA more easily.  
The EDMA library has the following features:

- Support color space transforms (RGB565, RGB555, RGB888 and YUV422) for VDMA.
- Support transfers data to and from memory or transfer data to and from APB.
- Support hardware Scatter-Gather function.

---

## 7.1. Programming Guide

### System Overview

The N9H26 contains an enhanced direct memory access (EDMA) controller that transfers data to and from memory or transfer data to and from APB. The EDMA controller has 11-channel DMA that include 3 channel VDMA (Video-DMA, Memory-to-Memory) and 8 channels PDMA (Peripheral-to-Memory or Memory-to-Peripheral). For channel 0/5/8 VDMA mode, it also support color format transform and stripe mode transfer. For PDMA channel (EDMA CH1~CH4, CH9~CH12), it can transfer data between the Peripherals APB IP (ex: UART, SPI, ADC....) and Memory. The N9H26 also support hardware scatter-gather function, software can set CSR<sub>x</sub> [SG\_EN] to enable scatter-gather function.

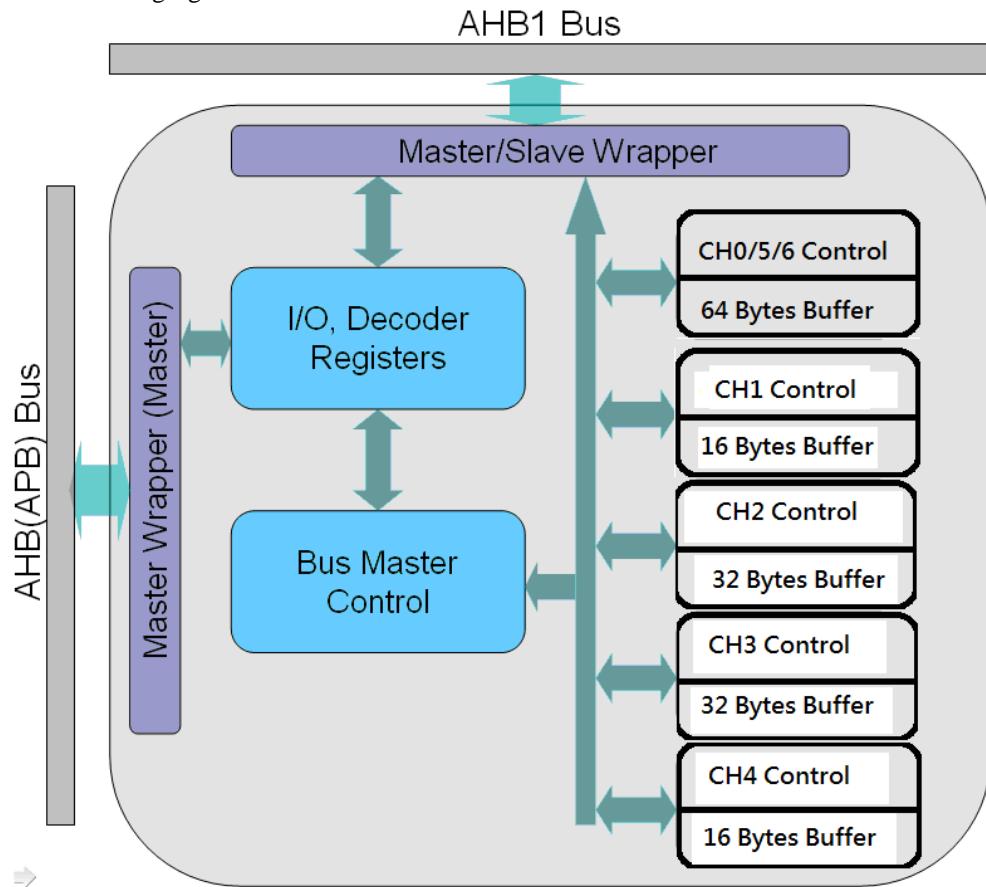
Software can stop the EDMA operation by disable DMA [DMACEN]. The CPU can recognize the completion of an EDMA operation by software polling or when it receives an internal EDMA interrupt. The N9H26 VDMA controller can increment source or destination address, decrement or fixed them as well, and the PDMA can increment source or destination, fixed or wrap around address.

#### EDMA Features

- AMBA AHB master/slave interface compatible, for data transfer and register read/write.
- Support packaging format color space transforms (RGB565, RGB555, RGB888 and YUV422) for VDMA.
- Support stride mode transfer mode for VDMA.
- VDMA support 32-bit source and destination addressing range, address increment, decrement and fixed.
- PDMA support 32-bit source and destination addressing range address increment, fixed and wrap around.
- Support hardware Scatter-Gather function.

## Block Diagram

The following figure describes the architecture of EDMA.



## EDMA Control

### ■ VDMA Transfer

The main purpose of VDMA channel is to perform a memory-to-memory transfer. Besides the pure memory copy, it also provides the color format transformation in packet during the transfer.

Software must enable DMA channel DMA [DMACEN] and then write a valid source address to the DMA\_SARx register, a destination address to the DMA\_DSABx register, and a transfer count to the DMA\_BCRx register. Next, trigger the DMA\_CSRx [Trig\_EN]. If the source address and destination are not in wrap around mode, the transfer will start transfer until DMA\_CBCRx reaches zero (in wrap around mode, when DMA\_CBCRx equal zero, the DMA will reload DMA\_CBCRx and work around until software disable DMA\_CSRx [DMACEN]). If an error occurs during the EDMA operation, the channel stops unless software clears the error condition, sets the DMA\_CSRx [SW\_RST] to reset the EDMA channel and set EDMA\_CSRx [EDMACEN] and [Trig\_EN] bits field to start again.

### ■ PDMA Transfer

The PDMA is used to transfer data between SDRAM and APB device. Currently, the APB device only supports UART 0/1, SPIMs 0/1 and ADC audio recording. The data direction can be from APB device or to APB device dependent on the setting of PDMA\_CSRx[MODE\_SEL]. Hardware IP will do the necessary handshaking signal between PDMA and APB device. In the PDMA transfer, the APB device data port should be set as the source or destination address dependent on the setting of PDMA\_CSRx[MODE\_SEL], and the address direction must be set as fixed for APB address. Besides this, the APB device has corresponding register setting to enable PDMA transfer.

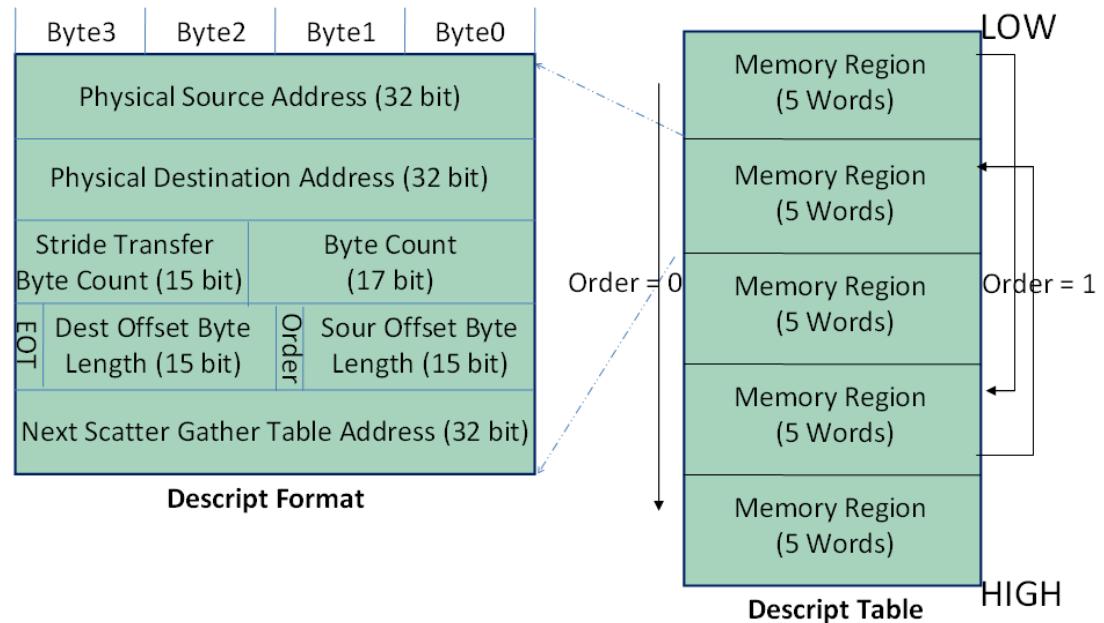
Below table lists the control register and control bit for it.

APB IP	Control Register	Control Bits
Uart 0/1	UA_IER (UA_BA0/1+0x04)	DMA_Tx_En and DMA_Rx_En
SPI0/1	SPI0/1_EDMA	EDMA_RW and EDMA_GO
ADC	AGCP1	EDMA_MODE

Moreover, the EDSSR register in global control is necessary to notice. The PDMA cannot use the same channel selection in it when PDMA is set.

## ■ Scatter Gather Transfer

The N9H26 also support hardware scatter-gather function, software can set DMA\_CSRx [SG\_EN] to enable scatter-gather function. When in scatter-gather function mode, some register will automatically updated by descriptor table. The descriptor table format is show as following:



The field definition of scatter table is as below:

- Physical Source Address (32 bits)
- Physical Destination Address (32 bits)
- Byte Count : Transfer Byte Count (17 bits)
- Stride Transfer Byte Count (15 bits)
- EOT : End of Table (1 bit)
- Source Offset Byte Length (15 bits)

- Oder : Scatter Gather table in Link list mode or not (1 bit)

- Destination Offset Byte length (15 bits)

- Next Scatter Gather Table Address (32 bits)

Note : only when in stride transfer mode (CTCSR[Stride\_EN]=1), Stride Transfer Byte count, Source Offset Byte length and Destination Offset Byte Length is meaningful

---

## 7.2. APIs Specification Functions

### ***EDMA\_Init***

#### **Synopsis**

```
int EDMA_Init(void)
```

#### **Description**

This function initializes the software resource.

#### **Parameter**

None

#### **Return Value**

0	Always successes
---	------------------

#### **Example**

```
EDMA_Init();
```

### ***EDMA\_Exit***

#### **Synopsis**

```
void EDMA_Exit(void)
```

#### **Description**

Disable EDMA engine clock.

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
EDMA_Exit();
```

## ***VDMA\_FindandRequest***

### **Synopsis**

```
int VDMA_FindandRequest(void)
```

### **Description**

This function tries to find a free channel in the specified priority group.

### **Parameter**

None

### **Return Value**

SUCCESS	Allocation channel is returned.
FAIL	EDMA_ERR_NODEV is returned.

### **Example**

```
int g_VdmaCh;  
  
g_VdmaCh = VDMA_FindandRequest();
```

## ***PDMA\_FindandRequest***

### **Synopsis**

```
int PDMA_FindandRequest(void)
```

### **Description**

This function tries to find a free channel in the specified priority group.

### **Parameter**

None

### **Return Value**

SUCCESS	Allocation channel is returned.
FAIL	EDMA_ERR_NODEV is returned.

### **Example**

```
int g_PdmaCh;  
  
g_PdmaCh = PDMA_FindandRequest();
```

## ***EDMA\_SetupHandlers***

### **Synopsis**

```
int EDMA_SetupHandlers(int channel, int interrupt, PFN_DRVEDMA_CALLBACK
irq_handler, void *data)
```

### Description

This function is used to setup EDMA channel notification handlers.

### Parameter

channel	EDMA channel number
interrupt	EDMA interrupt enable
irq_handler	The callback function pointer for specified EDMA channel .
data	User specified value to be passed to the handlers.

### Return Value

SUCCESS	0 is returned.
FAIL	EDMA_ERR_NODEV is returned.

### Example

```
/* Install Callback function */

EDMA_SetupHandlers(0, eDRV_EDMA_BLKD_FLAG, EdmaIrqHandler, 0);
```

## ***EDMA\_SetupSingle***

### Synopsis

```
int EDMA_SetupSingle(int channel, unsigned int src_addr, unsigned int dest_addr, unsigned
int dma_length)
```

### Description

This function is used to setup EDMA channel for linear memory to/from device transfer.

### Parameter

channel	EDMA channel number
src_addr	Source address
dest_addr	Destination address
dma_length	Length of the transfer request in bytes

### Return Value

SUCCESS	0 is returned.
FAIL	< 0 is returned.
	EDMA_ERR_BUSY : specified channel is busy.
	EDMA_ERR_INVAL : null address or zero length.

**Example**

```
EDMA_SetupSingle (0, SRC_ADDR, DEST_ADDR, 0x10000);
```

***EDMA\_Free*****Synopsis**

```
void EDMA_Free(int channel)
```

**Description**

This function is used to release previously acquired channel.

**Parameter**

Channel      EDMA channel number

**Return Value**

None

**Example**

```
EDMA_Free (0);
```

***EDMA\_SetupSG*****Synopsis**

```
int EDMA_SetupSG(int channel, unsigned int src_addr, unsigned int dest_addr, unsigned int  
dma_length)
```

**Description**

This function is used to setup EDMA channel SG list.

**Parameter**

channel	EDMA channel number
src_addr	Source address
dest_addr	Destination address
length	Total length of the transfer request in bytes

**Return Value**

SUCCESS	0 is returned.
FAIL	< 0 is returned.
EDMA_ERR_BUSY : specified channel is busy.	
EDMA_ERR_INVAL : zero length or address is not PAGE_SIZE alignment.	

**Example**

```
EDMA_SetupSG (0, SRC_ADDR, DEST_ADDR, 0x10000);
```

***EDMA\_FreeSG*****Synopsis**

```
void EDMA_FreeSG(int channel)
```

**Description**

This function is used to release previously acquired channel SG list.

**Parameter**

Channel      EDMA channel number

**Return Value**

None

**Example**

```
EDMA_FreeSG (0);
```

***EDMA\_SetupCST*****Synopsis**

```
int EDMA_SetupCST(int channel, E_DRVEDMA_COLOR_FORMAT eSrcFormat,  
E_DRVEDMA_COLOR_FORMAT eDestFormat)
```

**Description**

This function is used to setup EDMA channel for color space transform.

**Parameter**

channel      EDMA channel number  
eSrcFormat    The source color format  
eDestFormat   The destination color format

**Return Value**

SUCCESS      0 is returned.  
FAIL          EDMA\_ERR\_BUSY is returned.

**Example**

```
/* Setup color space transform RGB565 to YCbCr422 */  
  
EDMA_SetupCST(g_VdmaCh, eDRVVEDMA_RGB565, eDRVVEDMA_YCbCr422);
```

## ***EDMA\_ClearCST***

### **Synopsis**

```
int EDMA_ClearCST(int channel)
```

### **Description**

This function is used to disable EDMA channel color space transform.

### **Parameter**

channel      EDMA channel number

### **Return Value**

SUCCESS      0 is returned.

FAIL          EDMA\_ERR\_BUSY is returned.

### **Example**

```
/* Disable EDMA color space transform */

EDMA_ClearCST (g_VdmaCh);
```

## ***EDMA\_Trigger***

### **Synopsis**

```
void EDMA_Trigger(int channel)
```

### **Description**

This function is used to start EDMA channel transfer.

### **Parameter**

channel      EDMA channel number

### **Return Value**

None.

### **Example**

```
/* Trigger EDMA channel transfer */

EDMA_Trigger (g_VdmaCh);
```

## ***EDMA\_IsBusy***

### **Synopsis**

```
int EDMA_IsBusy(int channel)
```

**Description**

This function is used to query EDMA channel is busy or not.

**Parameter**

channel	EDMA channel number
---------	---------------------

**Return Value**

TRUE	EDMA channel is busy.
FALSE	EDMA channel is ready.

**Example**

```
EDMA_IsBusy (g_VdmaCh);
```

***EDMA\_SetAPB*****Synopsis**

```
int EDMA_SetAPB(int channel, E_DRVEDMA_APB_DEVICE eDevice,
E_DRVEDMA_APB_RW eRWAPB, E_DRVEDMA_TRANSFER_WIDTH eTransferWidth)
```

**Description**

This function is used to setup EDMA channel for APB device.

**Parameter**

channel	EDMA channel number
eDevice	Specify the APB device which will use the EDMA channel
eRWAPB	Indicate that read or write APB device
eTransferWidth	Set the transfer width for specified channel

**Return Value**

SUCCESS	0 is returned.
FAIL	EDMA_ERR_BUSY is returned.

**Example**

```
/* Setup ADC use EDMA channel*/
EDMA_SetAPB (g_PdmaCh, eDRVVEDMA_ADC, eDRVVEDMA_READ_APB, eDRVVEDMA_WIDTH_32BITS);
```

***EDMA\_SetWrapINTType*****Synopsis**

```
int EDMA_SetWrapINTType(int channel, int type)
```

**Description**

Set the EDMA wrap around interrupt select for specified channel.

**Parameter**

channel	EDMA channel number
type	Set the wrap around mode for specified channel

**Return Value**

SUCCESS	0 is returned.
FAIL	EDMA_ERR_BUSY is returned.

**Example**

```
/* Set wrap around mode with half and empty */

EDMA_SetWrapINTType (g_PdmaCh, eDRVVEDMA_WRAPAROUND_EMPTY |  
eDRVVEDMA_WRAPAROUND_HALF);
```

***EDMA\_SetDirection*****Synopsis**

```
int EDMA_SetDirection(int channel, int src_dir, int dest_dir)
```

**Description**

This function is used to set transfer direction for specified channel.

**Parameter**

channel	EDMA channel number
src_dir	The source transfer direction
dest_dir	The destination transfer direction

**Return Value**

SUCCESS	0 is returned.
FAIL	EDMA_ERR_BUSY is returned.

**Example**

```
/* Set source transfer direction fixed and destination wraparound*/

EDMA_SetDirection (g_PdmaCh , eDRVVEDMA_DIRECTION_FIXED,  
eDRVVEDMA_DIRECTION_WRAPAROUND);
```

### 7.3. Error Code Table

Code Name	Value	Description
EDMA_ERR_NODEV	0xFFFF0401	No device error
EDMA_ERR_INVAL	0xFFFF0402	Invalid parameter error
EDMA_ERR_BUSY	0xFFFF0403	Channel busy error

# 8. EMAC Library Introduction

This document is written for user applications which want to make use of EMAC through provided API.

## 8.1. Feature

- Supports IEEE Std. 802.3 CSMA/CD protocol.
- Supports both half and full duplex for 10M/100M bps operation.
- Supports RMII interface.
- Supports MII Management function.
- Supports pause and remote pause function for flow control.
- Supports long frame (more than 1518 bytes) and short frame (less than 64 bytes) reception.
- Supports 16 entries CAM function for Ethernet MAC address recognition.
- Supports internal loop back mode for diagnostic.
- Supports 256 bytes embedded transmit and receive FIFO.
- Supports DMA function.

## 8.2. API Data Structure

### *S\_FrameDescriptor*

Tx/Rx buffer descriptor.

Name	Type	Description
<i>Status1</i>	UINT32	RX/TX ownership, RX status, TX controlbits
<i>FrameDataPtr</i>	UINT32	Frame data pointer
<i>Status2</i>	UINT32	TX status, RX NAT info
<i>NextFrameDescriptor</i>	UINT32	Next frame descriptor pointer

### *S\_Etheader*

Ethernet header

Name	Type	Description
<i>DestinationAddr[6]</i>	UINT8	Destination MAC address
<i>SourceAddr[6]</i>	UINT8	Source MAC address
<i>LengthOrType[2]</i>	UINT8	Frame size or type

**S\_MACFrame**

MAC frame data structure.

Name	Type	Description
<i>Header</i>	S_Etheader	Ethernet header
<i>LLCData[1522]</i>	UINT8	Payload

**S\_MACTxStatus: reserve****S\_MACRxStatus: reserve****NVT\_EMAC\_T**

Major data structure for EMAC Init.

Name	Type	Description
<i>srcMACAddr[6]</i>	UINT8	Source MAC address
<i>speedMode</i>	UINT32	100M or 10M bps, Half or Full duplex
<i>recvPacketType</i>	UINT32	Accept packet type, ex: unicast, broadcast.
<i>rxFDBaseAddr</i>	UINT32	RX fame descriptor base address
<i>txFDBaseAddr</i>	UINT32	TX frame descriptor base address
<i>rxFBABaseAddr</i>	UINT32	RX frame buffer address base
<i>txFBABaseAddr</i>	UINT32	TX frame buffer address base
<i>portNo</i>	UINT32	EMAC port0 or port1

---

## 8.3. API Function

**EMAC\_Init****Synopsis**

```
BOOL EMAC_Init(NVT_EMAC_T *emac);
```

**Description**

Initialize EMAC and PHY.

**Parameter**

emac	Init setting structure as defined in NVT_EMAC_T.
------	--

**Return Value**

True	Sucess
False	Failed

***EMAC\_SetRxCallBack*****Synopsis**

```
BOOL EMAC_SetRxCallBack(PVOID pvFun);
```

**Description**

Set RX call back function

**Parameter**

pvFun	callback function pointer
-------	---------------------------

**Return Value**

True	Sucess
False	Failed

***EMAC\_SendPacket*****Synopsis**

```
BOOL EMAC_SendPacket(UINT8 *data,int size);
```

**Description**

Send one packet.

**Parameter**

data	Paket data pointer
size	Packet size

**Return Value**

True	Sucess
False	Failed

***EMAC\_EnableWakeOnLan*****Synopsis**

```
void EMAC_EnableWakeOnLan();
```

**Description**

To enable EMAC support WOL and set as one wake up source

**Parameter**

None

**Return Value**

None

***EMAC\_Exit*****Synopsis**

```
void EMAC_Exit()
```

**Description**

Close EMAC and let it down

**Parameter**

None

**Return Value**

None

---

## 8.4. Error Code Table

Code Name	Value	Description
True	1	Success
False	0	Failure

# 9. Font Library Overview

The N9H26 Font library provides a set of APIs to write character or draw rectangle border to frame buffer. With these APIs, user can quickly show some string on N9H26 demo board or evaluation board. The library is a software solution. After update the frame buffer, VPOST controller can show the content to panel or TV.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

## 9.1. Font Library API

### InitFont

#### Synopsis

```
void InitFont(S_DEMO_FONT* ptFont, UINT32 u32FrameBufAddr);
```

#### Description

This function is used to init the font library. To get some information of font library.

#### Parameter

ptFont	Font library information pointer.
u32FrameBufAddr	Frame buffer base address.

*Table 9-1:Font Information*

Field name	Data Type	Description
u32FontRectWidth	UINT32	Font width. Now fixed in 16
u32FontRectHeight	UINT32	Font height. Now fixed in 22
u32FontOffset	UINT32	Font Offset. Now fixed in 11
u32FontStep	UINT32	Font Step. Now fixed in 10
u32FontOutputStride	UINT32	Output Stride. It should same as the panel width
u32FontInitDone	UINT32	1 = Font library initialized done. 0 = Font library not yet initialized done or

		deinitialized.
u32FontSize	UINT32	Useless.
pu32FontFileTmp	UINT32	Useless
pu32FontFile	UINT32	Pointer of font file
au16FontColor[3]	UINT16	RGB565 color au16FontColor[0]: Font background color au16FontColor[1]: Font color au16FontColor[2]: Border color

**Return Value**

None

**Example**

```
/* Initialize font library */

__align(32) static S_DEMO_FONT s_sDemo_Font;

__align(32) UINT16 u16FrameBuffer[_LCM_WIDTH_*_LCM_HEIGHT_];

InitFont(&s_sDemo_Font, u16FrameBufAddr);
```

**DemoFont\_PaintA****Synopsis**

```
void DemoFont_PaintA(S_DEMO_FONT* ptFont, UINT32 u32x, UINT32 u32y, PCSTR
                      pszString)
```

**Description**

This function writes a specified string to frame buffer.

**Parameter**

ptFont	Font library information pointer. Reference the <a href="#">Table 9-1:Font Information</a>
u32x	start x position.
u32y	start y position.
pszString	The specified string for writing to frame buffer.

**Return Value**

None

**Example**

```

/* Draw a string to the position (0, 0) of frame buffer */

__align(32) static S_DEMO_FONT s_sDemo_Font

char szString[64];

sprintf(szString, "FA93 Font Code");

DemoFont_PaintA(&s_sDemo_Font, 0, 0, szString);

```

## ***UnInitFont***

### **Synopsis**

void UnInitFont(S\_DEMO\_FONT\* ptFont)

### **Description**

De-Initialize the font library.

### **Parameter**

ptFont           Font library information pointer. Reference the [Table 9-1:Font Information](#)

### **Return Value**

None

### **Example**

```

/* De-Initialize the font library */

__align(32) static S_DEMO_FONT s_sDemo_Font

UninitFont(&s_sDemo_Font);

```

## ***DemoFont\_Rect***

### **Synopsis**

void DemoFont\_Rect(SDEMO\_FONT\* ptFont, S\_DEMO\_RECT\* ptRect)

### **Description**

This function draws a solid rectangle to frame buffer.

### **Parameter**

ptFont           Font library information pointer. Reference the [Table 9-1:Font Information](#)  
 ptRect           Solid retangle pointer

Table 9-2:Rectangle Information

Field name	Data Type	Description
------------	-----------	-------------

u32StartX	UINT32	X position for the upper-left corner
u32StartY	UINT32	Y position for the upper-left corner
u32EndX	UINT32	X position for the lower-right corner
u32EndY	UINT32	Y position for the lower-right corner

**Return Value**

None

**Example**

```
/* Draw a solid rectangle with dimension 320x240*/
__align(32) static S_DEMO_FONT s_sDemo_Font;
static S_DEMO_RECT s_sDemo_Rect;
s_sDemo_Rect.u32StartX = 0;
s_sDemo_Rect.u32StartY = 0;
s_sDemo_Rect.u32EndX = 320-1;
s_sDemo_Rect.u32EndY = 240-1;
DemoFont_Rect(&ptFont,
               &s_sDemo_Rect);
```

***DemoFont\_RectClear*****Synopsis**

```
void DemoFont_RectClear(SDEMO_FONT* ptFont, S_DEMO_RECT* ptRect)
```

**Description**

This function clears a solid rectangle to background color in frame buffer. The background color was fixed as 0. It means the color is black for RGB565 format.

**Parameter**

ptFont	Font library information pointer. Reference the <a href="#">Table 9-1:Font Information</a>
ptRect	Solid retangle pointer. Reference the <a href="#">Table 9-2:Rectangle Information</a>

**Return Value**

None

**Example**

```
/* Clear a solid rectangle from position (0, 0) to (319, 240) */

__align(32) static S_DEMO_FONT s_sDemo_Font;

static S_DEMO_RECT s_sDemo_Rect;

s_sDemo_Rect.u32StartX = 0;

s_sDemo_Rect.u32StartY = 0;

s_sDemo_Rect.u32EndX = 320-1;

s_sDemo_Rect.u32EndY = 240-1;

DemoFont_RectClear(&ptFont,
&s_sDemo_Rect);
```

## ***Font\_ClrFrameBuffer***

### **Synopsis**

```
void Font_ClrFrameBuffer(UINT32 u32FrameBufAddr)
```

### **Description**

This function clears the specified frame buffer to fixed background color (black color). The dimension is specified in the header file- \_LCM\_WIDTH\_ and \_LCM\_HEIGHT\_ with 16-bit pixel format.

### **Parameter**

u32FrameBufAddr      Frame buffer base address.

### **Return Value**

None

### **Example**

```
__align(32) UINT16 u16FrameBuffer[_LCM_WIDTH_*_LCM_HEIGHT_];

/* Clear frame buffer to background color-black*/

Font_ClrFrameBuffer(u16FrameBuffer);
```

## ***DemoFont\_Border***

### **Synopsis**

```
void DemoFont_Border(S_DEMO_FONT* ptFont, S_DEMO_RECT* ptRect, UINT32 u32Width);
```

### Description

This function draw a hollow rectangle with the specified border width.

### Parameter

ptFont	Font library information pointer. Reference the <a href="#">Table 9-1:Font Information</a>
ptRect	Solid retangle pointer. Reference the <a href="#">Table 9-2:Rectangle Information</a> .
u32Width	Border width.

### Return Value

None

### Example

```
__align(32) static S_DEMO_FONT s_sDemo_Font;

S_DEMO_RECT s_sDemo_Rect;
__align(32) UINT16 u16FrameBuffer[_LCM_WIDTH_*_LCM_HEIGHT_];

InitFont(&s_sDemo_Font, u16FrameBuffer);

    s_sDemo_Rect.u32StartX =0;

s_sDemo_Rect.u32StartY = 0;
s_sDemo_Rect.u32EndX = _LCM_WIDTH_-1;
s_sDemo_Rect.u32EndY = _LCM_HEIGHT_-1;

/* Draw a hollow rectangle with dimension same as panel and border is 2 pixels width */
DemoFont_Border(&s_sDemoFont,
    &s_sDemo_Rect,
    2);
```

## **DemoFont\_ChangeFontColor**

### Synopsis

```
void DemoFont_ChangeFontColor(S_DEMO_FONT* ptFont, UINT16 u16RGB565);
```

**Description**

This function sets the font color. The format is RGB565.

**Parameter**

ptFont            Font library information pointer. Reference the [Table 9-1:Font Information](#)  
u16RGB565      RGB565n format

**Return Value**

None

**Example**

```
__align(32) static S_DEMO_FONT s_sDemo_Font;  
/* Set the blue font color */  
DemoFont_ChangeFontColor(&s_sDemo_Font, 0x001F);
```

***DemoFont\_GetFontColor*****Synopsis**

```
UINT16 DemoFont_GetFontColor(S_DEMO_FONT* ptFont);
```

**Description**

This function gets current font color. The return value format is RGB565.

**Parameter**

ptFont            Font library information pointer. Reference the [Table 9-1:Font Information](#)

**Return Value**

RGB565 format

**Example**

```
__align(32) static S_DEMO_FONT s_sDemo_Font;  
  
UINT16 u16FontColor;  
  
/* Get font color */  
  
u16FontColor = DemoFont_GetFontColor(&s_sDemo_Font);
```

# 10. GNAND Library Overview

N9H20/N9H26 Non-OS library consists of a sets of libraries. These libraries are built to access those on-chip functions such as VPOST, APU, SIC, USBH, USBD, GPIO, I2C, SPI and UART, as well as File System (NVTFS), TCP/IP protocol (lwip), USB MassStorage devices (UMAS) and NAND Flash devices (GNAND). This document describes the provided APIs of GNAND library. With these APIs, user can quickly build a binary target for GNAND library on N9H26 micro processor.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

---

## 10.1. GNAND Library Introduction

In GNAND library, a NAND was thought of as a disk. User can access NAND by logical block address and don't worry about the bad block issue. It's possible that a few leading physical blocks were reserved for boot code or information area. GNAND library will not access those reserved blocks.

The Generic NAND (GNAND) library has the following features:

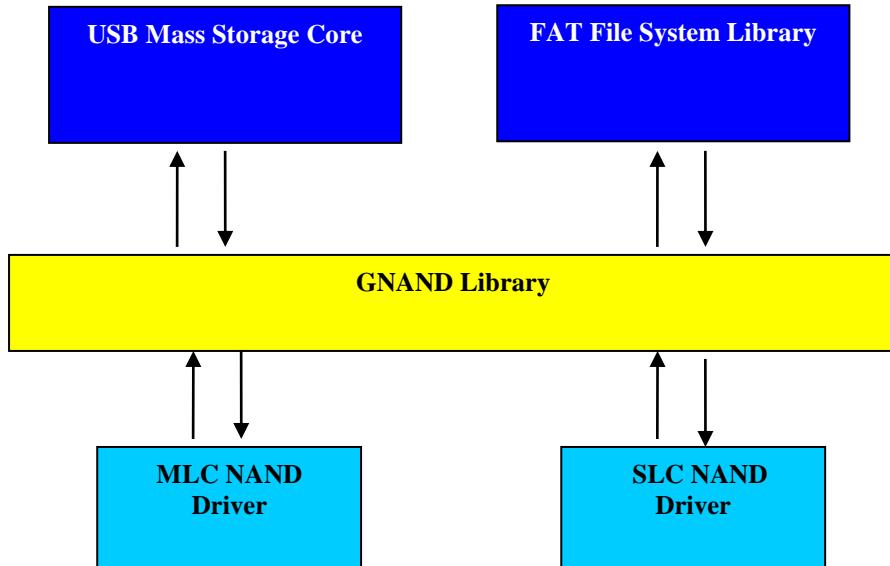
- Mapping between logical block and physical block to support bad block management
- Platform independent.
- Support both FAT file system and USB mass storage device
- Support both SLC and MLC NAND
- Able to recover from any power-off exceptions
- High performance, fast startup
- Support multiple NAND disk
- Support two disks in one NAND (reserved NAND partition)
- Dirty page management to support garbage collection feature
- Balanced usage on all physical blocks to support wear-leveling feature (will supported in the future)

---

## 10.2. Programming Guide

### System Overview

GNAND library works as a hardware independent library. NAND disk access service was provided by NAND driver. File system access service was provided by upper layer FAT file system library or USB mass storage device driver. The relationship between these component libraries was shown in the following picture:



### **Initialize GNAND Library**

To initialize GNAND library, just invoke **GNAND\_InitNAND()**. Application must give corresponding NAND driver as input argument to **GNAND\_InitNAND()**, then GNAND library can access NAND disk through NAND driver service.

GNAND library will validate the NAND disk is GNAND format or not. If it is not GNAND format, application can determine to program it as GNAND format or not. It depends on the third argument of **GNAND\_InitNAND()**.

### **GNAND work with Nuvoton FAT Library**

If **GNAND\_InitNAND()** returns **GNAND\_OK**, application can invoke **GNAND\_MountNandDisk()** to mount NAND disk to NVTFAT file system.

### **NAND driver function set**

To work as an underlying driver of GNAND, the NAND driver must provide the following function set and pass it to GNAND library with **GNAND\_InitNAND()**.

```

#define NDRV_T struct ndrv_t

struct ndrv_t
{
    INT (*init)(NDISK_T *NDInfo);
    INT (*pread)(INT nPBlockAddr, INT nPageNo, UINT8 *buff);
}
  
```

```

    INT (*pwrite)(INT nPBlockAddr, INT nPageNo, UINT8 *buff);

    INT (*is_page_dirty)(INT nPBlockAddr, INT nPageNo);

    INT (*is_valid_block)(INT nPBlockAddr);

    INT (*ioctl)(INT param1, INT param2, INT param3, INT param4);

    INT (*block_erase)(INT nPBlockAddr);

    INT (*chip_erase)(VOID);

    VOID *next;
};


```

In **init(NDISK\_T \*info)** function, NAND driver should detect NAND disk and fill NAND disk information into <NDISK\_T \*NDInfo>, which was passed as an argument. If success, return 0.

#### NDISKT\_T members

Member Name	Return by init()	Comments
vendor_ID	Optional	
device_ID	Optional	
NAND_type	Must	NAND_TYPE_SLC or NAND_TYPE_MLC
nZone	Must	Number of zones
nBlockPerZone	Must	Maximum number of physical blocks per zone
nPagePerBlock	Must	Number of pages per block
nLBPerZone	Must	Maximum number of allowed logical blocks per zone
nPageSize	Must	Page size in bytes
nStartBlock	Must	Reserved number of leading blocks
nBadBlockCount	Optional	Bad block count for all zones
driver	Must	NAND driver function set pointer
nNandNo	Optional	
pDisk	Optional	
reserved[60]	Ignore	
need2L2PN	Optional	Need second P2LN block or not
p2ln_block1	Optional	Physical block address for second P2LN block
p2lm	Ignore	GNAND internal used
l2pm	Ignore	GNAND internal used
dp_tbl	Ignore	GNAND internal used
db_idx[16]	Ignore	GNAND internal used
p2ln_block	Ignore	GNAND internal used

op_block	Ignore	GNAND internal used
op_offset	Ignore	GNAND internal used
last_op[32]	Ignore	GNAND internal used
err_sts	Ignore	GNAND internal used
next	Ignore	GNAND internal used

In **pread(INT nPBlockAddr, INT nPageNo, UINT8 \*buff)** function, NAND driver execute a page read operation from physical block <nPBlockAddr> page <nPageNo>. And <buff> was guaranteed to be non-cacheable memory.

In **pwrite(INT nPBlockAddr, INT nPageNo, UINT8 \*buff)** function, NAND driver execute a page programming operation to physical block <nPBlockAddr> page <nPageNo>. And <buff> was guaranteed to be non-cacheable memory.

In **is\_page\_dirty(INT nPBlockAddr, INT nPageNo)** function, NAND driver check the redundant area of physical block <nPBlockAddr> page <nPageNo>. If this page had ever been written, NAND driver should return 1, otherwise, return 0.

In **is\_valid\_block(INT nPBlockAddr)** function, NAND driver check if physical block <nPBlockAddr> is a valid block or not. If the block is a valid block, NAND driver should return 1, otherwise, return 0.

At current version, **ioctl()** was not used by GNAND library. NAND driver can give it a NULL value.

In **block\_erase(INT nPBlockAddr)** function, NAND driver execute a block erase operation on physical block <nPBlockAddr>.

In **chip\_erase()** function, NAND driver execute a chip erase operation on the NAND disk. Note that the whole GNAND information will lost after chip\_erase(). You have to call **GNAND\_InitNAND()** to rebuild GNAND format.

## 10.3. API Function

### ***GNAND\_InitNAND***

#### **Synopsis**

```
INT GNAND_InitNAND (NDRV_T *ndriver, NDISK_T *ptNDisk, BOOL  
bEraseIfNotGnandFormat)
```

#### **Description**

Initialize a NAND disk.

#### **Parameter**

ndriver	NAND driver function set to hook NAND driver on GNAND library.
ptNDisk	NAND disk information that GNAND initiated. You need this pointer to call other GNAND APIs.
bEraseIfNotGnandFormat	If NAND disk was GNAND format, ignore this argument. If NAND disk was not GNAND format, format it if this argument is 1, otherwise, return an GNERR_GNAND_FORMAT error.

#### **Return Value**

0	– Success
Otherwise	– error code defined in Error Code Table

#### **Example**

```
NDRV_T _nandDiskDriver0 =  
{  
    nandInit0,  
    nandpread0,  
    nandpwrite0,  
    nand_is_page_dirty0,  
    nand_is_valid_block0,  
    nand_ioctl,  
    nand_block_erase0,  
    nand_chip_erase0,
```

```
0
};

NDISK_T *ptNDisk;
int status;

fsInitFileSystem();

/* Initialize FMI */
sicIoctl(SIC_SET_CLOCK, 240000, 0, 0);
sicOpen();

ptNDisk = (NDISK_T *)malloc(sizeof(NDISK_T));
if (ptNDisk == NULL)
{
    printf("malloc error!!\n");
    return -1;
}

status = GNAND_InitNAND(&_nandDiskDriver0, ptNDisk, TRUE);
if (status < 0)
{
    printf("NAND disk init failed, status = %x\n", status);
    return status;
}

status = GNAND_MountNandDisk(ptNDisk);
if (status < 0)
{
```

```

    printf("Mount NAND disk failed, status = %x\n", status);

    return status;
}

```

## ***GNAND\_MountNandDisk***

### Synopsis

INT GNAND\_MountNandDisk (NDISK\_T \*ptNDisk)

### Description

Mount NAND disk to NVTFAT file system.

### Parameter

ptNDisk      The pointer refer to the NAND disk information that initiated by GNAND\_InitNAND().

### Return Value

0	– Success
Otherwise	– error code defined in Error Code Table

### Example

Refer to the example code of [GNAND\\_InitNAND\(\)](#);

## ***GNAND\_read***

### Synopsis

INT GNAND\_read (NDISK\_T \*ptNDisk, UINT32 nSectorNo, INT nSectorCnt, UINT8 \*buff)

### Description

Read logical sectors from NAND disk.

### Parameter

ptNDisk	The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().
nSectorNo	Read start sector number.
nSectorCnt	Number of sectors to be read.
buff	Memory buffer to receive data, which is 32 bytes aligned non-cacheable buffer.

### Return Value

0	– Success
---	-----------

Otherwise – error code defined in Error Code Table

#### Example

```
INT io_read(PDISK_T *pDisk, UINT32 sector_no, INT number_of_sector, UINT8 *buff)
{
    NDISK_T *ptNDisk = (NDISK_T *)pDisk->pvPrivate;
    return GNAND_read(ptNDisk, sector_no, number_of_sector, buff);
}
```

### ***GNAND\_write***

#### Synopsis

```
INT GNAND_write (NDISK_T *ptNDisk, UINT32 nSectorNo, INT nSectorCnt, UINT8
*buff)
```

#### Description

Write logical sectors to NAND disk

#### Parameter

ptNDisk	The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().
nSectorNo	Write start sector number.
nSectorCnt	Number of sectors to be written.
buff	Memory buffer to write data, which is 32 bytes aligned non-cacheable buffer

#### Return Value

0	– Success
Otherwise	– error code defined in Error Code Table

#### Example

```
INT io_write(PDISK_T *pDisk, UINT32 sector_no, INT number_of_sector, UINT8 *buff)
{
    NDISK_T *ptNDisk = (NDISK_T *)pDisk->pvPrivate;
    return GNAND_write(ptNDisk, sector_no, number_of_sector, buff);
}
```

## ***GNAND\_block\_erase***

### **Synopsis**

```
INT GNAND_block_erase (NDISK_T *ptNDisk, INT pba)
```

### **Description**

Erase a physical bock.

### **Parameter**

ptNDisk	The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().
pba	NAND physical block address.

### **Return Value**

0	– Success
Otherwise	– error code defined in Error Code Table

### **Example**

```
NDISK_T *ptNDisk;  
  
int status;  
  
  
/* erase physical block pba */  
  
status = GNAND_block_erase(ptNDisk, pba);  
  
if (status != 0)  
{  
  
/* handle error status */  
  
}
```

## ***GNAND\_chip\_erase***

### **Synopsis**

```
INT GNAND_chip_erase (NDISK_T *ptNDisk)
```

### **Description**

This function erase all blocks in NAND chip. All data in chip will lost that include information for GNAND library.

### **Parameter**

**ptNDisk**      The pointer refer to the NAND disk information that initiated by GNAND\_InitNAND().

#### Return Value

0	– Success
Otherwise	– error code defined in Error Code Table

#### Example

```
NDISK_T *ptNDisk;

int status;

/* erase whole NAND chip */

status = GNAND_chip_erase(ptNDisk, pba);

if (status != 0)

{

/* handle error status */

}
```

## ***GNAND\_UnMountNandDisk***

#### Synopsis

VOID GNAND\_UnMountNandDisk (NDISK\_T \*ptNDisk)

#### Description

Unmount NAND disk from NVTFAT file system.

#### Parameter

**ptNDisk**      The pointer refer to the NAND disk information that initiated by GNAND\_InitNAND().

#### Return Value

0	– Success
Otherwise	– error code defined in Error Code Table

#### Example

```
NDISK_T *ptNDisk;

int status;

status = GNAND_UnMountNandDisk(ptNDisk);
```

```
if (status != 0)
{
    /* handle error status */
}
```

## 10.4. Example code

The demo code test the GNAND library please refer to the SIC sample code of SDK Non-OS.

## 10.5. Error Code Table

CODE NAME	Value	Description
GNAND_OK	0	Success
GNERR_GENERAL	0xFFFFC001	General access error
GNERR_MEMORY_OUT	0xFFFFC005	No available memory
GNERR_GNAND_FORMAT	0xFFFFC010	NAND disk was not GNAND format
GNERR_FAT_FORMAT	0xFFFFC015	NAND disk was unformatted as FAT
GNERR_BLOCK_OUT	0xFFFFC020	There's no available physical blocks
GNERR_P2LN_SYNC	0xFFFFC025	Internal error for P2LN table sync problem
GNERR_READONLY_NAND	0xFFFFC026	Cannot write data into readonly NAND disk
GNERR_IO_ERR	0xFFFFC030	NAND read/write/erase access failed
GNERR_NAND_NOT_FOUND	0xFFFFC040	NAND driver cannot find NAND disk.
GNERR_UNKNOW_ID	0xFFFFC042	Not supported NAND disk type

# 11. GPIO Library Overview

The GPIO library provides a set of APIs to control on-chip GPIO pins. This library depends on N9H26 System Library.

## 11.1. API Functions

### ***gpio\_open***

#### **Synopsis**

```
int gpio_open (unsigned char port)
```

#### **Description**

It has replaced `gpio_open (unsigned char port)` with `gpio_configure (unsigned char port, unsigned short num)`.

### ***gpio\_configure***

#### **Synopsis**

```
int gpio_configure (unsigned char port, unsigned short num)
```

#### **Description**

This function configures the specified pin of a port as GPIO.

#### **Parameter**

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

num     pin number

#### **Return Value**

Return 0 on success. -1 for unknown port number

#### **Example**

```
/* Configure the pin0 of portD as GPIO*/
gpio_configure (GPIO_PORTD, 0);
```

## gpio\_readport

### Synopsis

```
int gpio_readport (unsigned char port, unsigned short *val)
```

### Description

This function reads back all pin value of a GPIO port, ignore the direction of each pin.

### Parameter

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

\*val     Return port value

### Return Value

Return 0 one success, -1 for unknown port number

### Example

```
/* Read PORTC value*/  
  
unsigned short val;  
  
gpio_readport(GPIO_PORTC, &val);
```

## gpio\_setportdir

### Synopsis

```
int gpio_setportdir (unsigned char port, unsigned short mask, unsigned short dir)
```

### Description

This function sets the pin direction of GPIO port. It could select the pin(s) to be configured with its second parameter.

### Parameter

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

mask     pin mask, each bit stands for one pin

dir     Direction, each bit configures one pin, 0 means input, 1 means output

### Return Value

Return 0 one success, -1 for unknown port number

### Example

```
/* Set PORTC pin1 to output mode, and pin0 to input mode */
```

```
gpio_setportdir (GPIO_PORTC, 0x3, 0x2);
```

## gpio\_setportval

### Synopsis

```
int gpio_setportval (unsigned char port, unsigned short mask, unsigned short val)
```

### Description

This function sets the output value of GPIO port. It could select the pin(s) to be configured with its second parameter.

### Parameter

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

mask    pin mask, each bit stands for one pin

val     Output value, each bit configures one pin, 0 means low, 1 means high

### Return Value

Return 0 one success, -1 for unknown port number

### Example

```
/* Set PORTC pin1 to output high, and pin0 to low */  
  
gpio_setportval (GPIO_PORTC, 0x3, 0x2);
```

## gpio\_setportpull

### Synopsis

```
int gpio_setportpull (unsigned char port, unsigned short mask, unsigned short pull)
```

### Description

This function sets the pull up/down resistor of GPIO port. It could select the pin(s) to be configured with its second parameter.

### Parameter

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

mask    pin mask, each bit stands for one pin

pull    Pull up/down resister state, each bit configures one pin, 0 means disable, 1 means enable

### Return Value

Return 0 one success, -1 for unknown port number

**Example**

```
/* Enable PORTC pin1 pull up resistor, and disable pin0 pull up resistor */  
gpio_setportpull (GPIO_PORTC, 0x3, 0x2);
```

**gpio\_setdebounce****Synopsis**

```
int gpio_setdebounce (unsigned char clk, unsigned char src)
```

**Description**

This function is used to configure external interrupt de-bounce time.

**Parameter**

clk      Debounce sampling clock, could be 1, 2, 4, 8, 16, 32, 64, 128, 256, 2\*256, 4\*256, 8\*256, 16\*256, 32\*256, 64\*256 and 128\*256

src      Debounce sampling interrupt source. Valid values are between 0~15. Each bit represents one interrupt source

**Return Value**

Return 0 on success, -1 on parameter error

**Example**

```
/* Set nIRQ0 debounce sampling clock to 128 clocks*/  
gpio_setdebounce (128, 1);
```

**gpio\_getdebounce****Synopsis**

```
void gpio_getdebounce (unsigned char *clk, unsigned char *src)
```

**Description**

This function gets current external interrupt de-bounce time setting.

**Parameter**

\*clk      Debounce sampling clock

\*src      Debounce sampling interrupt source

**Return Value**

None

**Example**

```
unsigned char clk;  
unsigned char src;  
gpio_getdebounce (&clk, &src);
```

## gpio\_setsrcgrp

### Synopsis

```
int gpio_setsrcgrp (unsigned char port, unsigned short mask, unsigned char irq)
```

### Description

This function is used to set external interrupt source group.

### Parameter

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH  
  
mask     pin mask, each bit stands for one pin  
  
irq      external irq number. Could be 0~3

### Return Value

Return 0 on success, -1 on parameter error

### Example

```
/* Set GPIO port C pin1 as source of nIRQ3 */  
  
gpio_setsrcgrp (GPIO_PORTC, 1, 3);
```

## gpio\_getsrcgrp

### Synopsis

```
int gpio_getsrcgrp (unsigned char port, unsigned int *val)
```

### Description

This function is used to get current external interrupt source setting.

### Parameter

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH  
  
\*val     Current source setting. Every two bits stands for the interrupt source each pin triggers

### Return Value

Return 0 on success, and -1 for unknown port number

**Example**

```
/* Read GPIO port C interrupt group status */

unsigned int val;

gpio_setsrcgrp (GPIO_PORTC, &val);
```

**gpio\_setintmode****Synopsis**

```
int gpio_setintmode (unsigned char port, unsigned short mask, unsigned short falling, unsigned short rising)
```

**Description**

This function sets the interrupt trigger mode of GPIO port. It could select the pin(s) to be configured with its second parameter.

**Parameter**

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

mask    Pin mask, each bit stands for one pin

falling Triggers on falling edge, each bit stands for one pin

rising Triggers on rising edge, each bit stands for one pin

**Return Value**

Return 0 on success, -1 for parameter error

**Example**

```
/* Set PORT C pin 1 triggers on both falling and rising edge */

gpio_setintmode (GPIO_PORTC, 1, 1, 1);
```

**gpio\_getintmode****Synopsis**

```
int gpio_getintmode (unsigned char port, unsigned short *falling, unsigned short *rising)
```

**Description**

This function is used to get interrupt trigger mode of GPIO port.

**Parameter**

port     GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
        GPIO\_PORTG, and GPIO\_PORTH

\*falling Triggers on falling edge, each bit stands for one pin  
\*rising Triggers on rising edge, each bit stands for one pin

#### Return Value

Return 0 on success, -1 for parameter error

#### Example

```
/* Get PORT C trigger mode */  
  
unsigned short falling;  
  
unsigned short rising;  
  
gpio_getintmode (GPIO_PORTC, &falling, &rising);
```

## gpio\_setlatchtrigger

#### Synopsis

```
int gpio_setlatchtrigger (unsigned char src)
```

#### Description

This function used to set latch trigger source.

#### Parameter

src Latch trigger source. Each bit stands for one external interrupt source. If the value is 1, GPIO port input value will be latched while interrupt triggers

#### Return Value

Return 0 on success, -1 for parameter error

#### Example

```
/* Enable latch for nIRQ0 and nIRQ3*/  
  
gpio_setlatchtrigger (9);
```

## gpio\_getlatchtrigger

#### Synopsis

```
void gpio_getlatchtrigger (unsigned char *src)
```

#### Description

This function used to get latch trigger source.

#### Parameter

\*src Latch trigger source

#### Return Value

None

#### Example

```
/* Get latch trigger source*/  
  
unsigned char src;  
  
gpio_getlatchtrigger (&src);
```

## gpio\_getlatchval

#### Synopsis

```
int gpio_getlatchval (unsigned char port, unsigned short *val)
```

#### Description

This function is used to get interrupt latch value.

#### Parameter

port GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE, GPIO\_PORTG, and GPIO\_PORTH

\*val Variable to store latch value

#### Return Value

Return 0 on success, -1 for parameter error

#### Example

```
/* Get port C latch value */  
  
unsigned short val;  
  
gpio_getlatchval (GPIO_PORTC, &val);
```

## gpio\_gettriggersrc

#### Synopsis

```
int gpio_gettriggersrc (unsigned char port, unsigned short *src)
```

#### Description

This function is used to get interrupt trigger source.

#### Parameter

port GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
GPIO\_PORTG, and GPIO\_PORTH

\*src Variable to store trigger source

#### Return Value

Return 0 on success, -1 for parameter error

#### Example

```
/* Get port C interrupt trigger source */  
  
unsigned short src;  
  
gpio_gettriggersrc (GPIO_PORTC, &src);
```

## gpio\_cleartriggersrc

#### Synopsis

```
int gpio_cleartriggersrc(unsigned char port)
```

#### Description

This function is used to clear interrupt trigger source.

#### Parameter

port GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, GPIO\_PORTE,  
GPIO\_PORTG, and GPIO\_PORTH

#### Return Value

Return 0 on success, -1 for parameter error

#### Example

```
/* Clear port C interrupt trigger source */  
  
gpio_cleartriggersrc (GPIO_PORTC);
```

# 12. H264 Codec Library Introduction

This document is written for user applications which want to use H264 Encoder or Decoder library API.

## 12.1. Feature

### Encoder Features

- Follows MPEG-4 AVC/JVT/H.264 (ISO/IEC 14496-10) video coding standards
- Supports baseline profile to level 3.1
- Supports resolutions from 128x80 to 1280x720 in a step of 16 units
- Supports I and P frame encodings
- CBR and VBR rate controls by firmware
- Supports programmable in-loop filter parameters
- Supports programmable chroma QP index offset parameter

### Decoder Features.

- Compliant with ITU-T Recommendation H.264|ISO/IEC 14496-10 Advanced Video Coding Standard (MPEG 4 Part 10)
- Supports baseline profile with a level from 1 to 3
- Supports resolutions of up to 720 x 480 at 60 fps
- Supports motion estimation with variable block sizes
- Supports quarter-pixel motion compensation
- Supports Context-based Adaptive Variable-Length Decoding (CAVLD)
- Supports I and P slices
- Supports in-loop de-blocking filter function (`disable_deblocking_filter_idc != 1`) to execute filtering function, including slice boundary
- Not supports Arbitrary Slice Order (ASO) or Flexible Macroblock Ordering (FMO)

## 12.2. Rate Control for Encoder

The encoded bitstream rate control is implemented in application level. At the beginning, the application needs to call `H264RateControlInit(...)` function to initialize the variable in `H264RateControl` structure. After that, `H264RateControlUpdate(...)` function is used to calculate the next Quant value for next encode frame which is stored in the `rtn_quant` field of `H264RateControl` structure. Specify this `rtn_quant` in `u32Quant` filed of `FAVC_ENC_PARAM` structure for next encode frame. Then, the rate control will be completed in such kind of loop.

## 12.3. API Data Structure

### **IOCTL COMMAND**

Ioctl command set for H264\_ioctl function

Name	Value	Description
<i>FAVC_IOCTL_DECODE_INIT</i>	<b>0x4170</b>	Init H264 decoder
<i>FAVC_IOCTL_DECODE_FRAME</i>	<b>0x4172</b>	Decode one H264 frame
<i>FAVC_IOCTL_ENCODE_INIT</i>	<b>0x4173</b>	Init H264 Encode
<i>FAVC_IOCTL_ENCODE_FRAME</i>	<b>0x4175</b>	Encode one H264 frame
<i>FAVC_IOCTL_GET_SPSPPS</i>	<b>0x4179</b>	Get the encoded SPS PPS bitstream

### **FAVC\_DEC\_RESULT**

H264 bitstream decoding result.

Name	Type	Description
<i>bEndOfDec</i>	UINT32	Used by library only
<i>u32Width</i>	UINT32	Decoded bitstream width
<i>u32Height</i>	UINT32	Decoded bitstream height
<i>u32UsedBytes</i>	UINT32	Reported used bitstream byte in buffer
<i>u32FrameNum</i>	UINT32	Decoded frame number
<i>isDisplayOut</i>	UINT32	0 -> Buffer in reorder buffer, 1 -> available buffer, -1 -> last flush frame
<i>isISlice</i>	UINT32	1-> I Slice, 0 -> P slice
<i>Reserved0</i>	UINT32	reserved

### **FAVC\_DEC\_PARAM**

H264 bitstream decoding paramter.

Name	Type	Description
<i>u32API_version</i>	UINT32	API version
<i>u32MaxWidth</i>	UINT32	Not used now
<i>u32MaxHeight</i>	UINT32	Decoded bitstream height
<i>u32FrameBufferWidth</i>	UINT32	if (u32FrameBufferWidth != -1), decoded image width is cropped with u32FrameBufferWidth if (u32FrameBufferWidth == -1),

		decoded image width is continued on memory
<i>u32FrameBufferHeight</i>	UINT32	if (u32FrameBufferHeight != -1), decoded image height is cropped with u32FrameBufferHeight if (u32FrameBufferHeight == -1), decoded image height is continued on memory
<i>u32Pkt_size</i>	UINT32	Current decoding bitstream length (the exact bitstream length for one frame)
<i>pu8Pkt_buf</i>	UINT8*	Current decoding bitstream buffer address (application ready bitstream here)
<i>pu8Display_addr[3]</i>	UINT32	Buffer address for decoded data
<i>got_picture</i>	UINT32	0 -> Decoding has something error. 1 -> decoding is OK in current bitstream
<i>pu8BitStream_phys</i>	UINT8*	physical address. buffer for bitstream (allocated and used by library only)
<i>u32OutputFmt</i>	UINT32	Decoded output format, 0-> Planar YUV420 format, 1-> Packet YUV422 format
<i>crop_x</i>	UINT32	pixel unit: crop x start point at decoded-frame (not supported now)
<i>crop_y</i>	UINT32	pixel unit: crop y start point at decoded-frame (not supported now)
<i>tResult</i>	FAVC_DEC_RESULT	Return decoding result by library

**FAVC\_ENC\_PARAM**

H264 encoding parameter.

Name	Type	Description
<i>u32API_version</i>	UINT32	API version
<i>u32BitRate</i>	UINT32	The encoded bitrate in bps.
<i>u32FrameWidth</i>	UINT32	The width of encoded frame in pels.
<i>u32FrameHeight</i>	UINT32	The height of encoded frame in pels
<i>fFrameRate</i>	UINT32	The base frame rate per second
<i>u32IPIInterval</i>	UINT32	The frame interval between I-frames.
<i>u32MaxQuant</i>	UINT32	The maximum quantization value. (max = 51)
<i>u32MinQuant</i>	UINT32	The minimum quantization value. (min=0)

<i>u32Quant</i>	UINT32	The frame quantization value for initialization
<i>ssp_output</i>	INT32	This variable tells the H.264 must be encoded out sps + pps before slice data. -> 1 : force the encoder to output sps+pps -> 0 : force the encoder to output sps+pps on any Slice I frame -> -1: (default) only output SPS+PPS on first IDR frame.
<i>intra</i>	INT32	This variable tells the H.264 must be encoded out an I-Slice type frame. -> 1 : forces the encoder to create a keyframe. -> 0 : forces the encoder not to create a keyframe. -> -1: (default) let the encoder decide (based on contents and u32IPIInterval)
<i>bROIEnable</i>	INT32	To enable the function of encoding rectangular region of interest(ROI) within captured frame
<i>u32ROIx</i>	UINT32	The upper-left corner x coordinate of rectangular region of interest
<i>u32ROIy</i>	UINT32	The upper-left corner coordinate y of region of interest
<i>u32ROIWidth</i>	UINT32	The width of user-defined rectangular region of interest
<i>u32ROIHeight</i>	UINT32	The height of user-defined rectangular region of interest
<i>pu8YFrameBaseAddr</i>	UINT8*	The base address for input Y frame buffer
<i>pu8UVFrameBaseAddr</i>	UINT8*	The base address for input UV frame buffer in H.264 2D mode
<i>pu8UFrameBaseAddr</i>	UINT8*	The base address for input U frame buffer
<i>pu8VFrameBaseAddr</i>	UINT8*	The base address for input V frame buffer
<i>bitstream</i>	UINT8*	Bitstream Buffer address for driver to write bitstream
<i>pu8BitstreamAddr</i>	UINT8*	The bitstream buffer address while encoding one single frame allocated by librar
<i>bitstream_size</i>	UINT32	Bitstream length for current frame
<i>keyframe</i>	UINT32	This parameter is indicated the Slice type of frame
<i>frame_cost</i>	UINT32	frame_cout is updated by driver
<i>no_frames</i>	UINT32	The number of frames to be encoded
<i>threshold_disable</i>	UINT32	The transform coefficients threshold
<i>chroma_threshold</i>	UINT32	The chroma coefficients threshold (0 ~ 7)
<i>luma_threshold</i>	UINT32	The luma coefficients threshold (0 ~ 7)

<i>beta_offset</i>	UINT32	The beta offset for in-loop filter.
<i>alpha_offset</i>	UINT32	The alpha offset for in-loop filter.
<i>chroma_qp_offset</i>	UINT32	The chroma qp offset (-12 to 12 inclusively)
<i>disable_ilf</i>	UINT32	To disable in-loop filter or not
<i>watermark_enable</i>	UINT32	To enable watermark function or not (Don't enable it now)
<i>watermark_interval</i>	UINT32	To specify the watermark interval if watermark function is enabled
<i>watermark_init_pattern</i>	UINT32	To specify the initial watermark pattern if watermark function is enabled
<i>pu8ReConstructFrame</i>	UINT8*	The address of reconstruct frame buffer.
<i>pu8ReferenceFrame</i>	UINT8*	The address of reference frame buffer
<i>pu8SysInfoBuffer</i>	UINT8*	The address of system info buffer
<i>pu8DMABuffer_virt</i>	UINT8*	The physical address of DMA buffer
<i>nvop_ioctl</i>	INT32	This parameter is valid only on FAVC_IOCTL_ENCODE_NVOP
<i>multi_slice</i>	UINT32	Multi-slice mode
<i>pic_height</i>	UINT32	This parameter is used to keep the frame height for sps and pps on Multi Slice mode
<i>pic_width</i>	UINT32	This parameter is used to keep the frame width for sps and pps on Multi Slice mode
<i>img_fmt</i>	UINT32	0: 2D format, CbCr interleave, named H264_2D (VideoIn supported only)
<i>control</i>	UINT32	0 : Do NOT force one frame as one slice(default), 1 : Force one frame as one slice

---

## 12.4. API Function

### *H264Dec\_Open*

#### Synopsis

```
int H264Dec_open(void);
```

#### Description

Initialize H264 Decoder and install interrupt service routine.

#### Parameter

None

#### Return Value

1	Success
-1	Fail

## ***H264Enc\_Open***

### **Synopsis**

```
int H264Enc_Open(void);
```

### **Description**

Initialize H264 Encoder and install interrupt service routine.

### **Parameter**

None

### **Return Value**

1	Success
-1	Fail

## ***H264\_ioctl***

### **Synopsis**

```
int H264_ioctl(int cmd, void* param);
```

### **Description**

Perform the H264 encoder/decoder related operation

### **Parameter**

cmd	Specify the operation for Encoder or Decoder which is defined in ioctl command set.
param	The pointer to FAVC_ENC_PARAM or FAVC_DEC_PARAM dependent on cmd

### **Return Value**

0	Success
-1	Fail

## ***H264Enc\_Close***

### **Synopsis**

```
void H264Enc_Close(void);
```

### **Description**

Close H264 Encoder and free related buffer allocation.

**Parameter**

None

**Return Value**

None

***H264Dec\_Close*****Synopsis**

```
void H264Dec_Close (void);
```

**Description**

Close H264 Decoder and free related buffer allocation.

**Parameter**

None

**Return Value**

None

***nv\_malloc*****Synopsis**

```
void* nv_malloc(int size, int alignment);
```

**Description**

Allocate memory in size which is alignment.

**Parameter**

size	specify the allocated memory size
alignment	specify the allocated memory alignment

**Return Value**

Pointer to allocated memory

***nv\_free*****Synopsis**

```
int nv_free(void* ptr);
```

**Description**

Free the memory specified by ptr.

**Parameter**

ptr pointer to memory which is to free.

**Return Value**

0 Success

---

## 12.5. Retun Code Table

### **AVC\_RET**

API return value for decoder

Name	Value	Description
<i>RETCODE_OK</i>	0	
<i>RETCODE_ERR_MEMORY</i>	1	
<i>RETCODE_ERR_API</i>	2	
<i>RETCODE_ERR_HEADER</i>	3	
<i>RETCODE_ERR_FILL_BUFFER</i>	4	
<i>RETCODE_ERR_FILE_OPEN</i>	5	
<i>RETCODE_HEADER_READY</i>	6	
<i>RETCODE_BS_EMPTY</i>	7	
<i>RETCODE_WAITING</i>	8	
<i>RETCODE_DEC_OVERFLOW</i>	9	
<i>RETCODE_HEADER_FINISH</i>	10	
<i>RETCODE_DEC_TIMEOUT</i>	11	
<i>RETCODE_PARSING_TIMEOUT</i>	12	
<i>RETCODE_ERR_GENERAL</i>	13	
<i>RETCODE_NOT_SUPPORT</i>	14	
<i>RETCODE_FAILURE</i>	15	
<i>RETCODE_FRAME_NOT_COMPLETE</i>	16	

# 13. I2C Library Overview

This library provides APIs for programmers to access I2C slaves connecting with N9H26 I2C interfaces. The default clock frequency is configured at 100 kHz after i2cOpen() is called, programmers could use i2cIoctl() function to change the frequency.

The maximum receive/transmit buffer length of this library is 450 bytes, which includes slave address and sub address. Data beyond this range will be ignored.

The I2C library will get the APB clock frequency from system library, application must set the CPU clock before using I2C library.

---

## 13.1. I2C Library APIs Specification

### *i2cInit*

#### Synopsis

INT32 i2cInit(VOID)

#### Description

This function configures GPIO to I2C mode.

#### Parameter

None

#### Return Value

0 Always successes

#### Example

```
i2cInit();
```

### *i2cOpen*

#### Synopsis

INT32 i2cOpen(VOID)

#### Description

This function initializes the software resource, enables I2C engine clock and sets the clock frequency to 100 kHz.

**Parameter**

None

**Return Value**

0	Successful
I2C_ERR_BUSY	Interface already opened

**Example**

```
INT32 status;  
  
status = i2cOpen();
```

**i2cClose****Synopsis**

INT32 i2cClose(VOID)

**Description**

This function disables I2C engine clock.

**Parameter**

None

**Return Value**

0	Successful
---	------------

**Example**

```
i2cClose();
```

**i2cRead****Synopsis**

INT32 i2cRead(PUINT8 buf, UINT32 len)

**Description**

This function reads data from I2C slave.

**Parameter**

buf	Receive buffer pointer
len	Receive buffer length

**Return Value**

> 0	Return read length on success
I2C_ERR_BUSY	Interface busy
I2C_ERR_IO	Interface not opened
I2C_ERR_NACK	Slave returns an erroneous ACK
I2C_ERR_LOSTARBITRATION	Arbitration lost during transmission

**Example**

```
UCHAR8 buf[8];

INT32 len = 0;

len = i2cRead(buf, 8); // Read 8 bytes from i2c slave
```

**i2cRead\_OV****Synopsis**

```
INT32 i2cRead_OV(PUINT8 buf, UINT32 len)
```

**Description**

This function reads data from OmniVision sensor.

**Parameter**

buf	Receive buffer pointer
len	Receive buffer length

**Return Value**

> 0	Return read length on success
I2C_ERR_BUSY	Interface busy
I2C_ERR_IO	Interface not opened
I2C_ERR_NACK	Slave returns an erroneous ACK
I2C_ERR_LOSTARBITRATION	Arbitration lost during transmission

**Example**

```
UCHAR8 buf[1];

INT32 len = 0;

len = i2cRead_OV(buf, 1); // Read one bytes from OmniVision sensor
```

***i2cWrite*****Synopsis**

```
INT32 i2cWrite(PUINT8 buf, UINT32 len)
```

**Description**

This function writes data to I2C slave.

**Parameter**

- |     |                         |
|-----|-------------------------|
| buf | Transmit buffer pointer |
| len | Transmit buffer length  |

**Return Value**

> 0	Return writes length on success
I2C_ERR_BUSY	Interface busy
I2C_ERR_IO	Interface not opened
I2C_ERR_NACK	Slave returns an erroneous ACK
I2C_ERR_LOSTARBITRATION	Arbitration lost during transmission

**Example**

```
UINT8 buf [5] = {0x00, 0x01, 0x02, 0x03, 0x04};

UINT32 len;

len = i2cWrite(buf, 5); // Write 5 bytes to I2C slave
```

***i2cIoctl*****Synopsis**

```
INT32 i2cIoctl(UINT32 cmd, UINT32 arg0, UINT32 arg1)
```

**Description**

This function allows programmers configure I2C interface, the supported command and arguments listed in the table below.

Command	Argument 0	Argument 1	Description
I2C_IOC_SET_DEV_ADDRESS	Unsigned integer stores the slave address	Not used	This command sets the slave address
I2C_IOC_SET_SPEED	Unsigned integer stores the new frequency	Not used	Valid clock frequencies are 100 kHz and 400 kHz
I2C_IOC_SET_SUB_ADDRESS	Unsigned integer stores the sub address	Sub-address length	This command sets the sub-address and its length
I2C_IOC_SET_SINGLE_MASTER	Enable single master mode	Not used	This command enable/disable single master mode

**Parameter**

- cmd Command
- arg0 First argument of the command
- arg1 Second argument of the command

**Return Value**

- |               |   |
|---------------|---|
| 0             | On Success                              |
| I2C_ERR_IO    | Interface not activated                 |
| I2C_ERR_NOTTY | Command not support, or parameter error |

**Example**

```
/* Set clock frequency to 400 kHz */

i2cIoctl(I2C_IOC_SET_SPEED, 400, 0);
```

***i2cExit*****Synopsis**

```
INT32 i2cExit(VOID)
```

**Description**

This function does nothing.

**Parameter**

None

**Return Value**

- |   |                   |
|---|-------------------|
| 0 | Always successful |
|---|-------------------|

**Example**

```
i2cExit();
```

---

**13.2. Error Code Table**

Code Name	Value	Description
I2C_ERR_LOSTARBITRATION	0xFFFF1101	Arbitration lost during transmission
I2C_ERR_NACK	0xFFFF1103	Slave returns an erroneous ACK
I2C_ERR_SLAVENACK	0xFFFF1104	slave not respond after address

I2C_ERR_NODEV	0xFFFF1105	Interface number out of range
I2C_ERR_BUSY	0xFFFF1106	Interface busy
I2C_ERR_IO	0xFFFF1107	Interface not activated
I2C_ERR_NOTTY	0xFFFF1108	Command not support, or parameter error

# 14. I2S Library Description

This library provides APIs for programmers to play/record PCM audio data from I2S engine.

## 14.1. API Functions

### *DrvI2S\_Open*

#### Synopsis

```
VOID DrvI2S_Open(VOID)
```

#### Description

This function will open I2S pins and engine clock.

#### Parameter

None

#### Return Value

None

#### Example

```
DrvI2S_Open();
```

### *DrvI2S\_Close*

#### Synopsis

```
VOID DrvI2S_Close(VOID)
```

#### Description

This function will close I2S pins and engine clock.

#### Parameter

None

#### Return Value

None

#### Example

```
DrvI2S_Close();  
  
VOID DrvI2S_StartPlay (  
    S_DRV12S_PLAY* psPlayStruct  
)
```

### ***DrvI2S\_StartPlay***

#### Synopsis

```
VOID DrvI2S_StartPlay(S_DRV12S_PLAY* psPlayStruct)
```

#### Description

After opening I2S pins and engine clock, this function will trigger I2S engine to start playing.

#### Parameter

psPlayStruct      Structure pointer for Play related parameters

#### Return Value

None

#### Example

```
DrvI2S_StartPlay((S_DRV12S_PLAY*) &g_sPlay);
```

### ***DrvI2S\_StopPlay***

#### Synopsis

```
VOID DrvI2S_StopPlay (VOID)
```

#### Description

Stop playing.

#### Parameter

None

#### Return Value

None

#### Example

```
DrvI2S_StopPlay();
```

## ***DrvI2S\_StartRecord***

### **Synopsis**

```
VOID DrvI2S_StartRecord(S_DRV I2S_RECORD* psRecordStruct)
```

### **Description**

After opening I2S pins and engine clock, this function will trigger I2S engine to start recording.

### **Parameter**

psRecordStruct Structure pointer for Record related parameters

### **Return Value**

None

### **Example**

```
DrvI2S_StartRecord((S_DRV I2S_RECORD*) &g_sRecord);
```

## ***DrvI2S\_StopRecord***

### **Synopsis**

```
VOID DrvI2S_StopRecord (VOID)
```

### **Description**

Stop recording.

### **Parameter**

None

### **Return Value**

None

### **Example**

```
DrvI2S_StopRecord();
```

## ***DrvI2S\_SetSampleRate***

### **Synopsis**

```
VOID DrvI2S_SetSampleRate(E_DRV I2S_SAMPLING eSamplaeRate)
```

### **Description**

Set Play/Record sampling rate.

**Parameter**

eSampleRate     Given sampling rate.

**Return Value**

None

**Example**

```
DrvI2S_SetSampleRate((E_DRVI2S_SAMPLING) eDRV12S_FREQ_44100);
```

# 15. JPEG Library Overview

N9H26 Non-OS library consists of a set of libraries. These libraries are built to access those on-chip functions such as VPOST, APU, SIC, USBH, USBD, GPIO, I2C, SPI and UART, as well as File System (NVTFS), USB Mass Storage devices (UMAS) and NAND Flash devices (GNAND). This document describes the provided APIs of JPEG library. With these APIs, user can quickly build a binary target for JPEG library on N9H26 micro processor.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

---

## 15.1. JPEG Overview

This library is designed to make user application to use N9H26 JPEG more easily.

The JPEG library has the following features:

- JPEG Normal / Encode function
- JPEG Encode Upscale function
- JPEG Decode Downscale function
- JPEG Window Decode function
- JPEG Decode Input Wait function
- JPEG Decode Output Wait function

---

## 15.2. Programming Guide

### System Overview

The JPEG Codec supports Baseline Sequential Mode JPEG still image compression and decompression that is fully compliant with ISO/IEC International Standard 10918-1 (T.81). The features and capability of the JPEG codec are listed below.

#### JPEG Features

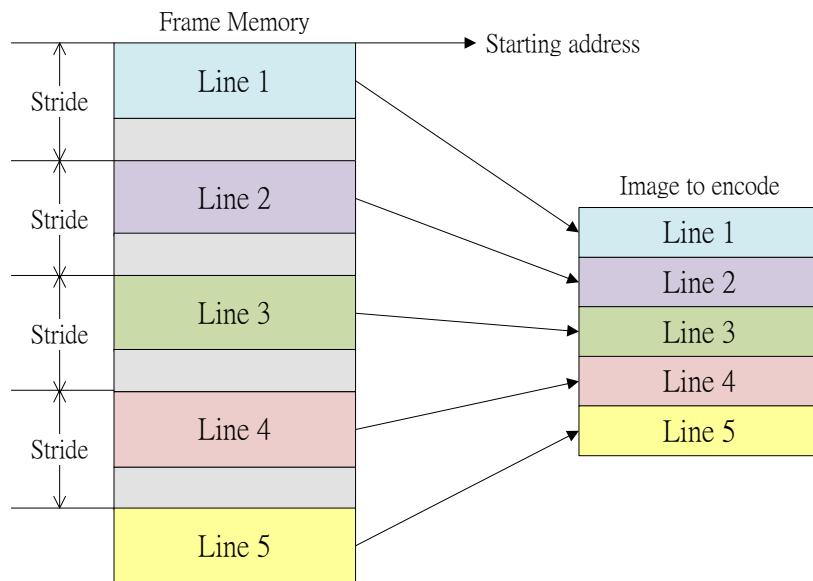
- Support to encode interleaved YCbCr 4:2:2/4:2:0 and gray-level (Y only) format image
- Support to decode interleaved YCbCr 4:4:4/4:4:2/2:4:2:0/4:1:1 and gray-level (Y only) format image
- Support to decode YCbCr 4:2:2 transpose format
- The encoded JPEG bit-stream format is fully compatible with JFIF and EXIF standards

- Support Capture and JPEG hardware on-the-fly access mode for encode
- Support JPEG and Playback hardware on-the-fly access mode for decode
- Support software input/output on-the-fly access mode for both encode and decode
- Support arbitrary width and height image encode and decode
- Support three programmable quantization-tables
- Support standard default Huffman-table and programmable Huffman-table for decode
- Support arbitrarily 1X~8X image up-scaling function for encode mode
- Support down-scaling function for encode and decode modes
- Support specified window decode mode
- Support quantization-table adjustment for bit-rate and quality control in encode mode
- Support rotate function in encode mode

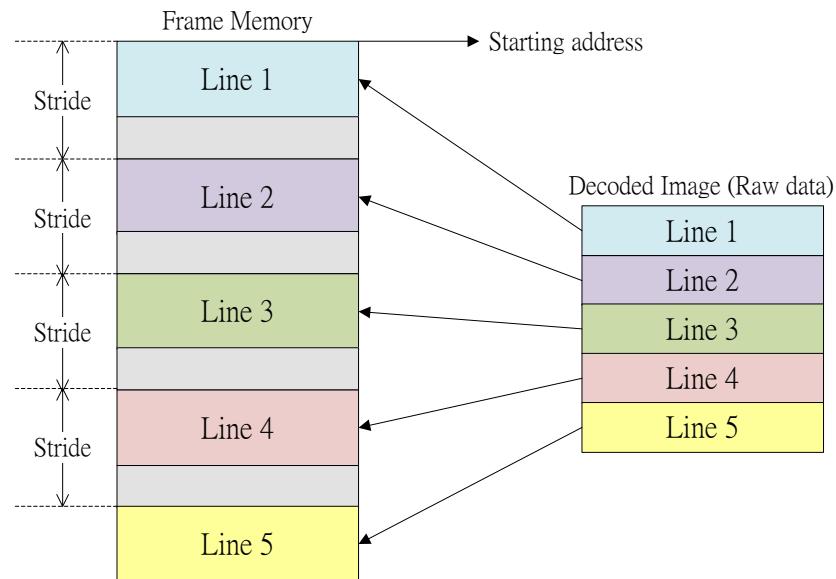
## JPEG Operation Control

### ■ Memory access

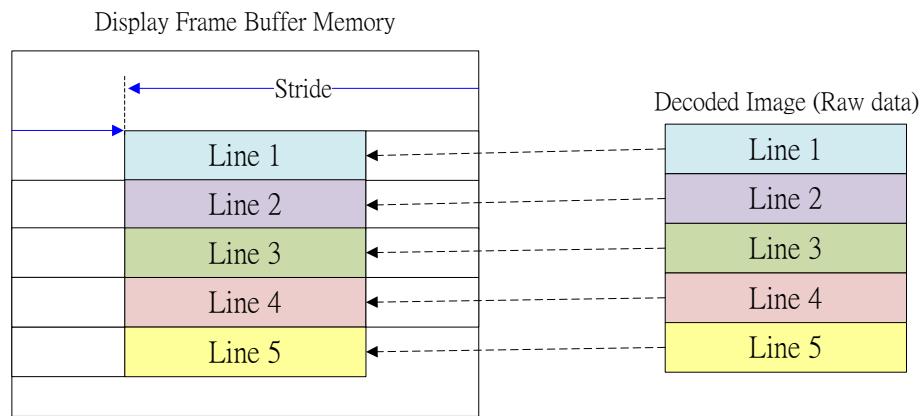
The following figure shows the encode mode to access the source data which are from sensor normally and stored on the SDRAM.



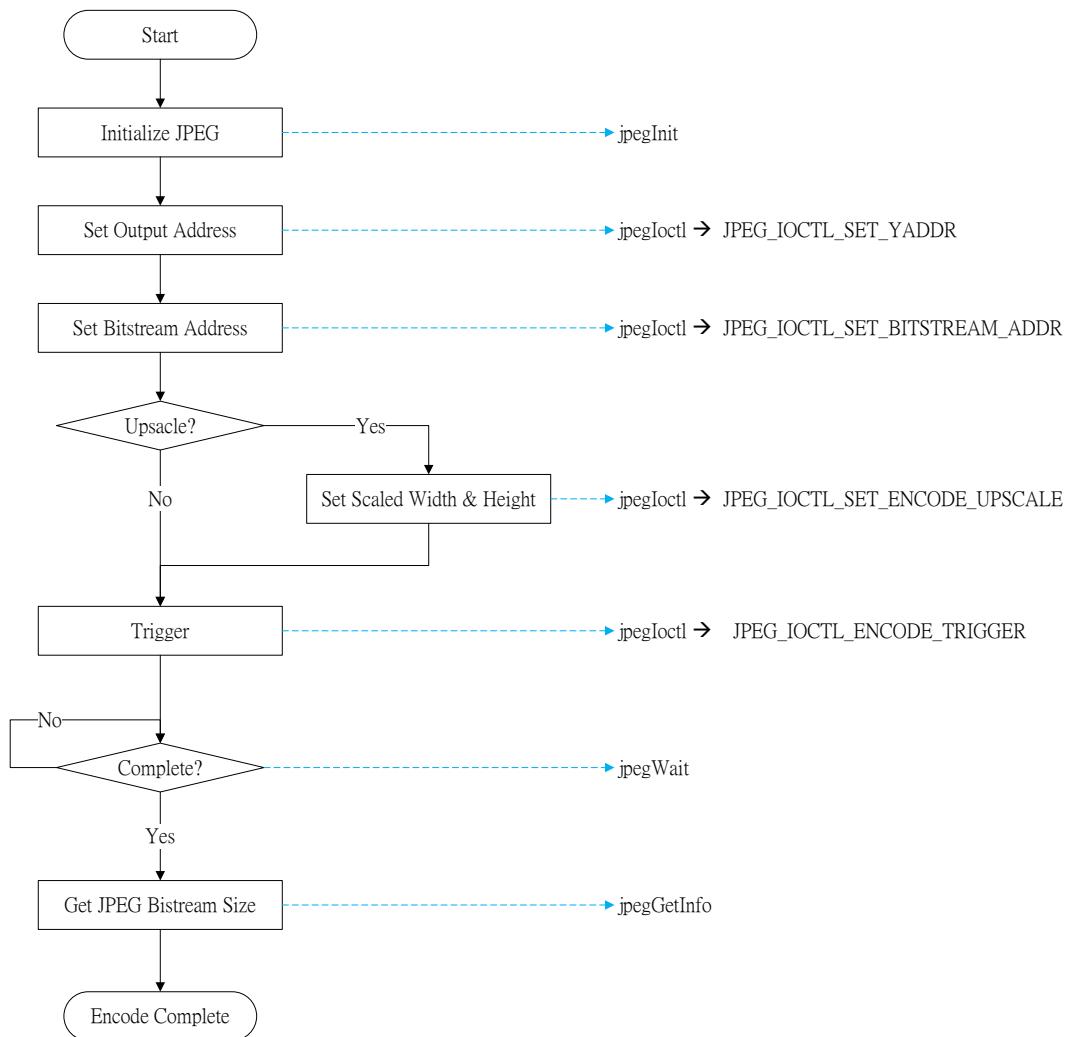
Following figure shows the decode mode to output the decoded raw data on the SDRAM.



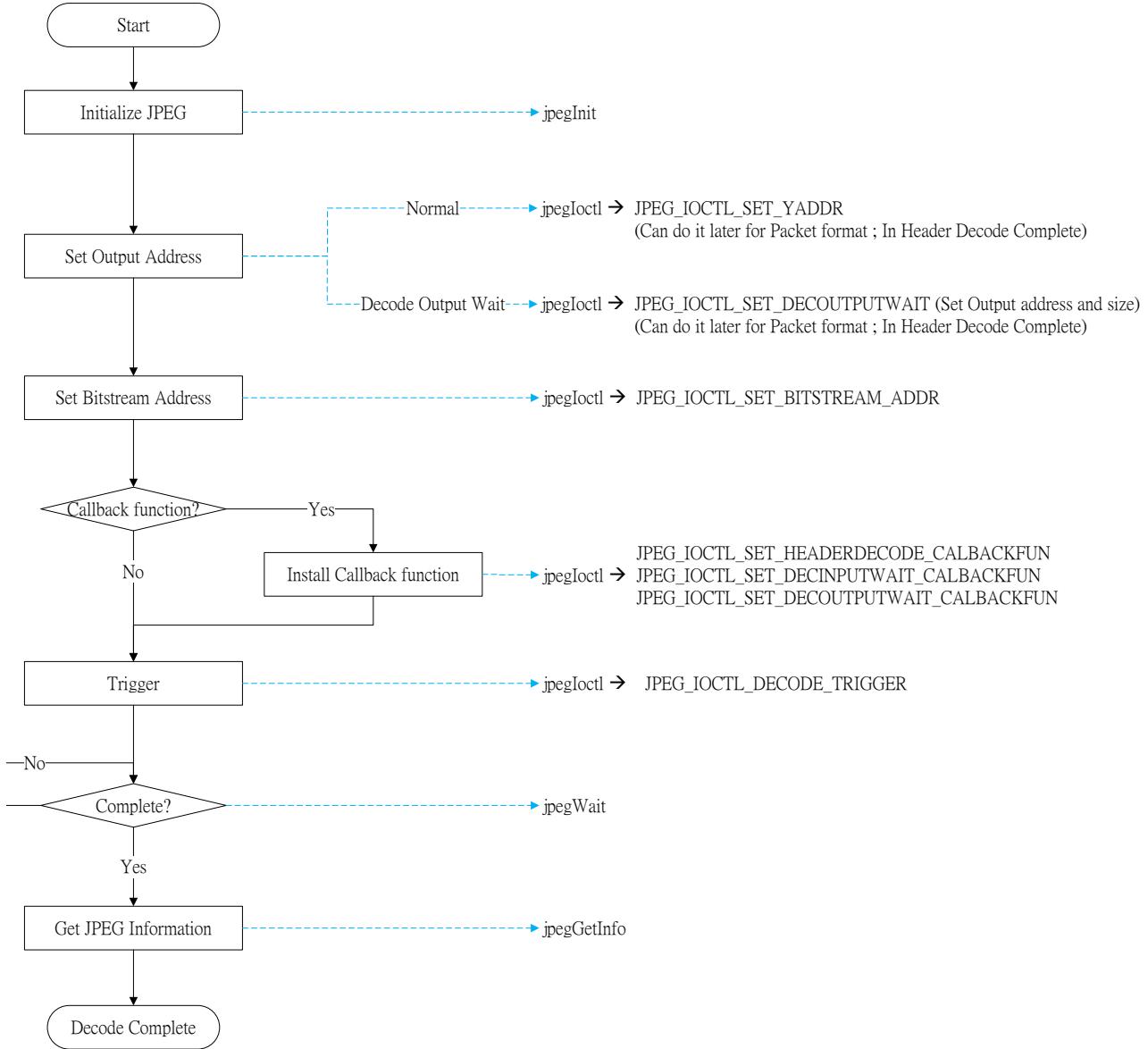
User can use stride function to output decoded image to any position on the Display Frame Buffer for Display. Following figure shows the decode mode with stride to output the decoded raw data on the Display Frame Buffer.



#### ■ Encode operation flow

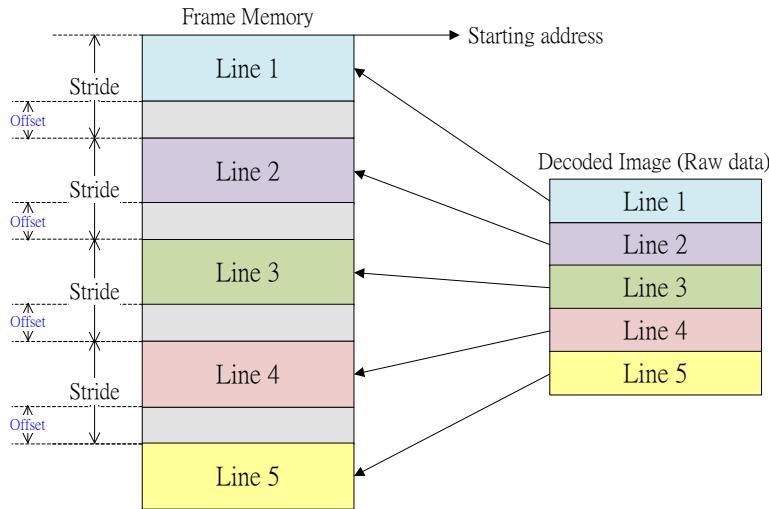


### ■ Decode operation flow



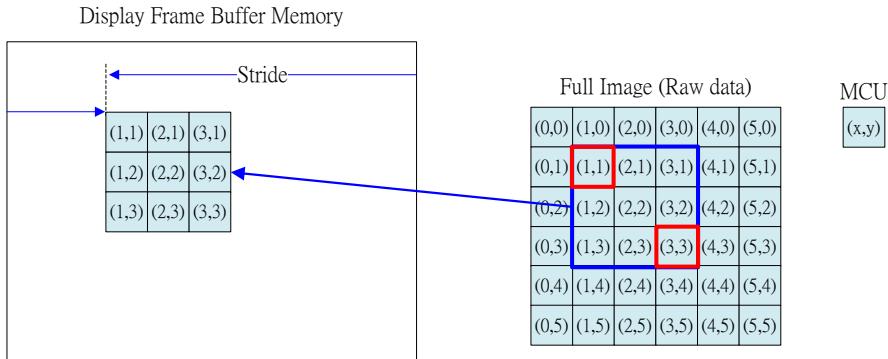
### ■ Decode stride

Before clearing Header Decode End interrupt, the value of stride must be set to stride value instead of original width. Offset is the difference between Stride and Image width. If Offset is 0, the decoded Raw data is continuous.



## ■ Window Decode

The JPEG decoder supports specified window decode mode. This function allows user to specify a sub-window region within the whole image to be decoded as shown in the following figure. Only the specified window region image will be decoded and stored to frame memory.

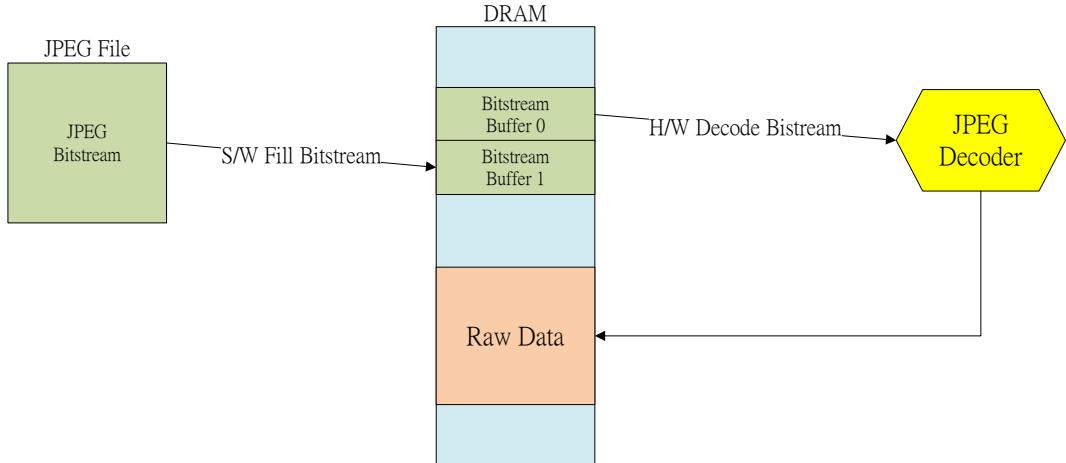


## ■ Decode Input Wait

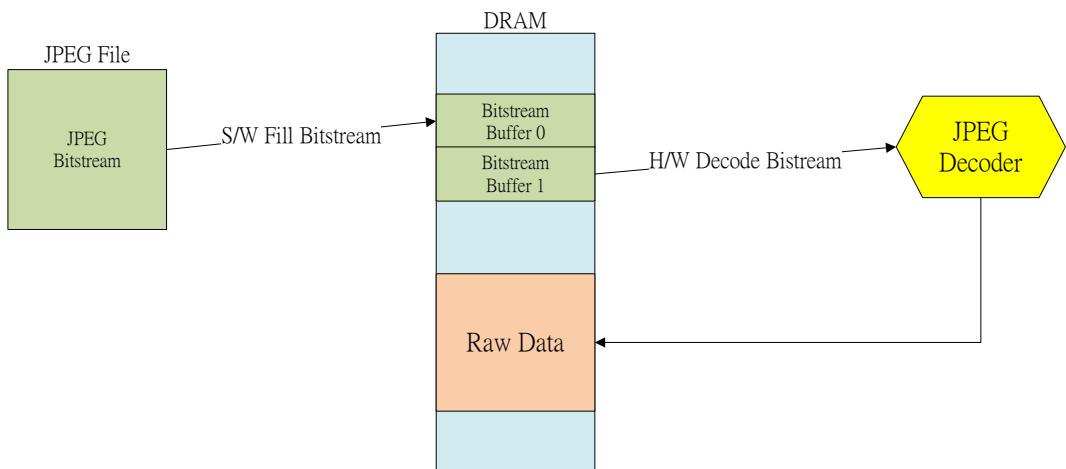
When the JPEG is in decoding mode, the input source is the JPEG bit-stream written by software. The bit-stream buffer size is in 2K unit dual-buffer manner. If the buffer-size is 2KB, user needs to fill 1KB bit-stream into one of the half buffer region before resuming JPEG operation when an input-wait interrupt is generated.

When JPEG engine decodes one of the half buffers, the Decode Input Wait call back function will be called. The Only thing user needs to do is to fill bit stream to the other buffer like the following Step 1 & Step 2 until entired bistream is filled into the buffer.

[Step 1] JPEG engine decodes the data in Buffer 0 and S/W fills the data into Buffer 1



[Step 2] JPEG engine decodes the data in Buffer1 and S/W fills the data into Buffer 0

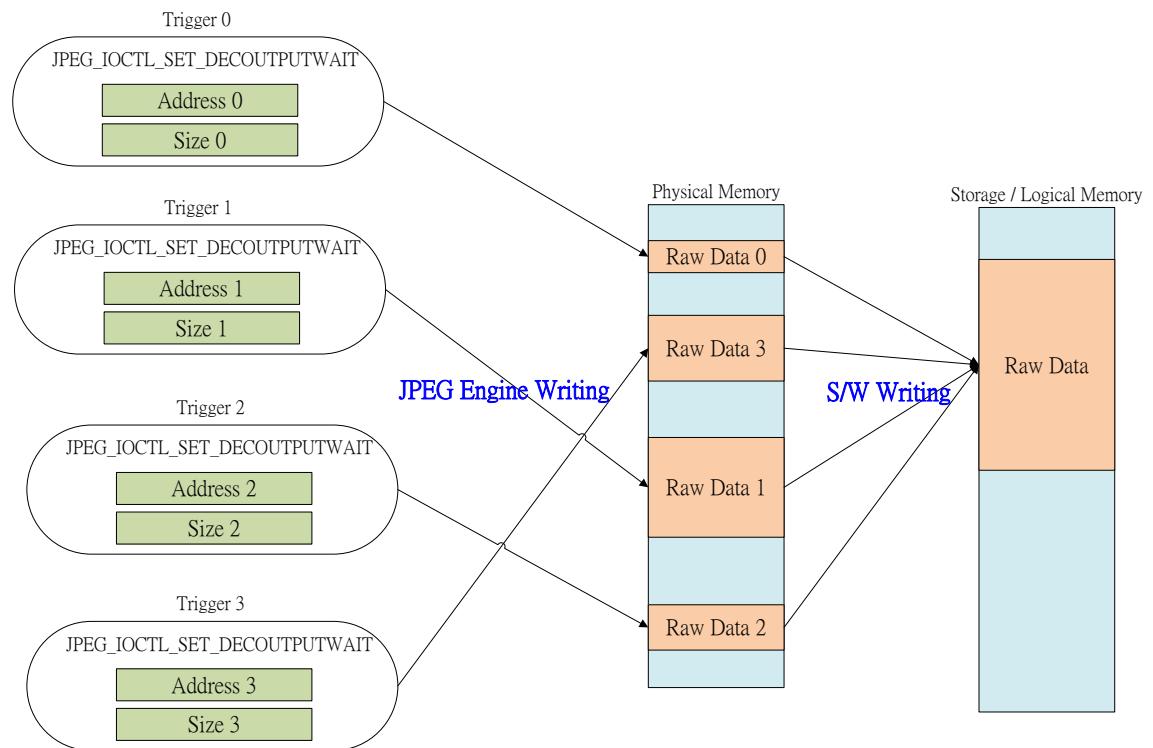


#### ■ Decode Output Wait

When there is not enough continuous space to store the decode output raw data, JPEG engine support a function to output data partially. User can get the whole data by assigning several output data address and size settings (JPEG\_IOCTL\_SET\_DECOUTPUTWAIT). Using this function, user can get the JPEG decoded image that larger than the available continuous memory space.

The decode output wait call back function will be called when the Output Data size is equal to the Size assigned by IOCTL - JPEG\_IOCTL\_SET\_DECOUTPUTWAIT. In the call back function, user should move the data to it's destination address and call the IOCTL again to set next address and data size.

[Note 1]. The data size of the final ICOTL must equal to the extract output size. Otherwise, user will get the Decode mplete interrupt instead of Decode output wait interrupt.



### ■ Header Decode Complete

In the callback function, user can get JPEG image width and height by calling `jpegGetInfo()`. After getting the information, user can use `jpegIoctl` to

- Allocate and set output buffer → `JPEG_IOCTL_SET_YADDR` for Packet format Only
- Change output buffer address → `JPEG_IOCTL_SET_YADDR` for Packet format Only
- Set Downscale → `JPEG_IOCTL_SET_DECODE_DOWNSCALE`
- Set Decode output Stride → `JPEG_IOCTL_SET_DECODE_STRIDE`
- Set windows decode → `JPEG_IOCTL_SET_WINDOW_DECODE`

## ***JPEG Library Constant Definition***

### ■ Encode operation

Name	Value	Description
Encode format		
<code>JPEG_ENC_PRIMARY</code>	0	Encode operation : Primary JPEG
<code>JPEG_ENC_THUMBNAIL</code>	1	Encode operation : Thumbnail JPEG
<code>JPEG_ENC_SOURCE_PLANAR</code>	0	Encode source : planar format
<code>JPEG_ENC_SOURCE_PACKET</code>	1	Primary Encode source : packet format

JPEG_ENC_PRIMARY_YUV420	0xA0	Primary Encode image format : YUV 4:2:0
JPEG_ENC_PRIMARY_YUV422	0xA8	Primary Encode image format : YUV 4:2:2
JPEG_ENC_PRIMARY_GRAY	0xA1	Primary Encode image format : GRAY
JPEG_ENC_THUMBNAIL_YUV420	0x90	Thumbnail Encode image format : YUV 4:2:0
JPEG_ENC_THUMBNAIL_YUV422	0x98	Thumbnail Encode image format : YUV 4:2:2
JPEG_ENC_THUMBNAIL_GRAY	0x91	Thumbnail Encode image format : GRAY
<b>Encode Header control</b>		
JPEG_ENC_PRIMARY_DRI	0x10	Restart Interval in Primary JPEG Header
JPEG_ENC_PRIMARY_QTAB	0x20	Quantization-Table in Primary JPEG Header
JPEG_ENC_PRIMARY_HTAB	0x40	Huffman-Table in Primary JPEG Header
JPEG_ENC_PRIMARY_JFIF	0x80	JFIF Header in Primary JPEG Header
JPEG_ENC_THUMBNAIL_DRI	0x1	Restart Interval in Thumbnail JPEG Header
JPEG_ENC_THUMBNAIL_QTAB	0x2	Quantization-Table in Thumbnail JPEG Header
JPEG_ENC_THUMBNAIL_HTAB	0x4	Huffman-Table in Thumbnail JPEG Header
JPEG_ENC_THUMBNAIL_JFIF	0x8	JFIF Header in Thumbnail JPEG Header

#### ■ Decode operation

Name	Value	Description
Decode output format		
JPEG_DEC_PRIMARY_PLANAR_YUV	0x8021	Primary Decode output format : planar format
JPEG_DEC_PRIMARY_PACKET_YUV422	0x0021	Primary Decode output format : planar YUV422
JPEG_DEC_PRIMARY_PACKET_RGB555	0x004021	Primary Decode output format : packet RGB555
JPEG_DEC_PRIMARY_PACKET_RGB555R1	0x404021	Primary Decode output format : packet RGB555R1
JPEG_DEC_PRIMARY_PACKET_RGB555R2	0x804021	Primary Decode output format : packet RGB555R2
JPEG_DEC_PRIMARY_PACKET_RGB565	0x006021	Primary Decode output format : packet RGB565
JPEG_DEC_PRIMARY_PACKET_RGB565R1	0x406021	Primary Decode output

		format : packet RGB565R1
JPEG_DEC_PRIMARY_PACKET_RGB565R2	0x806021	Primary Decode output format : packet RGB565R2
JPEG_DEC_PRIMARY_PACKET_RGB888	0x14021	Primary Decode output format : packet RGB888
JPEG_DEC_THUMBNAIL_PLANAR_YUV	0x8031	Thumbnail Decode output format : planar YUV
JPEG_DEC_THUMBNAIL_PACKET_YUV422	0x0031	Thumbnail Decode output format : packet RGB555
JPEG_DEC_THUMBNAIL_PACKET_RGB555	0x4031	Thumbnail Decode output format : packet RGB565
<b>JPEG format</b>		
JPEG_DEC_YUV420	0x000	JPEG format is YUV420
JPEG_DEC_YUV422	0x100	JPEG format is YUV422
JPEG_DEC_YUV444	0x200	JPEG format is YUV444
JPEG_DEC_YUV411	0x300	JPEG format is YUV411
JPEG_DEC_GRAY	0x400	JPEG format is Gray
JPEG_DEC_YUV422T	0x500	JPEG format is YUV422 Transport

### **JPEG Library Property Definition**

The JPEG library provide property structure to set JPEG property.

JPEG\_INFO\_T;

Name	Value	Description
<b>yuvformat</b>	JPEG_DEC_YUV420 JPEG_DEC_YUV422 JPEG_DEC_YUV444 JPEG_DEC_YUV411 JPEG_DEC_GRAY JPEG_DEC_YUV422T	JPEG format (Decode only)
<b>width</b>	< 8192	Decode Output width (Decode only)
<b>height</b>	< 8192	Decode Output height (Decode only)
<b>jpeg_width</b>	< 65535	JPEG width (Decode only)
<b>jpeg_height</b>	< 65535	JPEG height (Decode only)
<b>stride</b>	< 8192	Decode output Stride (Decode only)

<b>bufferend</b>	Reserved	Reserved
<b>image_size[2]</b>	< $2^{24}$ - 1	Encode Bitstream Size (Encode Only)

The JPEG library provide window decode function, user can partially decode the JPEG image by MCU unit (16 pixels \*16 pixels).

#### JPEG\_WINDOW\_DECODE\_T

Name	Value	Description
u16StartMCUX	0~511	Decode MCU Horizontal Start index
u16StartMCUY	0~511	Decode MCU Vertical Start index
u16EndMCUX	0~511	Decode MCU Horizontal End index
u16EndMCUY	0~511	Decode MCU Vertical End index
u32Stride	< 8192	Decode output Stride

## 15.3. JPEG API

### *jpegOpen*

#### Synopsis

```
INT jpegOpen(VOID)
```

#### Description

This function initializes the software resource, sets the engine clock and enables its interrupt

#### Parameter

None

#### Return Value

E_SUCCESS	- Always successes
-----------	--------------------

#### Example

```
jpegOpen();
```

### *jpegClose*

#### Synopsis

```
VOID jpegClose(VOID)
```

**Description**

Disable clock of JPEG engine and disable its interrupt

**Parameter**

None

**Return Value**

None

**Example**

```
jpegClose();
```

***jpegInit*****Synopsis**

```
VOID jpegInit(VOID)
```

**Description**

Reset JPEG engine and set default value to its registers

**Parameter**

None

**Return Value**

None

**Example**

```
jpegInit();
```

***jpegGetInfo*****Synopsis**

```
VOID jpegGetInfo(JPEG_INFO_T *info)
```

**Description**

This function can get JPEG width and height after header decode complete and get JPEG bit stream size after encode complete.

**Parameter**

info           JPEG Data type pointer stores the returned JPEG header information

**Return Value**

None

**Example**

```
JPEG_INFO_T jpegInfo;  
  
/* Get JPEG Header information */  
  
jpegGetInfo(&jpegInfo);
```

***jpegWait*****Synopsis**

```
INT jpegWait(VOID)
```

**Description**

After triggers JPEG engine, application need to wait the completion flag while JPEG engine completes its job.

**Parameter**

None

**Return Value**

E_FAIL	Error happened
E_SUCCESS	Action is done

**Example**

```
jpegWait();
```

***jpegIsReady*****Synopsis**

```
BOOL jpegIsReady(VOID)
```

**Description**

The function can get the JPEG engine status.

**Parameter**

None

**Return Value**

TRUE	Engine is ready
FALSE	Engine is busy

**Example**

```
jpegIsReady ();
```

## ***jpegSetQTAB***

### **Synopsis**

```
INT jpegSetQTAB(
    PUINT8      puQTable0,
    PUINT8      puQTable1,
    PUINT8      puQTable2,
    UINT8       u8num
);
```

### **Description**

The function can specify the Quantization table

### **Parameter**

puQTable0	Specify the address of Quantization table 0
puQTable1	Specify the address of Quantization table 1
puQTable2	Specify the address of Quantization table 2
u8num	Specify the number of Quantization table

### **Return Value**

E\_SUCCESS : Success

E\_JPEG\_TIMEOUT : Set Quantization table timeout

### **Example**

```
jpegSetQTAB(g_au8QTable0,g_au8QTable1, 0, 2);
```

## ***jpegIoctl***

### **Synopsis**

```
VOID jpegIoctl(UINT32 cmd, UINT32 arg0, UINT32 arg1)
```

### **Description**

This function allows programmers configure JPEG engine, the supported command and arguments listed in the table below.

Command	Argument 0	Argument 1	comment
JPEG_IOCTL_SET_YADDR	JPEG Y component frame buffer address		Specify the JPEG Y component

			frame buffer address.
JPEG_IOCTL_SET_YSTRIDE	JPEG Y component frame buffer stride		Specify the JPEG Y component frame buffer stride
JPEG_IOCTL_SET_USTRIDE	JPEG U component frame buffer stride		Specify the JPEG U component frame buffer stride
JPEG_IOCTL_SET_VSTRIDE	JPEG V component frame buffer stride		Specify the JPEG V component frame buffer stride
JPEG_IOCTL_SET_BITSTREAM_ADDR	JPEG bit stream buffer starting address		Specify the bit stream frame buffer starting address
JPEG_IOCTL_SET_SOURCE_IMAGE_HEIGHT	The encode source image height in pixel		Specify the encode source image height in pixel
JPEG_IOCTL_ENC_SET_HEADER_CONTROL	JPEG_ENC_PRIMARY_DRI JPEG_ENC_PRIMARY_QTAB JPEG_ENC_PRIMARY_HTAB JPEG_ENC_PRIMARY_JFIF		Specify the header information includes in the encoding bit stream
JPEG_IOCTL_SET_DEFAULT_QTAB			Specify the Quantization table
JPEG_IOCTL_SET_DECODE_MODE	JPEG_DEC_PRIMARY_PLANAR_YUV JPEG_DEC_PRIMARY_PACKET_YUV42 2 JPEG_DEC_PRIMARY_PACKET_RGB55 5 JPEG_DEC_PRIMARY_PACKET_RGB55 5R1		
	JPEG_DEC_PRIMARY_PACKET_RGB55 5R2 JPEG_DEC_PRIMARY_PACKET_RGB56 5 JPEG_DEC_PRIMARY_PACKET_RGB56 5R1 JPEG_DEC_PRIMARY_PACKET_RGB56 5R2 JPEG_DEC_PRIMARY_PACKET_RGB88 8		Specify the decoded image output format
JPEG_IOCTL_SET_ENCODE_MODE	JPEG_ENC_SOURCE_PLANAR JPEG_ENC_SOURCE_PACKET	JPEG_ENC_PRIMARY_YUV420 JPEG_ENC_PRIMARY_YUV422	Specify the encode source format and encoding image format
JPEG_IOCTL_SET_DIMENSION	Image height	Image width	Set the encode image dimension or decode output image dimension
JPEG_IOCTL_ENCODE_TRIGGER			Trigger the JPEG operation for encoding
JPEG_IOCTL_DECODE_TRIGGER			Trigger the JPEG operation for decoding
JPEG_IOCTL_WINDOW_DECODE	JPEG_WINDOW_DECODE_T		Enable window decode mode and set the decode window region
JPEG_IOCTL_SET_DECODE_STRIDE	Decode Output Stride (in pixel)		Specify the decode output stride
JPEG_IOCTL_SET_DECODE_DOWNSCALE	Scaled Height	Scaled Width	Set Decode downscale function
JPEG_IOCTL_SET_ENCODE_UPSCALE	Scaled Height	Scaled Width	Set Encode Upscale function
JPEG_IOCTL_SET_HEADER_DECODE_CALLBACKFUN	Header Decode Complete Call Back function pointer		Set Header Decode Complete Call Back function pointer
JPEG_IOCTL_SET_DECINPUT_TWAIT_CALLBACKFUN	Decode Input Wait Call Back function pointer		Set Decode Input Wait Call Back function pointer

JPEG_IOCTL_ADJUST_QTAB	JPEG_ENC_PRIMARY JPEG_ENC_THUMBNAIL	Quantization-Table Adjustment and control values[0]	Set Quantization-Table Adjustment and control
JPEG_IOCTL_ENC_RESERVED_FOR_SOFTWARE	Reserved size		Reserve memory space for user application
JPEG_IOCTL_SET_UADDR	Address for U Component		Set address for U Component
JPEG_IOCTL_SET_VADDR	Address for V Component		Set address for V Component
JPEG_IOCTL_SET_ENCODE_PRIMARY_RESTART_INTERVAL	Primary Restart interval		Set Primary Restart interval size
JPEG_IOCTL_SET_ENCODE_THUMBNAIL_RESTART_INTERVAL	Thumbnail Restart interval		Set Thumbnail Restart interval size
JPEG_IOCTL_GET_ENCODE_PRIMARY_RESTART_INTERVAL	The pointer to store Primary Restart interval size		Get Primary Restart interval size
JPEG_IOCTL_GET_ENCODE_THUMBNAIL_RESTART_INTERVAL	The pointer to store Thumbnail Restart interval size		Get Thumbnail Restart interval size
JPEG_IOCTL_SET_THUMBNAIL_DIMENSION	Thumbnail Heightt	Thumbnail Width	Set Thumbnail Dimension
JPEG_IOCTL_SET_ENCODE_SW_OFFSET	Offset		Set Software Encode Offset
JPEG_IOCTL_GET_THUMBNAIL_DIMENSION	The pointer to store Thumbnail Heightt	The pointer to store Thumbnail Width	Get Thumbnail Dimension
JPEG_IOCTL_GET_ENCODE_SW_OFFSET	The pointer to store Encode Offset		Get Software Encode Offset
JPEG_IOCTL_SET_ENCODE_PRIMARY_DOWNSCALE	Primary Downscaled Heightt	Primary Downscaled Width	Set Primary Encode downscale Size (Planar format only)
JPEG_IOCTL_SET_ENCODE_THUMBNAIL_DOWNSCALE	Thumbnail Downscaled Heightt	Thumbnail Downscaled Width	Set Thumbnail Encode downscale Size (Planar format only)
JPEG_IOCTL_SET_ENCODE_PRIMARY_ROTATE_RIGHT			Encode rotate right (Planar format only)
JPEG_IOCTL_SET_ENCODE_PRIMARY_ROTATE_LEFT			Encode rotate left (Planar format only)
JPEG_IOCTL_SET_ENCODE_PRIMARY_ROTATE_NORMAL			Encode no rotate (Planar format only)
JPEG_IOCTL_SET_DECOUT_PUTWAIT_CALLBACKFUN	Decode Output Wait call back function pointer		Set Decode Output Wait call back function (Packetformat Only)
JPEG_IOCTL_SET_DECOUT_PUTWAIT	Data Output Address	Data Output Size	Set Decode Output Wait address and size
JPEG_IOCTL_GET_DECOUT_PUTWAIT_ADDR	The pointer to store Decode Output Wait Address		Get Decode Output Wait Address
JPEG_IOCTL_GET_DECOUT_PUTWAIT_SIZE	The pointer to store Decode Output Wait Size		Get Decode Output Wait Size

**Parameter**

- cmd      Command  
 arg0     First argument of the command  
 arg1     Second argument of the command

**Return Value**

None

**Example**

```
/* Set Downscale to QVGA */

jpegIoctl(JPEG_IOCTL_SET_DECODE_DOWNSCALE, 240, 320);

/* Set Decode Stride to Panel width (480 pixel)*

jpegIoctl(JPEG_IOCTL_SET_DECODE_STRIDE, 480, 0);

/* Set Decoded Image Address */

jpegIoctl(JPEG_IOCTL_SET_YADDR, u32FrameBuffer, 0);

/* Set Bit stream Address */

jpegIoctl(JPEG_IOCTL_SET_BITSTREAM_ADDR, u32BitStream, 0);

/* Set Decode Input Wait mode (Input wait buffer is 8192) */

jpegIoctl(JPEG_IOCTL_SET_DECINPUTWAIT_CALLBACKFUN, (UINT32) JpegDecInputWait, 8192);

/* Decode mode */

jpegIoctl(JPEG_IOCTL_SET_DECODE_MODE, JPEG_DEC_PRIMARY_PACKET_YUV422, 0);

/* Set JPEG Header Decode End Call Back Function */

jpegIoctl(JPEG_IOCTL_SET_HEADERDECODE_CALLBACKFUN, (UINT32) JpegDecHeaderComplete,
0);

/* Trigger JPEG decoder */

jpegIoctl(JPEG_IOCTL_DECODE_TRIGGER, 0, 0);
```

```
/* Set Source Y/U/V Stride */

jpegIoctl(JPEG_IOCTL_SET_YSTRIDE, u16Width, 0);

jpegIoctl(JPEG_IOCTL_SET_USTRIDE, u16Width/2, 0);

jpegIoctl(JPEG_IOCTL_SET_VSTRIDE, u16Width/2, 0);

/* Primary Encode Image Width / Height */

jpegIoctl(JPEG_IOCTL_SET_DIMENSION, u16Height, u16Width);

/* Encode upscale 2x */

jpegIoctl(JPEG_IOCTL_SET_ENCODE_UPSCALE, u16Height * 2, u16Width * 2);

/* Set Encode Source Image Height */

jpegIoctl(JPEG_IOCTL_SET_SOURCE_IMAGE_HEIGHT, u16Height, 0);

/* Include Quantization-Table and Huffman-Table */

jpegIoctl(JPEG_IOCTL_ENC_SET_HEADER_CONTROL, JPEG_ENC_PRIMARY_QTAB | 
JPEG_ENC_PRIMARY_HTAB, 0);

/* Use the default Quantization-table 0, Quantization-table 1 */

jpegIoctl(JPEG_IOCTL_SET_DEFAULT_QTAB, 0, 0);
```

**Note [0]**

8 bits Quantization-Table Adjustment and control value.

7	6	5	4	3	2	1	0
<b>P_QADJUST</b>				<b>P_QVS</b>			

Bits	Descriptions
------	--------------

[7:4]	<b>P_QADJUST</b>	<p><b>Primary Quantization-Table Adjustment</b></p> <p>If the sum of the position (x, y) of quantization-table is greater than <b>P_QADJUST</b>, the quantization value will be set to 127. Otherwise the value will keep as the original.</p> <p>8x8 DCT block: x = 0~7, y = 0~7</p> <pre>if ((x+y) &gt; P_QADJUST) =&gt; Q' = 127 else           =&gt; Q' = Q</pre>
[3:0]	<b>P_QVS</b>	<p><b>Primary Quantization-Table Scaling Control</b></p> $Q' = (P_QVS[3]*2*Q)+(P_QVS[2]*Q)+(P_QVS[1]*Q/2)+(P_QVS[0]*Q/4)$

---

## 15.4. Example code

This demo code has sample code for “Normal Encode”, “Encode Upscale”, “Normal Decode”, “Decode Downscale & Stride”, “Decode Input Wait”, and “Decode Output Wait” (write/read from SD Card). Please refer to the JPEG sample code of SDK Non-OS.

# 16. PWM Library Overview

This library is designed to make user application to set N9H26 PWM more easily.  
The PWM library has the following features:

- PWM signal frequency and duty setting
- PWM Capture function

---

## 16.1. Programming Guide

### System Overview

The N9H26 have 4 channels pwm-timers. The 4 channels pwm-timers has 2 prescaler, 2 clock divider, 4 clock selectors, 4 16-bit counters, 4 16-bit comparators, 2 Dead-Zone generator. They are all driven by system clock. Each channel can be used as a timer and issue interrupt independently.

Each two channels pwm-timers share the same prescaler(channel0-1 share prescalar0 and channel2-3 share prescalar1). Clock divider provides each channel with 5 clock sources (1, 1/2, 1/4, 1/8, 1/16).

Each channel receives its own clock signal from clock divider which receives clock from 8-bit prescaler. The 16-bit counter in each channel receive clock signal from clock selector and can be used to handle one pwm period. The 16-bit comparator compares number in counter with threshold number in register loaded previously to generate pwm duty cycle.

The N9H26 have 4 channels pwm-timers and each pwm-timer includes a capture channel. The Capture 0 and PWM 0 share a timer that included in PWM 0; and the Capture 1 and PWM 1 share another timer, and etc. Therefore user must setup the PWM-timer before turn on Capture feature. After enabling capture feature, the capture always latched PWM-counter to CRLR when input channel has a rising transition and latched PWM-counter to CFLR when input channel has a falling transition.

Capture channel 0 interrupt is programmable by setting CCR0[1] (Rising latch Interrupt enable) and CCR0[2] (Falling latch Interrupt enable) to decide the condition of interrupt occur. Capture channel 1 has the same feature by setting CCR0[17] and CCR0[18]. And capture channel 2 & 3 has the same feature by setting CCR1[1],CCR1[2] and CCR1[17], CCR1[18] respectively. Whenever Capture issues Interrupt 0/1/2/3, the PWM counter 0/1/2/3 will be reload at this moment.

There are only four interrupts from PWM to advanced interrupt controller (AIC). PWM 0 and Capture 0 share the same interrupt channel, PWM1 and Capture 1 share the same interrupt and so on. Therefore, PWM function and Capture function in the same channel cannot be used at the same time.

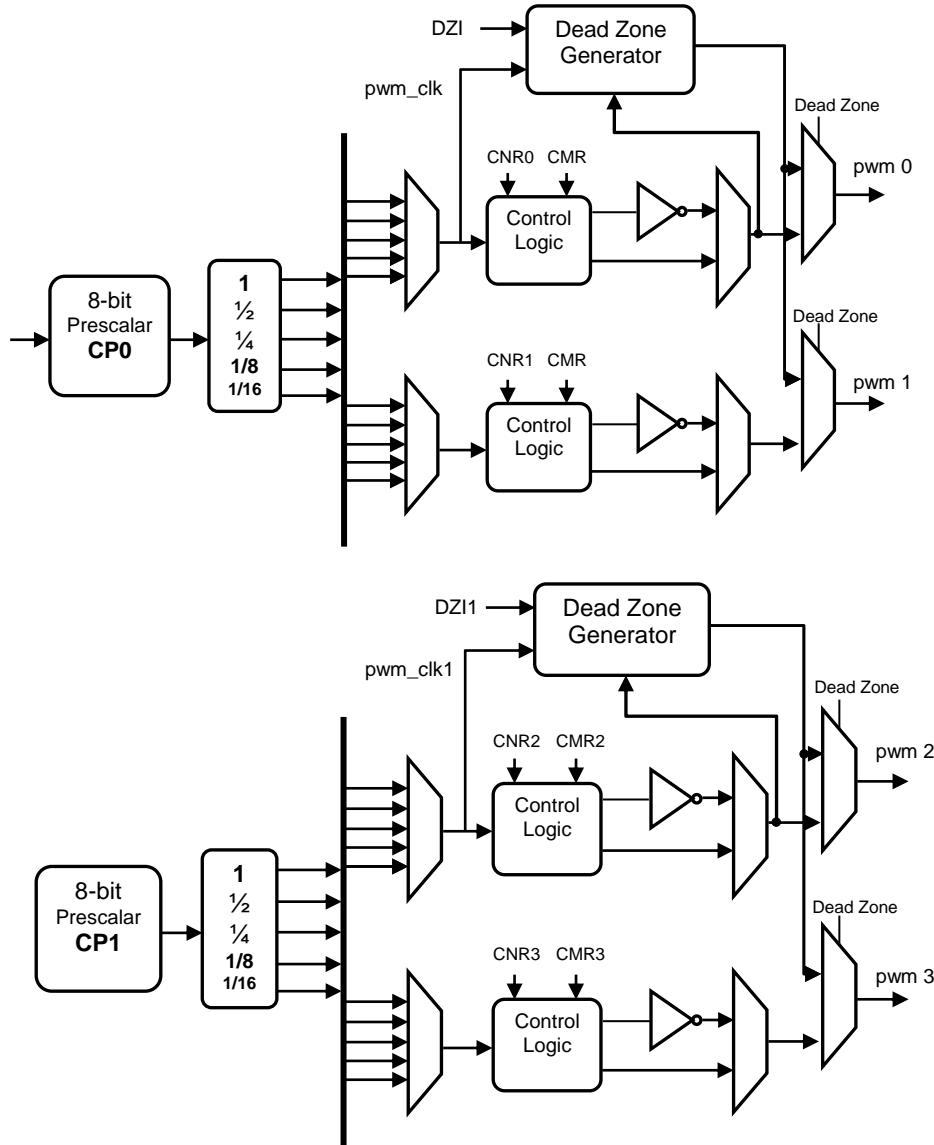
### PWM Features

- Two 8-bit prescalers and Two clock dividers
- Four clock selectors

- Four 16-bit counters and four 16-bit comparators
- Two Dead-Zone generator
- Capture function

### Block Diagram

The following figure describes the architecture of pwm in one group. (channel0&1 are in one group and channel2&3 are in another group)



### PWM Timer Control

## ■ Prescaler and clock selector

The PWM has two groups (two channels in each group) of timers. The clock input of the group is according to the PWM Prescaler Register (**PPR**) value. The PWM prescaler divided the clock input by  $PPR+1$  before it is fed to the counter. Please notice that when the PPR value equals zero, the prescaler output clock will stop. Furthermore, according to the PWM Clock Select Register (**CSR**) value, the clock input of PWM timer channel can be divided by 1,2,4,8 and 16.

Consider following examples, which explain the PWM timer period (Duty).

$$\text{period} = \frac{1}{(\text{SourceClock}) \div (PPR+1) \div CSR}$$

[Note 1]. PWM source clock can be APLL/UPLL/XIN.

When the PWM engine clock = 60 MHz, the maximum and minimum PWM timer counting period is described as follows.

Maximum period: PPR = 255 (since the length of PPR is 8bit) and CSR = 16

$$\text{period}_{\max} = \frac{1}{(60\text{MHz}) \div (255 + 1) \div 16} = 68.266\mu\text{s}$$

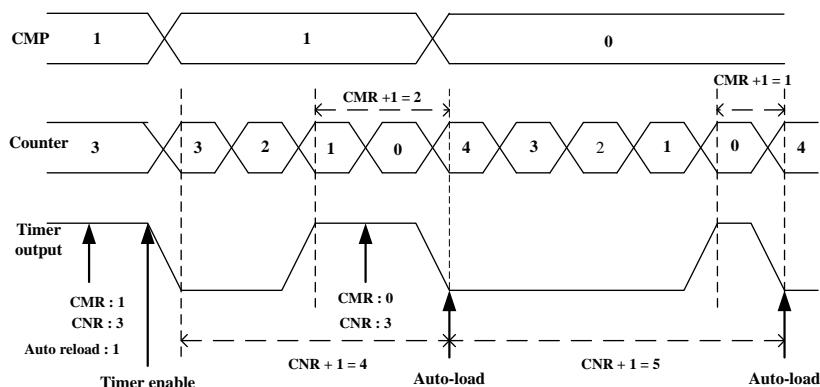
Minimum period: PCLK = 60 MHz, PPR=1 and CSR=1

$$\text{period}_{\min} = \frac{1}{(60\text{MHz}) \div (1 + 1) \div 1} = 0.0333\mu\text{s}$$

The maximum and minimum intervals between two interrupts depend on the  $\text{period}_{\max}$ ,  $\text{period}_{\min}$  and PWM Counter Register(**CNRx**) length. The maximum interval between two interrupts is  $(65535) \times (51.2\mu\text{s})$  since the length of CNR is 16bit. Please notice that the above calculation is based on the PCLK = 60MHz. Therefore, all of the values need to be recalculated when the PCLK is not equal to 60Mhz.

## ■ Basic Timer Operation

Basic Timer operation



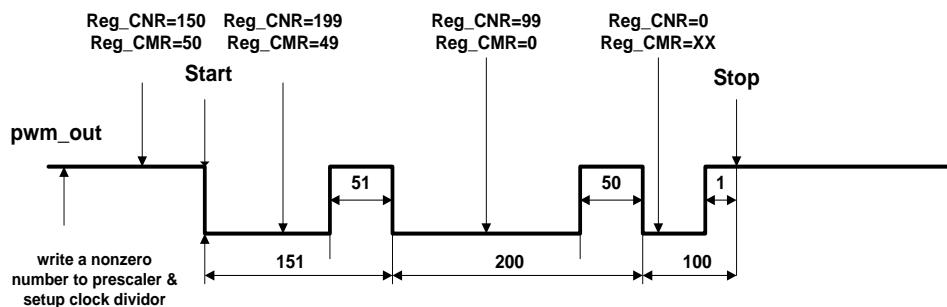
## ■ PWM Double Buffering and Automatic Reload

N9H26 PWM Timers have a double buffering function, enabling the reload value changed for next timer operation without stopping current timer operation. Although new timer value is set, current timer operation still operate successfully.

The counter value can be written into CNR0~3 and current counter value can be read from PDR0~3.

The auto-reload operation copies from CNR0~3 to down-counter when down-counter reaches zero. If CNR0~3 are set as zero, counter will be halt when counter count to zero. If auto-reload bit is set as zero, counter will be stopped immediately

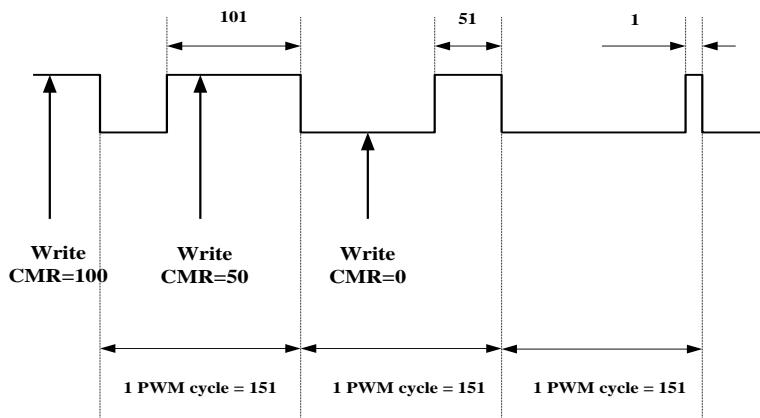
#### PWM double buffering



#### PWM Double Buffering and Automatic Reload

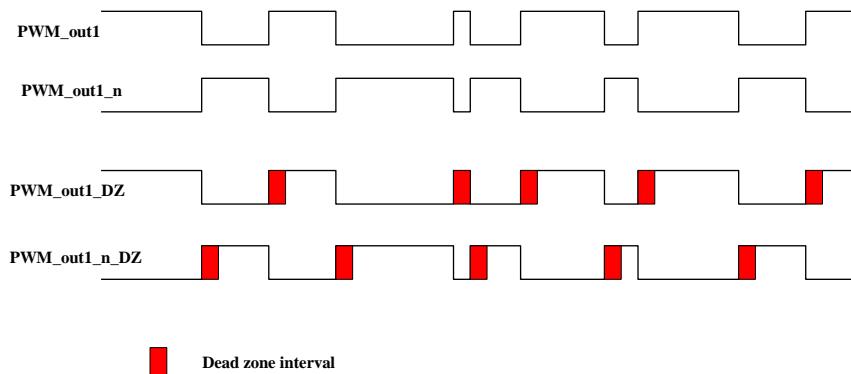
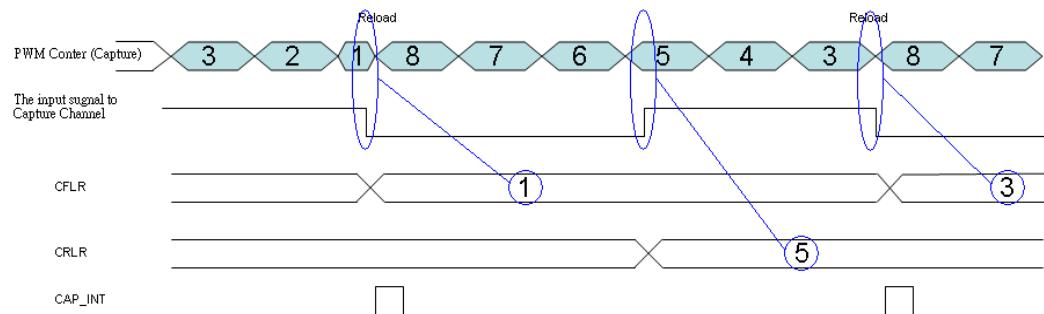
The double buffering function allows CMR written at any point in current cycle. The loaded value will take effect from next cycle.

##### Modulate PWM controller output duty ratio(CNR = 150)



#### PWM Double Buffering and Automatic Reload

N9H26 PWM is implemented with Dead Zone generator. They are built for power device protection. This function enables generation of a programmable time gap at the rising of PWM output waveform. User can program PPR [31:24] and PPR [23:16] to determine the two Dead Zone interval respectively.

**Dead zone generator operation****■ Capture Basic Timer Operation**

At this case, the CNR is 8:

1. When set falling interrupt enable, the pwm counter will be reload at time of interrupt occur.
2. The channel low pulse width is (CNR – CRLR).
3. The channel high pulse width is (CRLR - CFLR).
4. The channel cycle time is (CNR – CFLR).

**PWM Library Constant Definition**

Name	Value	Description
PWM_TIMER0	0x0	PWM Timer 0
PWM_TIMER1	0x1	PWM Timer 1
PWM_TIMER2	0x2	PWM Timer 2
PWM_TIMER3	0x3	PWM Timer 3
PWM_CAP0	0x0	PWM Capture 0
PWM_CAP1	0x1	PWM Capture 1
PWM_CAP2	0x2	PWM Capture 2
PWM_CAP3	0x3	PWM Capture 3

PWM_CAP_NO_INT	0	No PWM Capture Interrupt
PWM_CAP_RISING_INT	1	PWM Capture Rising Interrupt
PWM_CAP_FALLING_INT	2	PWM Capture Falling Interrupt
PWM_CAP_RISING_FLAG	6	Capture rising interrupt flag
PWM_CAP_FALLING_FLAG	7	Capture falling interrupt flag
PWM_CLOCK_DIV_1	4	Input clock divided by 1
PWM_CLOCK_DIV_2	0	Input clock divided by 2
PWM_CLOCK_DIV_4	1	Input clock divided by 4
PWM_CLOCK_DIV_8	2	Input clock divided by 8
PWM_CLOCK_DIV_16	3	Input clock divided by 16
PWM_TOGGLE_MODE	TRUE	PWM Timer Toggle mode
PWM_ONE_SHOT_MODE	FALSE	PWM Timer One-shot mode

### PWM Library Property Definition

The PWM library provides property structure to set PWM timer property.

Name	Value	Description
<i>u8Frequency</i>	$\geq 0$	The timer/capture frequency[0]
<i>u8HighPulseRatio</i>	1~100	High pulse ratio
<i>u8Mode</i>	PWM_ONE_SHOT_MODE / PWM_TOGGLE_MODE	PWM Timer Trigger mode
<i>bInverter</i>	TRUE / FALSE	Inverter Enable / Inverter Disable
<i>u8ClockSelector</i>	PWM_CLOCK_DIV_1/ PWM_CLOCK_DIV_2/ PWM_CLOCK_DIV_4/ PWM_CLOCK_DIV_8/ PWM_CLOCK_DIV_16	Clock Selector [1]
<i>u16PreScale</i>	2 ~ 256	Clock Prescale [1]
<i>u32Duty</i>	0~65535	Pulse duty [2]

[0] **PWM provides two timer setting mode: Frequency-setting and Property-setting modes.**

- Frequency-setting mode (*u8Frequency* > 0)

User doesn't need to set *u8ClockSelector* / *u16PreScale* / *u32Duty* fields. PWM library will set the proper values according to current APB clock automatically.

- Property-setting mode (*u8Frequency* = 0)
- User must set *u8ClockSelector* / *u16PreScale* / *u32Duty* fields by himself. Please refer to the previous section ‘Prescaler and clock selector.’
  - [1] The value take effect only when Property-setting mode.
  - [2] The value takes effect when Property-setting mode or the Capture functions. It is the capture monitor period.

---

## 16.2. PWM API

### **PWM\_Open**

#### **Synopsis**

```
VOID PWM_Open (VOID)
```

#### **Description**

Enable PWM engine clock and reset PWM

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
/* Enable PWM clock */  
PWM_Open();
```

### **PWM\_Close**

#### **Synopsis**

```
VOID PWM_Close (VOID)
```

#### **Description**

Disable PWM engine clock and the I/O enable

#### **Parameter**

None

#### **Return Value**

None

**Example**

```
/* Disable PWM clock */
PWM_Close();
```

**PWM\_SetClockSetting****Synopsis**

```
BOOL PWM_SetClockSetting(E_SYS_SRC_CLK eSrcClk, UINT32 u32PllDiver, UINT32
u32EngineDiver)
```

**Description**

This function is used to set PWM engine clock source and divie

**Parameter**

eSrcClk	PWM clock source. It could be eSYS_EXT=0, eSYS_APPLL= 2 and eSYS_UPPLL = 3.
u32PllDiver	PWM PLL Divider Selection (1~8) ()
u32EngineDiver	Engine Clock divider (1~256)

**Return Value**

TRUE	- Success.
FALSE	- Setting Fail..

**Note**

1. Parameter “u32PllDiver” is only be valid when eSrcClk is eSYS\_APPLL or eSYS\_UPPLL

**Example**

```
/* PWM Engine clock is UPPLL / 4, and Engine Clock divider is 2 */
PWM_SetClockSetting(eSYS_UPPLL, 4, 2);
```

**PWM\_GetEngineClock****Synopsis**

```
UINT32 PWM_GetEngineClock(E_SYS_SRC_CLK* peSrcClk)
```

**Description**

This function is used to get Current PWM engine clock

**Parameter**

peSrcClk	System clock source.
----------	----------------------

It could be eSYS\_EXT=0, eSYS\_APLL= 2 and eSYS\_UPPLL = 3.

#### Return Value

PWM Engine Clock (Hz)

#### Example

```
u32PWMClock = PWM_GetEngineClock(&eSrcClk) ;
sysprintf("PWM Clock Source is ");
switch(eSrcClk)
{
    case eSYS_EXT:
        sysprintf("External Crystal\n");
        break;
    case eSYS_APLL:
        sysprintf("APLL\n");
        break;
    case eSYS_UPPLL:
        sysprintf("UPPLL\n");
        break;
}
sysprintf("PWM Clock is %dHz\n",u32PWMClock);
```

### ***PWM\_SetTimerClk***

#### Synopsis

FLOAT PWM\_SetTimerClk (UINT8 u8Timer, PWM\_TIME\_DATA\_T \*sPt)

#### Description

This function is used to configure the frequency/pulse mode/inverter function

#### Parameter

u8Timer           The function to be set

PWM\_TIMER0 ~ PWM\_TIMER3: PWM timer 0 ~ 3

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

sPt                PWM property information

#### Return Value

= 0               - Setting Fail.

> 0               - Success. The actual frequency by PWM timer.

**Note**

2. The function will set the frequency property automatically (It will change the parameters to the values that it sets to hardware) when user set a nonzero frequency value
3. The function can set the proper frequency property (Clock selector/Prescale) for capture function and user needs to set the proper pulse duty by himself.

**Example**

```
/* Set PWM Timer 0 Configuration */  
PWM_SetTimerClk(PWM_TIMER0,&sPt);
```

**PWM\_SetTimerIO****Synopsis**

```
VOID PWM_SetTimerIO (UINT8 u8Timer, BOOL bEnable)
```

**Description**

This function is used to enable/disable PWM timer/capture I/O function

**Parameter**

u8Timer	The function to be set
	PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3
	PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

bEnable      Enable (TRUE) / Disable (FALSE)

**Return Value**

None

**Example**

```
/* Enable Output for PWM Timer 0 */  
PWM_SetTimerIO(PWM_TIMER0,TRUE);
```

**PWM\_Enable****Synopsis**

```
VOID PWM_Enable (UINT8 u8Timer, BOOL bEnable)
```

**Description**

This function is used to enable PWM timer / capture function

**Parameter**

u8Timer	The function to be set
	PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

bEnable      Enable (TRUE) / Disable (FALSE)

#### Return Value

None

#### Example

```
/* Enable the PWM Timer0 */
PWM_Enable(PWM_TIMER0,TRUE);
```

## **PWM\_IsTimerEnabled**

#### Synopsis

```
BOOL PWM_IsTimerEnabled (UINT8 u8Timer)
```

#### Description

This function is used to get PWM specified timer enable/disable state

#### Parameter

u8Timer      The function to be set

PWM\_TIMER0 ~ PWM\_TIMER3: PWM timer 0 ~ 3

#### Return Value

TRUE      - The specified timer is enabled.

FALSE      - The specified timer is disabled.

#### Example

```
/* Check PWM Timer0 is enabled or not */
If (PWM_IsTimerEnabled(PWM_TIMER0))
    sysprintf("PWM Timer 0 is enabled\n");
else
    sysprintf("PWM Timer 0 isn't enabled\n");
```

## **PWM\_SetTimerCounter**

#### Synopsis

```
VOID PWM_SetTimerCounter (UINT8 u8Timer, UINT16 u16Counter)
```

#### Description

This function is used to set the PWM specified timer counter

**Parameter**

u8Timer	The function to be set PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3
u16Counter	The timer value. (0~65535)

**Return Value**

None

**Note**

If the counter is set to 0, the timer will stop.

**Example**

```
/* Set PWM Timer 0 counter as 0 */
PWM_SetTimerCounter(PWM_TIMER0,0);
```

***PWM\_GetTimerCounter*****Synopsis**

```
UINT32 PWM_GetTimerCounter (UINT8 u8Timer)
```

**Description**

This function is used to get the PWM specified timer counter value

**Parameter**

u8Timer	The function to be set PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3
u16Counter	The timer value. (0~65535)

**Return Value**

The specified timer-counter value

**Example**

```
/* Loop when Counter of PWM Timer0 isn't 0 */

while(PWM_GetTimerCounter(PWM_TIMER0));
```

***PWM\_EnableDeadZone*****Synopsis**

```
VOID PWM_EnableDeadZone (UINT8 u8Timer, UINT8 u8Length, BOOL
bEnableDeadZone)
```

**Description**

This function is used to set the dead zone length and enable/disable Dead Zone function

**Parameter**

u8Timer	The function to be set PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3
u8Length	Dead Zone Length : 0~255
bEnableDeadZone	Enable DeadZone (TRUE) / Disable DeadZone (FALSE)

**Return Value**

None

**Note**

1. If Deadzone for PWM\_TIMER0 or PWM\_TIMER1 is enabled, the output of PWM\_TIMER1 is inverse waveform of PWM\_TIMER0.
2. If Deadzone for PWM\_TIMER2 or PWM\_TIMER3 is enabled, the output of PWM\_TIMER3 is inverse waveform of PWM\_TIMER2.

**Example**

```
/* Enable Deadzone of PWM Timer 0 and set it to 100 units*/
PWM_EnableDeadZone(PWM_TIMER0, 100, TRUE)
```

**PWM\_EnableInt****Synopsis**

VOID PWM\_EnableInt (UINT8 u8Timer, UINT8 u8Int)

**Description**

This function is used to enable the PWM timer/capture interrupt

**Parameter**

u8Timer	The function to be set PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3
u8Int	PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 Capture interrupt type (The parameter is valid only when capture function) PWM_CAP_RISING_INT: The capture rising interrupt. PWM_CAP_FALLING_INT: The capture falling interrupt. PWM_CAP_ALL_INT: All capture interrupt.

**Return Value**

None

**Example**

```
/* Enable Interrupt Sources of PWM Timer 0 */

PWM_EnableInt(PWM_TIMER0,0);

/* Enable Interrupt Sources of PWM Capture3 */

PWM_EnableInt(PWM_CAP3, PWM_CAP_FALLING_INT);
```

***PWM\_DisableInt*****Synopsis**

VOID PWM\_DisableInt (UINT8 u8Timer, UINT8 u8Int)

**Description**

This function is used to disable the PWM timer/capture interrupt

**Parameter**

u8Timer            The function to be set

PWM\_TIMER0 ~ PWM\_TIMER3: PWM timer 0 ~ 3

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

u8Int            Capture interrupt type (The parameter is valid only when capture function)

PWM\_CAP\_RISING\_INT: The capture rising interrupt.

PWM\_CAP\_FALLING\_INT: The capture falling interrupt.

PWM\_CAP\_ALL\_INT: All capture interrupt.

**Return Value**

None

**Example**

```
/* Disable Capture Interrupt */

PWM_DisableInt(PWM_CAP3,PWM_CAP_ALL_INT);
```

***PWM\_InstallCallBack*****Synopsis**

VOID PWM\_InstallCallBack (UINT8 u8Timer, PFN\_PWM\_CALLBACK pfncallback,  
PFN\_PWM\_CALLBACK \*pfnOldcallback)

**Description**

This function is used to install the specified PWM timer/capture interrupt call back function

**Parameter**

u8Timer	The function to be set PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3 PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3
pfncallback	The callback function pointer for specified timer / capture.
pfnOldcallback	The previous callback function pointer for specified timer / capture.

**Return Value**

None

**Example**

```
/* Install Callback function */  
  
PWM_InstallCallBack(PWM_TIMER0, PWM_PwmIRQHandler, &pfnOldcallback);
```

**PWM\_ClearInt****Synopsis**

VOID PWM\_ClearInt (UINT8 u8Timer)

**Description**

This function is used to clear the PWM timer/capture interrupt.

**Parameter**

u8Timer	The function to be set PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3 PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3
---------	--

**Return Value**

None

**Example**

```
/* Clear the PWM Capture 3 Interrupt */  
  
PWM_ClearInt(PWM_CAP3);
```

**PWM\_GetIntFlag****Synopsis**

**BOOL PWM\_GetIntFlag (UINT8 u8Timer)**

#### Description

This function is used to get the PWM timer/capture interrupt flag

#### Parameter

u8Timer      The function to be set

PWM\_TIMER0 ~ PWM\_TIMER3: PWM timer 0 ~ 3

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

#### Return Value

TRUE      - The specified interrupt occurs.

FLASE      - The specified interrupt doesn't occur.

#### Example

```
/* Get PWM Timer 0 Interrupt flag*/
PWM_GetIntFlag(PWM_TIMER0);
```

## **PWM\_GetCaptureIntStatus**

#### Synopsis

VOID PWM\_GetCaptureIntStatus (UINT8 u8Capture, UINT8 u8IntType)

#### Description

Check if there's a rising / falling transition

#### Parameter

u8Timer      The function to be set

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

u8Int      Capture interrupt type (The parameter is valid only when capture function)

PWM\_CAP\_RISING\_INT: The capture rising interrupt.

PWM\_CAP\_FALLING\_INT: The capture falling interrupt.

#### Return Value

TRUE      - The specified interrupt occurs.

FLASE      - The specified interrupt doesn't occur.

#### Example

```
/* Wait for Interrupt Flag (Falling) */
```

```
while(PWM_GetCaptureIntStatus(PWM_CAP0, PWM_CAP_FALLING_FLAG) != TRUE);
```

## PWM\_ClearCaptureIntStatus

### Synopsis

```
VOID PWM_ClearCaptureIntStatus (UINT8 u8Capture, UINT8 u8IntType)
```

### Description

Clear the rising / falling transition interrupt flag

### Parameter

u8Timer      The function to be set

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

u8Int      Capture interrupt type (The parameter is valid only when capture function)

PWM\_CAP\_RISING\_INT: The capture rising interrupt.

PWM\_CAP\_FALLING\_INT: The capture falling interrupt.

### Return Value

None

### Example

```
/* Clear the Capture Interrupt Flag */  
  
PWM_ClearCaptureIntStatus(PWM_CAP0, PWM_CAP_FALLING_FLAG);
```

## PWM\_GetRisingCounter

### Synopsis

```
UINT16 PWM_GetRisingCounter (UINT8 u8Capture)
```

### Description

The value which latches the counter when there's a rising transition

### Parameter

u8Timer      The function to be set

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

### Return Value

This function is used to get value which latches the counter when there's a rising transition

### Example

```
/* Get the Rising Counter Data */
```

```
u32Count[u32i++] = PWM_GetRisingCounter(PWM_CAP0);
```

## **PWM\_GetFallingCounter**

### **Synopsis**

```
UINT16 PWM_GetFallingCounter (UINT8 u8Capture)
```

### **Description**

The value which latches the counter when there's a falling transition

### **Parameter**

u8Timer            The function to be set

PWM\_CAP0 ~ PWM\_CAP3: PWM capture 0 ~ 3

### **Return Value**

This function is used to get value which latches the counter when there's a falling transition

### **Example**

```
/* Get the Falling Counter Data */  
  
u32Count[u32i++] = PWM_GetFallingCounter(PWM_CAP0);
```

# 17. RFC Library Introduction

The RF-CODEC includes the Convolution encode, Viterbi decode, Inner Interleave, and Inner De- Interleave. These are a forward error correction code (FEC) for wireless transceiver. The convolution encode includes a puncture function to change the coding rate from 1/2 to 2/3, 3/4, 5/6, or 7/8. If selecting 7/8 coding rate, the transfer data rate is maximum; otherwise, if selecting 1/2 coding rate, it gains the maximum BER performance. The Viterbi Decode is hard decision and the trace-back length is 32. The interleave function is used to disperse the transfer data. Because the performance of the Viterbi decode will be worst by burst error. The RF-CODEC block diagram is in Figure1. One thing is important that the RF-CODEC only supports PDMA function to handle the data from or to memory.

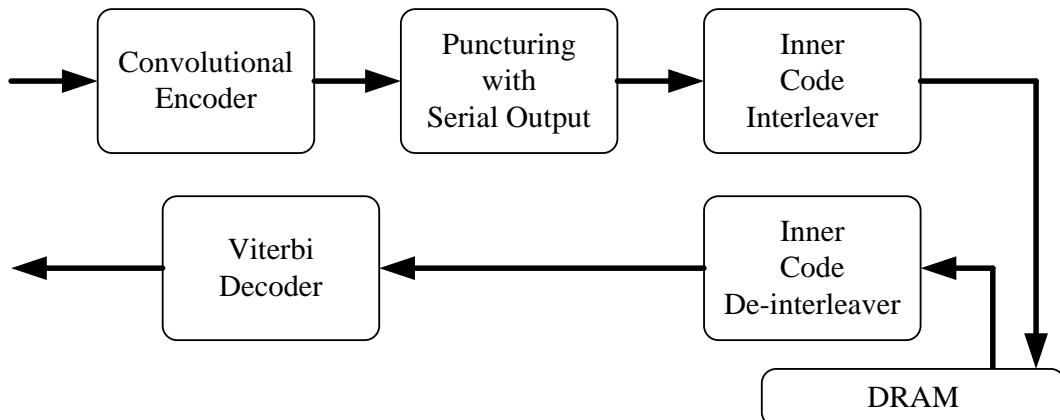


Figure1. RF-CODEC Block Diagram

## 17.1. Feature

- Supports Convolution encode and Viterbi decode
  - Coding rate supports 1/2, 2/3, 3/4, 5/6 and 7/8
- Supports Inner Interleave and Inner De-Interleave
- Supports PDMA function to handle the data from or to memory

## 17.2. API Data Structure

### ***E\_RF\_PNCTR\_MODE***

The coding rate setting in the puncture function.

Name	Value	Description
<i>E_PNCTR_1_2</i>	0	Coding rate is 1/2
<i>E_PNCTR_2_3</i>	1	Coding rate is 2/3
<i>E_PNCTR_3_4</i>	2	Coding rate is 3/4
<i>E_PNCTR_5_6</i>	3	Coding rate is 5/6
<i>E_PNCTR_7_8</i>	4	Coding rate is 7/8

---

## 17.3. API Function

### *RF\_Open*

#### Synopsis

```
INT32 RF_Open(void);
```

#### Description

Initialize RFC engine, install interrupt service routine, and call EDMA\_Init to initialize PDMA engine.

#### Parameter

None

#### Return Value

Successful	Always returns Successful
------------	---------------------------

### *RF\_Close*

#### Synopsis

```
void RF_Close(void);
```

#### Description

Tear down RFC engine.

#### Parameter

None

#### Return Value

None

## ***RF\_Enable\_Int***

### **Synopsis**

```
void RF_Enable_Int(void);
```

### **Description**

Enable RFC interrupt souce.

### **Parameter**

None

### **Return Value**

None

## ***RF\_Disable\_Int***

### **Synopsis**

```
void RF_Disable_Int(void);
```

### **Description**

Disable RFC interrupt souce.

### **Parameter**

None

### **Return Value**

None

## ***RF\_Set\_Puncture***

### **Synopsis**

```
INT32 RF_Set_Puncture(E_RF_PNCTR_MODE ePnctrMod);
```

### **Description**

Set the coding rate of the puncture function.

### **Parameter**

ePnctrMod	The coding rate of the puncture function
-----------	--

### **Return Value**

Successful	Set puncture is successful
RFC_ERR_PNCTR_MODE	Invalid puncture coding rate

## ***RF\_Get\_Puncture***

### **Synopsis**

```
E_RF_PNCTR_MODE RF_Get_Puncture(void);
```

### **Description**

Get the coding rate of the puncture function.

### **Parameter**

None

### **Return Value**

Successful	The puncture coding rate is returned
------------	--------------------------------------

## ***RF\_Encrypt***

### **Synopsis**

```
INT32 RF_Encrypt(UINT8* plainBuf, UINT8* cipherBuf, INT32 plainDataLen);
```

### **Description**

Start to run a RFC encryption calculation and wait for its finish.

### **Parameter**

plainBuf	Pointer to input plain text buffer
cipherBuf	Pointer to output cipher text buffer
plainDataLen	Length of plain buffer in bytes

### **Return Value**

(Value > 0)	Length of output buffer in bytes
RFC_ERR_DATA_BUF	RFC input buffer address is wrong

## ***RF\_Decrypt***

### **Synopsis**

```
INT32 RF_Decrypt(UINT8* cipherBuf, UINT8* plainBuf, INT32 plainDataLen);
```

### **Description**

Start to run a RFC decryption calculation and wait for its finish.

### **Parameter**

cipherBuf	Pointer to input cipher text buffer
plainBuf	Pointer to output plain text buffer

plainDataLen                    Length of plain buffer in bytes

**Return Value**

(Value > 0)                    Length of output buffer in bytes

RFC\_ERR\_DATA\_BUF            RFC input buffer address is wrong

---

## 17.4. Error Code Table

Code Name	Value	Description
Successful	0	Success
RFC_ERR_FAIL	RFC_ERR_ID   0x01	Internal error
RFC_ERR_PNCTR_MODE	RFC_ERR_ID   0x02	Invalid puncture coding rate
RFC_ERR_DATA_BUF	RFC_ERR_ID   0x03	NULL buffer address

# 18. Rotation Library Overview

The N9H26 Rotation library provides a set of APIs to rotate image in SDRAM. It use SRAM as temporary buffer. With these APIs, user can rotate image quickly.

The Rotation engine supports rotation left 90 degree and right 90 degree. It doesn't support downscale and format conversion. It only supports to rotate packet RGB565, packet XRGB888 and packet YUV422. It supports source line offset and destination line offset.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

---

## 18.1. Rotation Library APIs Specification

### ***rotOpen***

#### **Synopsis**

```
VOID rotOpen(VOID)
```

#### **Description**

This function is used to open the rotation library.

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
/* Open Rotation engine clock */  
rotOpen();
```

### ***rotClose***

#### **Synopsis**

VOID rotClose(VOID)

#### Description

Close the rot library.

#### Parameter

None

#### Return Value

Noe

#### Example

```
/* Close Rotation library*/
rotClose();
```

## ***rotInstallISR***

#### Synopsis

```
void rotInstallCallback (UINT32 u32IntNum,
                        PVOID pvIsr);
```

#### Description

This function is used to install callback function that is used to notice the upper layer for specified rotation image is done, buffer overrun or memory abort.

#### Parameter

u32IntNum	Rotation interrupt type. Please refer <i>Table 18-1:Interrupt Type</i>
pvIsr	Callback function

*Table 18-1:Interrupt Type*

Field name	Value	Description
E_ROT_COMP_INT	0	Rotation done interrupt
E_ROT_ABORT_INT	1	Memory abort interrupt
E_ROT_OVERFLOW_INT	2	Buffer overrun interrupt

#### Return Value

None.

#### Example

```
rotOpen();
```

```

rotInstallISR(E_ROT_COMP_INT, (PVOID)rotDoneHandler); /* Rotation done */
rotInstallISR(E_ROT_ABORT_INT, (PVOID)rotAbortHandler); /* Memory Abort */

```

## ***rotImageConfig***

### **Synopsis**

INT32 rotImageConfig(T\_ROT\_CONF\* ptRotConf)

### **Description**

This function is used to config Rotation engine.

### **Parameter**

ptRotConf      The structure to config Rotation engine.

Please refer Table 18-2: The definition of configuration structure

*Table 18-2: The definition of configuration structure*

Field name	Value	Description
eRotFormat	E_ROT_PACKET_RGB565 = 0 E_ROT_PACKET_RGB888 = 1 E_ROT_PACKET_YUV422 = 2	Pixel Format
eBufSize	E_LBUF_4 = 0 E_LBUF_8 = 2 E_LBUF_16 = 16	Use SRAM line.
eRotDir	E_ROT_ROT_R90 = 0 E_ROT_ROT_L90 = 1	Right or left rotation
u32RotDimHW	[31:16] : Rotate image height [15:0] : Rotate image width	Rotate image dimension
u32SrcLineOffset		Source line offset
u32DstLineOffset		Source line offset
u32SrcAddr		Source Buffer Address
u32DstAddr		Destination Buffer Address

### **Return Value**

Successful

### **Example**

```

T_ROT_CONF tRotConf;

rotOpen();

rotInstallISR(E_ROT_COMP_INT, (PVOID)rotDoneHandler); /* Rotation done */

rotInstallISR(E_ROT_ABORT_INT, (PVOID)rotAbortHandler); /* Memory Abort */

tRotConf.eBufSize = E_LBUF_4;

tRotConf.eRotDir = E_ROT_ROT_L90;

tRotConf.eRotFormat = E_ROT_PACKET_RGB565;

tRotConf.u32RotDimHW = 0x01E00280;

tRotConf.u32SrcLineOffset = 0;

tRotConf.u32DstLineOffset= 0;

tRotConf.u32SrcAddr = ADDR_ROT_SRC_ADDR;

tRotConf.u32DstAddr = ADDR_ROT_DST_ADDR

```

## ***rotGetPacketPixelWidth***

### **Synopsis**

INT32 rotGetPacketPixelWidth(E\_ROTENG\_FMT ePacFormat)

### **Description**

The function is used to get data width for the specified format.

### **Parameter**

ePacFormat      Rotation format. Please refer Table 18-3:Rotation Format

*Table 18-3:Rotation Format*

Field name	Value	Description
E_ROT_PACKET_RGB565	0	Packet RGB565. The data width is 2
E_ROT_PACKET_RGB888	1	Packet RGB888. The data width is 4
E_ROT_PACKET_RGB888	2	Packet YUV422. The data width is 2

### **Return Value**

The data width of rotation image. Byte unit.

### **Example**

```
UINT8 u8PixelWidth;
```

```
u8PixelWidth=rotGetPacketPixelWidth(ptRotConf->eRotFormat);
```

### ***rotTrigger***

#### **Synopsis**

```
INT32 rotTrigger (void)
```

#### **Description**

The function is used to get data width for the specified format

#### **Parameter**

None

#### **Return Value**

1: Meaning Rotation engine busy

0: Successful.

#### **Example**

```
UINT8 u8PixelWidth;  
  
u8PixelWidth=rotGetPacketPixelWidth(ptRotConf->eRotFormat);
```

---

## **18.2. Error Code Table**

<b>Code Name</b>	<b>Value</b>	<b>Description</b>
Successful	0	Successful
ERR_ROT_BUSY	0xFFFF2001	Rotation engine is busy

# 19. RSC Library Introduction

The RS\_CODEC controller performs two main functions - Reed-Solomon Encoder / Decoder and Convolutional Interleaver / Deinterleaver. When in encode mode, data from system bus can be encoded by Reed-Solomon Encoder and interleaved by convolutional interleaver. When in decode mode, data from system bus can be de-interleaved and decoded by Reed-Solomon Decoder.

---

## 19.1. Feature

- Supports Reed-Solomon Encoder / Decoder
  - (N=204, K=188, t=8) with the Field Generator Polynomial:  $p(x)=x^8+x^4+x^3+x^2+1$
  - Can correct 8 bytes error in 188 bytes block transmission
- Supports Convolutional Interleaver / Deinterleaver
  - Convolutional byte-wise interleaving with depth I=12 and 17 bytes FIFO
- Support PDMA to access RSC read / write buffers

---

## 19.2. Function

### ***RS\_Open***

#### **Synopsis**

```
INT32 RS_Open(void);
```

#### **Description**

Initialize RSC engine, install interrupt service routine, and call EDMA\_Init to initialize PDMA engine.

#### **Parameter**

None

#### **Return Value**

Successful	Always returns Successful
------------	---------------------------

***RS\_Close*****Synopsis**

```
void RS_Close(void);
```

**Description**

Tear down RSC engine.

**Parameter**

None

**Return Value**

None

***RS\_Enable\_Int*****Synopsis**

```
void RS_Enable_Int(void);
```

**Description**

Enable RSC interrupt souce.

**Parameter**

None

**Return Value**

None

***RS\_Disable\_Int*****Synopsis**

```
void RS_Disable_Int(void);
```

**Description**

Disable RSC interrupt souce.

**Parameter**

None

**Return Value**

None

## ***RS\_Encrypt***

### **Synopsis**

```
INT32 RS_Encrypt(UINT8* plainBuf, UINT8* cipherBuf, INT32 dataLen, UINT8  
isInterleave);
```

### **Description**

Start to run a RSC encryption calculation and wait for its finish.

### **Parameter**

plainBuf	Pointer to input plain text buffer
cipherBuf	Pointer to output cipher text buffer
dataLen	Length of input buffer in bytes
isInterleave	RSC runs in interleave or deinterleave mode

### **Return Value**

(Value > 0)	Length of output buffer in bytes
RSC_ERR_DATA_LEN	RSC input data length is wrong
RSC_ERR_DATA_BUF	RSC input buffer address is wrong

## ***RS\_Decrypt***

### **Synopsis**

```
INT32 RS_Decrypt(UINT8* cipherBuf, UINT8* plainBuf, UINT32 dataLen, UINT8  
isInterleave);
```

### **Description**

Start to run a RSC decryption calculation and wait for its finish.

### **Parameter**

cipherBuf	Pointer to input cipher text buffer
plainBuf	Pointer to output plain text buffer
dataLen	Length of input buffer in bytes
isInterleave	RSC runs in interleave or deinterleave mode

### **Return Value**

(Value > 0)	Length of output buffer in bytes
RSC_ERR_DATA_LEN	RSC input data length is wrong
RSC_ERR_DATA_BUF	RSC input buffer address is wrong
RSC_ERR_DEC_ERROR	RSC decode error and cannot fix

---

### 19.3. Error Code Table

Code Name	Value	Description
Successful	0	Success
RSC_ERR_FAIL	RSC_ERR_ID   0x01	Internal error
RSC_ERR_DATA_LEN	RSC_ERR_ID   0x02	Data length is not block alignment
RSC_ERR_DATA_BUF	RSC_ERR_ID   0x03	NULL buffer address
RSC_ERR_DEC_ERROR	RSC_ERR_ID   0x04	RSC decode error

# 20. RTC Library Overview

This library is designed to make user application access N9H26 RTC more easily.  
The RTC library has the following features:

- Support RTC Current/Alarm time access.
- Support System Power Off Control

---

## 20.1. Programming Guide

### System Overview

Real Time Clock (RTC) block can be operated by independent power supply while the system power is off. The RTC uses a 32.768 KHz external crystal. It can transmit data to CPU with BCD values. The data includes the time by (second, minute and hour), the day by (day, month and year). In addition, to achieve better frequency accuracy, the RTC counter can be adjusted by software.

The built in RTC is designed to generate the alarm interrupt and periodic interrupt signals. The period interrupt can be 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second. The alarm interrupt indicates that time counter and calendar counter have counted to a specified time recorded in TAR and CAR.

The wakeup signal is used to wake the system up from sleep mode.

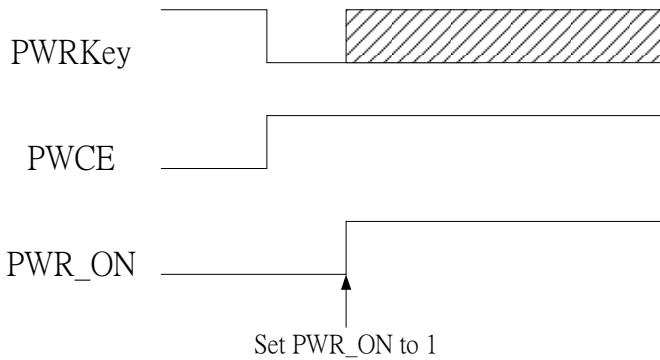
### RTC Features

- There is a time counter (second, minute, hour) and calendar counter (day, month, year) for user to check the time.
- Absolute Alarm register (second, minute, hour, day, month, year).
- Relative Alarm
- Alarm Mask for Minutely / Hourly / Daily / Weekly / Monthly/Yearly Alarm
- 12-hour or 24-hour mode is selectable.
- Recognize leap year automatically.
- The day of week counter.
- Frequency compensate register (FCR).
- Beside FCR, all clock and alarm data expressed in BCD code.
- Support time tick interrupt.
- Support wake up function.
- System Power off Control function

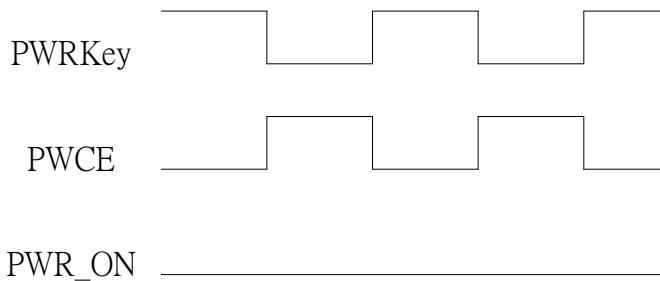
### System Power Control Flow

- Power On from Key

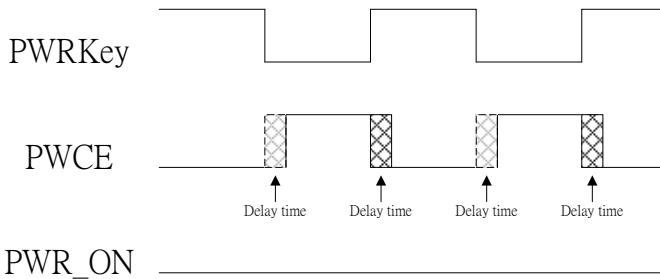
User presses the Power Key to make the Power Control Signal, PWCE to high. If PWR\_ON bit is set, the Power Key can be released and the PWCE will be keep on. When system is power on, IBR will set PWR\_ON first.



If PWR\_ON doesn't be set to 1, the PWCE will back to low when the Power Key is released.



And RTC supports a function (POWER\_KEY\_DURATION\_LENGTH) to postpone the time to set PWCE to high when pressing Power Key or to postpone the time to set PWCE to low when releasing Power Key. The delay time is from 62ms to 868ms. The function is default disabled. User can enable it after first power on and the setting can be kept when RTC is powered even whole system is power-off.



[Note] The function (POWER\_KEY\_DURATION\_LENGTH) only works only when Power Key is pressed or released.

#### ■ Power Key Duration (POWER\_KEY\_DURATION)

The delay time between Power Key pressed and PWCE high (or Power Key released and PWCE low). Minimum duration that power key must be pressed to turn on core power.

- Minimum power key duration =  
 $0.25 * (\text{POWER\_KEY\_DURATION} + 1)$  sec.

#### ■ Normal system Power Control Flow

The control steps are as follows

1. User press the power key, RPWR, to makes the power control signal, PWCE pin, to high. If the PWR\_ON bit, PWRON[0], be set, the power key can be released and the PWCE will keep on. If the PWR\_ON bit, PWRON [0], doesn't be set as 1, the PWCE will back to low when the power key is released.
2. If there is another pulse on power key when the PWR\_ON bit is set, the system will get an interrupt signal (PSWI). User can decide to clear the PWR\_ON or not. If this bit is clear, the PWCE will go to low to turn off the core power. If the PWRON bit is also kept high, the PWCE pin will keep in high level. If there is not any pulse on the power key and the PWR\_ON bit is clear by user, the PWCE pin is also set to low at this time.

The follow table is the system power control flow true table.

Input		Output PWCE	Note
PWRKey	PWR_ON		
X1	X2	Y	
1	0	0	RTC powered only (Default state)
0	0	1	Press key, Power On
0	1	1	keep key & S/W Set X2, Power On
1	1	1	Left key, Power keep On
0	1	1	Press key, get INT, intend to power Off
1	0	0	Left key & S/W clean X2, power Off Or S/W clean X2 , don't need press key, power off
X	1	1	RST_ active, still keep power whenX2=1

**PWCE is open drain output**  
**X1, internal pull-up**  
**X2, it is R/W able**  
**There is Interrupt from key be pressed**

#### ■ Force system Power Off Control Flow

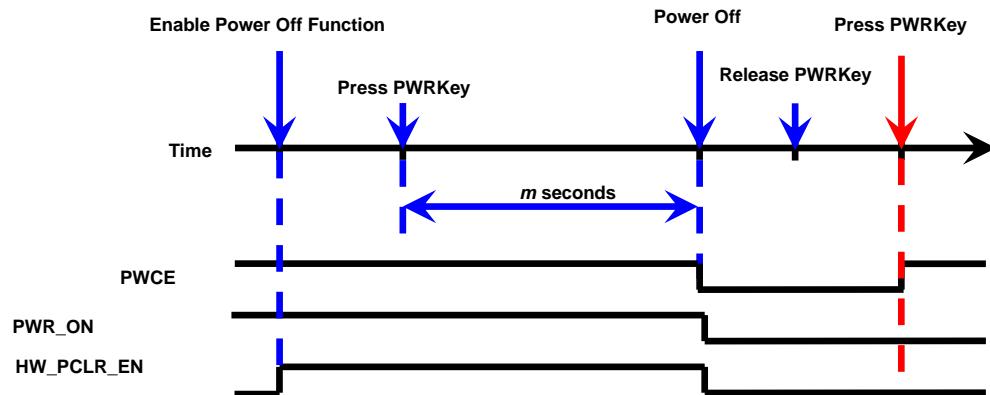
The RTC supports a hardware automatic power off function and a software power off function like Notebook. For hardware power off function, it can be enable and disable in HW\_PCLR\_EN

bit and the user presses the power button for a few seconds to power off system. The time to press power the button to power off is configured in PCLR\_TIME.

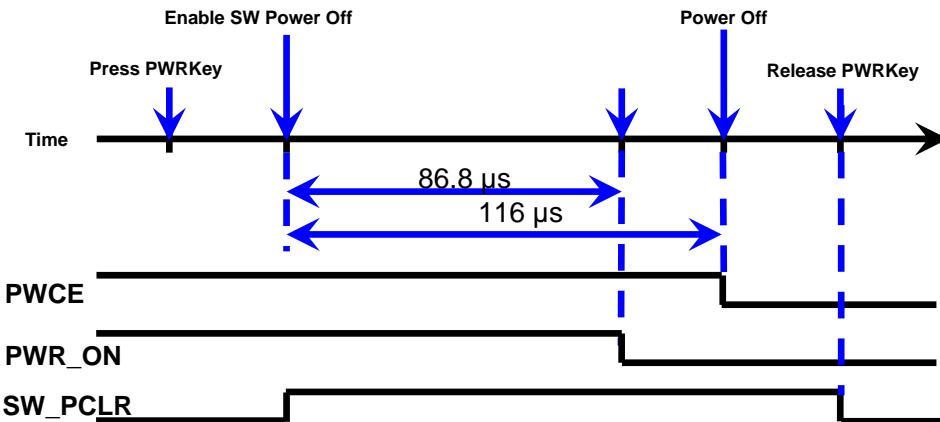
PCLR_TIME Setting	Pressed time to power off	PCLR_TIME Setting	Pressed time to power off
0	2~3 second	8	10~11 seconds
1	3~4 second	9	11~12 seconds
2	4~5 seconds	10	12~13 seconds
3	5~6 seconds	11	13~14 seconds
4	6~7 seconds	12	14~15 seconds
5	7~8 seconds	13	15~16 seconds
6	8~9 seconds	14	16~17 seconds
7	9~10 seconds	15	17~18 seconds

The RTC supports a hardware power off function to provide the power off flow like Notebook. The user presses the power button for a few seconds to power off the system. The time to press power key to power off is counted by hardware. After the time, hardware will set the PWCE to low and clear the PWR\_ON and HW\_PCLR\_EN. After power off, user can decide to set the PWR\_ON bit to power on system or not when the PWRKey is pressed.

The timing of the hardware power off function is following



The RTC also supports a software power off function. The user presses the power button for a few seconds to power off the system. The time to press power key to power off is counted by user. When the PWR\_ON bit is cleared by user, the PWCE outputs low after 116us and the SW\_PCLR bit is cleared when the power key is released. See the timing Figure as following.



### RTC Library Constant Definition

Name	Value	Description
RTC_CLOCK_12	0	12-Hour mode
RTC_CLOCK_24	1	24-Hour mode
RTC_AM	1	a.m.
RTC_PM	2	p.m.
RTC_LEAP_YEAR	1	Leap year
RTC_TICK_1_SEC	0	1 tick per second
RTC_TICK_1_2_SEC	1	2 tick per second
RTC_TICK_1_4_SEC	2	4 tick per second
RTC_TICK_1_8_SEC	3	8 tick per second
RTC_TICK_1_16_SEC	4	16 tick per second
RTC_TICK_1_32_SEC	5	32 tick per second
RTC_TICK_1_64_SEC	6	64 tick per second
RTC_TICK_1_128_SEC	7	128 tick per second
RTC_SUNDAY	0	Day of Week: Sunday
RTC_MONDAY	1	Day of Week: Monday
RTC_TUESDAY	2	Day of Week: Tuesday
RTC_WEDNESDAY	3	Day of Week: Wednesday
RTC_THURSDAY	4	Day of Week: Thursday
RTC_FRIDAY	5	Day of Week: Friday
RTC_SATURDAY	6	Day of Week: Saturday
RTC_ALARM_INT	0x01	Absolute Alarm Interrupt
RTC_TICK_INT	0x02	Tick Interrupt
RTC_PSWI_INT	0x04	Power Switch Interrupt

RTC_RELATIVE_ALARM_INT	0x08	Relative Alarm Interrupt
RTC_ALL_INT	0x0F	All Interrupt
RTC_IOC_IDENTIFY_LEAP_YEAR	0	Identify the leap year command
RTC_IOC_SET_TICK_MODE	1	Set tick mode command
RTC_IOC_GET_TICK	2	Get tick command
RTC_IOC_RESTORE_TICK	3	Restore tick command
RTC_IOC_ENABLE_INT	4	Enable interrupt command
RTC_IOC_DISABLE_INT	5	Disable interrupt command
RTC_IOC_SET_CURRENT_TIME	6	Set Current time command
RTC_IOC_SET_ALAMRM_TIME	7	Set Alarm time command
RTC_IOC_SET_FREQUENCY	8	Set Frequency command
RTC_IOC_SET_POWER_ON	9	Set Power On (Set PWR_ON to 1)
RTC_IOC_SET_POWER_OFF	10	Set Power Off (Set PWR_ON to 0)
RTC_IOC_SET_POWER_OFF_PERIOD	11	Set Power Off Period (PCLR_TIME)
RTC_IOC_ENABLE_HW_POWEROFF	12	Enable H/W Power Off
RTC_IOC_DISABLE_HW_POWEROFF	13	Disable H/W Power Off
RTC_IOC_GET_POWERKEY_STATUS	14	Get Power Key Status
RTC_IOC_SET_PSWI_CALLBACK	15	Set Power Switch Interrupt Callback function
RTC_IOC_GET_SW_STATUS	16	Get SW Status data (8 bits)
RTC_IOC_SET_SW_STATUS	17	Set SW Status data (8 bits)
RTC_IOC_SET_RELATIVE_ALARM	18	Set relative alarm and install call backfunvtion
RTC_IOC_SET_POWER_KEY_DURATION	19	Set Power Control Signal Delay
RTC_IOC_SET_POWER_KEY_LEVEL_TRIG	20	Set Power Key Trigger Mode – Level Trigger
RTC_IOC_SET_POWER_KEY_EDGE_TRIG	21	Set Power Key Trigger Mode – Edge Trigger
RTC_IOC_GET_SW_REGISTER0	22	Get SW Status Register 0 data (32 bits)
RTC_IOC_SET_SW_REGISTER0	23	Set SW Status Register 0 data (32 bits)
RTC_IOC_GET_SW_REGISTER1	24	Get SW Status Register 1 data (32 bits)
RTC_IOC_SET_SW_REGISTER1	25	Set SW Status Register 1 data (32 bits)
RTC_CURRENT_TIME	0	Current time
RTC_ALARM_TIME	1	Alarm time
RTC_WAIT_COUNT	10000	RTC Initial Time out Value
RTC_YEAR2000	2000	RTC Year Reference Value

## RTC Library Time and Date Definition

The RTC library provides time structure to access RTC time property.

Name	Value	Description
<i>u8cClockDisplay</i>	RTC_CLOCK_12 / RTC_CLOCK_24	12 Hour Clock / 24 Hour Clock
<i>u8cAmPm</i>	RTC_AM / RTC_PM	the AM hours / the PM hours
<i>u32cSecond</i>	0~59	Second value
<i>u32cMinute</i>	0~59	Minute value
<i>u32cHour</i>	1~11 / 0~23	Hour value
<i>u32cDayOfWeek</i>	RTC_SUNDAY~ RTC_SATURDAY	Day of week
<i>u32cDay</i>	1~31	Day value
<i>u32cMonth</i>	1~12	Month value
<i>u32Year</i>	0~99	Year value
<i>u32AlarmMaskDayOfWeek</i>	0/1 (Disable/Enable)	Dya of Week Alarm Mask Enable
<i>u32AlarmMaskSecond</i>	0/1 (Disable/Enable)	Second Alarm Mask Enable
<i>u32AlarmMaskMinute</i>	0/1 (Disable/Enable)	Minute Alarm Mask Enable
<i>u32AlarmMaskHour</i>	0/1 (Disable/Enable)	Hour Alarm Mask Enable
<i>u32AlarmMaskDay</i>	0/1 (Disable/Enable)	Day Alarm Mask Enable
<i>u32AlarmMaskMonth</i>	0/1 (Disable/Enable)	Month Alarm Mask Enable
<i>u32AlarmMaskYear</i>	0/1 (Disable/Enable)	Year Alarm Mask Enable

---

## 20.2. RTC API

### RTC\_Init

#### Synopsis

```
UINT32 RTC_Init (VOID)
```

#### Description

This function is to initialize RTC and install Interrupt service routine

#### Parameter

None

#### Return Value

E_SUCCESS	- Success
E_RTC_ERR_EIO	- Access RTC Failed.

#### Example

```
/* RTC Initialize */
RTC_Init();
```

## RTC\_Open

#### Synopsis

```
UINT32 RTC_Open (RTC_TIME_DATA_T *sPt )
```

#### Description

This function configures RTC current time.

#### Parameter

sPt	RTC time property and current time information
-----	--

#### Return Value

E_SUCCESS	- Success
E_RTC_ERR_EIO	- Access RTC Failed.
E_RTC_ERR_CALENDAR_VALUE	- Wrong Calendar Value
E_RTC_ERR_TIMESACLE_VALUE	- Wrong Time Scale Value
E_RTC_ERR_TIME_VALUE	- Wrong Time Value
E_RTC_ERR_DWR_VALUE	- Wrong Day Value
E_RTC_ERR_FCR_VALUE	- Wrong Compensation value

#### Example

```
/* Initialization the RTC timer */

if(RTC_Open(&sInitTime) !=E_RTC_SUCCESS)

    sysprintf("Open Fail!!\n");
```

## RTC\_Close

#### Synopsis

```
UINT32 RTC_Close (VOID)
```

**Description**

Disable AIC channel of RTC and both tick and alarm interrupt

**Parameter**

None

**Return Value**

E\_SUCCESS - Success

**Example**

```
/* Disable RTC */  
RTC_Close();
```

***RTC\_Read*****Synopsis**

UINT32 RTC\_Read (E\_RTC\_TIME\_SELECT eTime, RTC\_TIME\_DATA\_T \*sPt)

**Description**

Read current date/time or alarm date/time from RTC

**Parameter**

eTime The current/alarm time to be read

RTC\_CURRENT\_TIME - Current time

RTC\_ALARM\_TIME - Alarm time

sPt RTC time property and current time information

**Return Value**

E\_SUCCESS - Success

E\_RTC\_ERR\_EIO - Access RTC Failed.

E\_RTC\_ERR\_ENOTTY - Command not support, or incorrect parameters.

**Example**

```
/* Get the current time */  
RTC_Read(RTC_CURRENT_TIME, &sCurTime);
```

***RTC\_WriteEnable*****Synopsis**

UITN32 RTC\_WriteEnable (BOOL bEnable)

#### Description

Enable /Disable RTC register access

#### Parameter

bEnable	TRUE/FALSE
---------	------------

#### Return Value

E_SUCCESS	- Success
E_RTC_ERR_EIO	- Access RTC Failed.

#### Example

```
/* Enable RTC Access */  
  
RTC_WriteEnable(TRUE);  
  
/* Disable RTC Access */  
  
RTC_WriteEnable(FALSE);
```

## ***RTC\_DoFrequencyCompensation***

#### Synopsis

BOOL RTC\_DoFrequencyCompensation(VOID)

#### Description

Set Frequency Compensation Data if RTC crystal frequency isn't accurate.

#### Parameter

None

#### Return Value

E_SUCCESS	- Success
E_RTC_ERR_FCR_VALUE	- Can't do compensation.

#### Example

```
RTC_DoFrequencyCompensation ()
```

## ***RTC\_Ioctl***

#### Synopsis

UINT32 RTC\_Ioctl (INT32 i32Num, E\_RTC\_CMD eCmd,, UINT32 u32Arg0, UINT32 u32Arg1)

### Description

This function allows user to set some commands for application, the support commands and arguments listed in the table below (Argument 1 is reserved for feature use).

Command	Argument 0	Argument 1	Comment
RTC_IOC_IDENTIFY_LEAP_YEAR	Unsigned integer pointer to store the return leap year value	None	Get the leap year
RTC_IOC_SET_TICK_MODE	Unsigned integer stores the tick mode data	None	Set Tick mode
RTC_IOC_GET_TICK	Unsigned integer pointer to store the return tick number	None	Get the tick counter
RTC_IOC_RESTORE_TICK	None	None	Restore the tick counter
RTC_IOC_ENABLE_INT	interrupt type	None	Enable interrupt
RTC_IOC_DISABLE_INT	interrupt type	None	Disable interrupt
RTC_IOC_SET_CURRENT_TIME	None	None	Set current time
RTC_IOC_SET_ALAMRM_TIME	None	None	Set alarm time
RTC_IOC_SET_FREQUENCY	Unsigned integer stores the Frequency Compensation value	None	Set Frequency Compensation Data
RTC_IOC_SET_PWRON	None	None	Set Power on
RTC_IOC_SET_PWROFF	None	None	Set Power off
RTC_IOC_SET_POWER_OFF_PERIOD	Unsigned integer stores the power off period value : 0~15	None	Set Power Off Period
RTC_IOC_ENABLE_HW_POWEROFF	None	None	Enable H/W Power Off
RTC_IOC_DISABLE_HW_POWEROF	None	None	Disable H/W Power Off
RTC_IOC_GET_POWERKEY_STATUS	Unsigned integer pointer to store the return Power Key status	None	Get Power Key Status
RTC_IOC_SET_PSWI_CALLBACK	The call back function pointer for Power Switch Interrupts	None	Set Power Switch Interrupt Callback function
RTC_IOC_GET_SW_STATUS	Unsigned integer pointer to store the return SW Status (8 Bits)	None	Get SW Status data (8 bits)
RTC_IOC_SET_SW_STATUS	Unsigned integer stores the SW Status data (8 Bits)	None	Set SW Status data (8 bits)

RTC_IOC_SET_RELATIVE_ALARM	The call back function pointer for Relative Alarm Interrupts	Alarm time (0~4095)	Set relative alarm and install call backfunvtion
RTC_IOC_SET_POWER_KEY_DURATION	power key duration	None	Set Power Control Signal Delay
RTC_IOC_SET_POWER_KEY_LEVEL_TRIG	None	None	Set Power Key Trigger Mode – Level Trigger
RTC_IOC_SET_POWER_KEY_EDGE_TRIG	None	None	Set Power Key Trigger Mode – Edge Trigger
RTC_IOC_GET_SW_REGISTER0	Unsigned integer pointer to store the return SW Status Register 0 (32 Bits)	None	Get SW Status Register 0 data (32 bits)
RTC_IOC_SET_SW_REGISTER0	Unsigned integer stores the SW Status Register 0 (32 Bits)	None	Set SW Status Register 0 data (32 bits)
RTC_IOC_GET_SW_REGISTER1	Unsigned integer pointer to store the return SW Status Register 1 (32 Bits)	None	Get SW Status Register 1 data (32 bits)
RTC_IOC_SET_SW_REGISTER1	Unsigned integer stores the SW Status Register 1 (32 Bits)	None	Set SW Status Register 1 data (32 bits)

**Parameter**

sicFeature	SIC_SET_CLOCK, SIC_SET_CALLBACK
u32Arg0	Depend on feature setting
u32Arg1	Depend on feature setting

**Return Value**

None

**Example**

```
/* Set Tick setting */

RTC_Ioctl(0,RTC_IOC_SET_TICK_MODE, (UINT32)&sTick,0);

/* Enable RTC Tick Interrupt and install tick call back function */

RTC_Ioctl(0,RTC_IOC_ENABLE_INT, (UINT32)RTC_TICK_INT,0);

/* Press Power Key during 6 sec to Power off */

RTC_Ioctl(0, RTC_IOC_SET_POWER_OFF_PERIOD, 6, 0);
```

```
/* Install the callback function for Power Key Press */

RTC_Ioctl(0, RTC_IOC_SET_PSWI_CALLBACK, (UINT32)PowerKeyPress, 0);

/* Enable Hardware Power off */

RTC_Ioctl(0, RTC_IOC_ENABLE_HW_POWEROFF, 0, 0);

/* Query Power Key Status */

RTC_Ioctl(0, RTC_IOC_GET_POWERKEY_STATUS, (UINT32)&u32PowerKeyStatus, 0);

/* Power Off - S/W can call the API to power off any time he wants */

RTC_Ioctl(0, RTC_IOC_SET_POWER_OFF, 0, 0);

/* Enable RTC Relative alarm Interrupt and install call back function */

RTC_Ioctl(0,RTC_IOC_SET_RELATIVE_ALARM, 10, (UINT32)RTC_Relative_AlarmISR);

/* Set Power Control Signal Delay */

    RTC_Ioctl(0,RTC_IOC_SET_POWER_KEY_DELAY, 1, 0);

Delay time Formula

Minimum Power key duration = 0.25*(POWER_KEY_DURATION+1) sec
```

---

## 20.3. Example code

The demo code test “Time display”, “Absolute Alarm”, “Relative Alarm”, “Power down Wakeup”, “Software Power Off (Normal Case) Control Flow”, “Hardware Power Off (System Crash) Control Flow”, “Software Force to Power Off”, and “Alarm Mask”. Please refer to the RTC sample code of SDK Non-OS.

---

## 20.4. Error Code Table

Code Name	Value	Description
E_RTC_SUCCESS	0	Operation success
E_RTC_ERR_CALENDAR_VALUE	1	Wrong Calendar Value
E_RTC_ERR_TIMESACLE_VALUE	2	Wrong Time Scale Value
E_RTC_ERR_TIME_VALUE	3	Wrong Time Value
E_RTC_ERR_DWR_VALUE	4	Wrong Day Value
E_RTC_ERR_FCR_VALUE	5	Wrong Compensation value
E_RTC_ERR_EIO	6	Access RTC Failed.
E_RTC_ERR_ENOTTY	7	Command not support, or parameter incorrect.
E_RTC_ERR_ENODEV	8	Interface number incorrect.

# 21. RTC Power Control Library Overview

N9H26 RTC provides a power control function for system power control. The application note describes the function in detail.

## 21.1. Power Control Function Overview

RTC Power Control supports power on and power off functions:

### ***Power On***

N9H26 can be power on by Power Key or RTC alarm.

### ***Power Off***

RTC provides three modes to Power off.

- Hardware Force Power Off
  - Hardware Force Power Off function is to power off system by holding the power key about the specified period even when software crash.
- Software Force Power Off
  - When Power Key is pressed, system can be power off by software.
- Software Power off
  - When Power Key is released, system can be power off by software.

## 21.2. Preliminary Definition

### ***Power Key (PWRKEY)***

PWRKEY is the power key for user to power On/Off system power except RTC power.

### ***Power Control Signal (PWCE)***

PWCE is the power control signal to control the power control unit.

## **Power Key Status (PWR\_KEY)**

Power Key Status

- 1: Indicated the power key status is high
- 0: Indicated the power key is pressed to low.

## **Software Status (SW\_STATUS)**

RTC provides 8 bits to store software information.

## **Hardware Force Power Off Enable (HW\_PCLR\_EN)**

RTC provides a Hardware Force Power Off function to power off system by holding the power key about the specified period even when software crash. The bit is to enable the Hardware Force Power Off function.

## **Hardware Force Power Off Period (PCLR\_TIME)**

If user holds the power key about the specified period, the system will be power off. The following table is the period setting.

Setting	Hold Time to power off	Setting	Hold Time to power off
0	2~3 second	8	10~11 seconds
1	3~4 second	9	11~12 seconds
2	4~5 seconds	10	12~13 seconds
3	5~6 seconds	11	13~14 seconds
4	6~7 seconds	12	14~15 seconds
5	7~8 seconds	13	15~16 seconds
6	8~9 seconds	14	16~17 seconds
7	9~10 seconds	15	17~18 seconds

## **Power on (PWR\_ON)**

The bit is used to control PWCE to control system power.

- PWCE will change to high state when this bit changes from 0 to 1 and the Power Key is pressed.
- PWCE will change to low state when this bit changes from 1 to 0 and the Power Key is released.

## **Power-down Mode**

Whole system is power-off except RTC.

## **Power key duration (POWER\_KEY\_DURATION\_LENGTH)**

The delay time between Power Key pressed and PWCE high (or Power Key released and PWCE low)

- Minimum power key duration =  $0.25 * (\text{POWER\_KEY\_DURATION} + 1)$  sec.

### Interrupt Enable & Status

RTC provides three interrupts (They can't keep when Power-down Mode)

- Alarm Interrupt (AIER & AI)
- Time Tick Interrupt (TIER & TI)
- Power Switch Interrupt (PSWIER & PSWI)

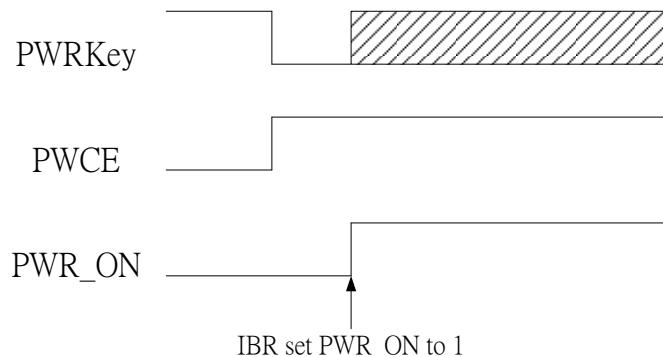
It indicates that the Power Key has been pressed.

## 21.2.1. System Power On Control Flow

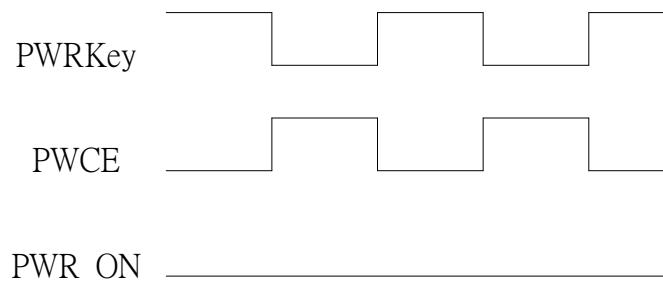
Two ways to power on system:

### Power On from Power Key

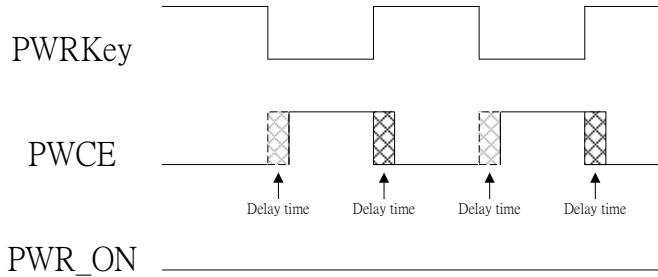
User presses the Power Key to make the Power Control Signal, PWCE to high. If PWR\_ON bit is set, the Power Key can be released and the PWCE will be keep on. When system is power on, IBR will set PWR\_ON first.



If PWR\_ON doesn't be set to 1, the PWCE will back to low when the Power Key is released.



And RTC supports a function (**POWER\_KEY\_DURATION\_LENGTH**) to postpone the time to set PWCE to high when pressing Power Key or to postpone the time to set PWCE to low when releasing Power Key. The delay time is from 62ms to 868ms. The function is default disabled. User can enable it after first power on and the setting can be kept when RTC is powered even whole system is power-off.



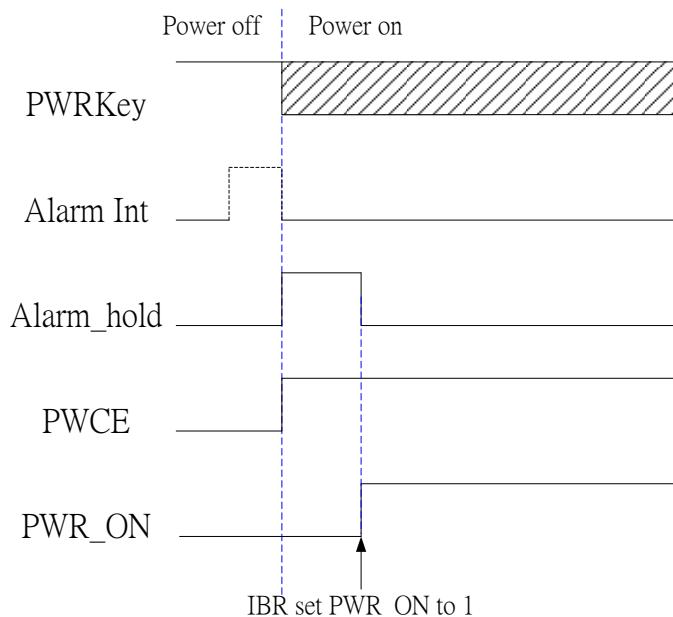
[Note 1]. The function (POWER\_KEY\_DURATION\_LENGTH) only works only when Power Key is pressed or released.

[Note 2]. POWER KEY TRIGGER MODE

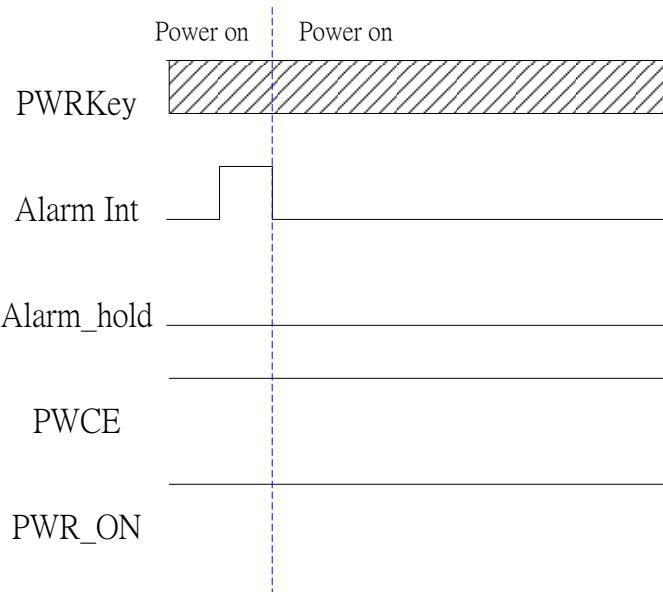
- ◊ 1: EDGE TRIGE
  - RTC is powered on while power key is pressed longer than programmed duration and then released
- ◊ 0: LEVEL TRIGGER
  - RTC is powered on while power key is pressed longer programmed duration

### **Power On from RTC Alarm**

In Power-down mode, if RTC alarm occurs, RTC keeps an internal Alarm\_hold signal to force the PWCE to high. After PWR\_ON is set, the Alarm\_hold signal is clear and PWCE is released, then PWCE keeps on by PWR\_ON. When system is power on, IBR will set PWR\_ON first.



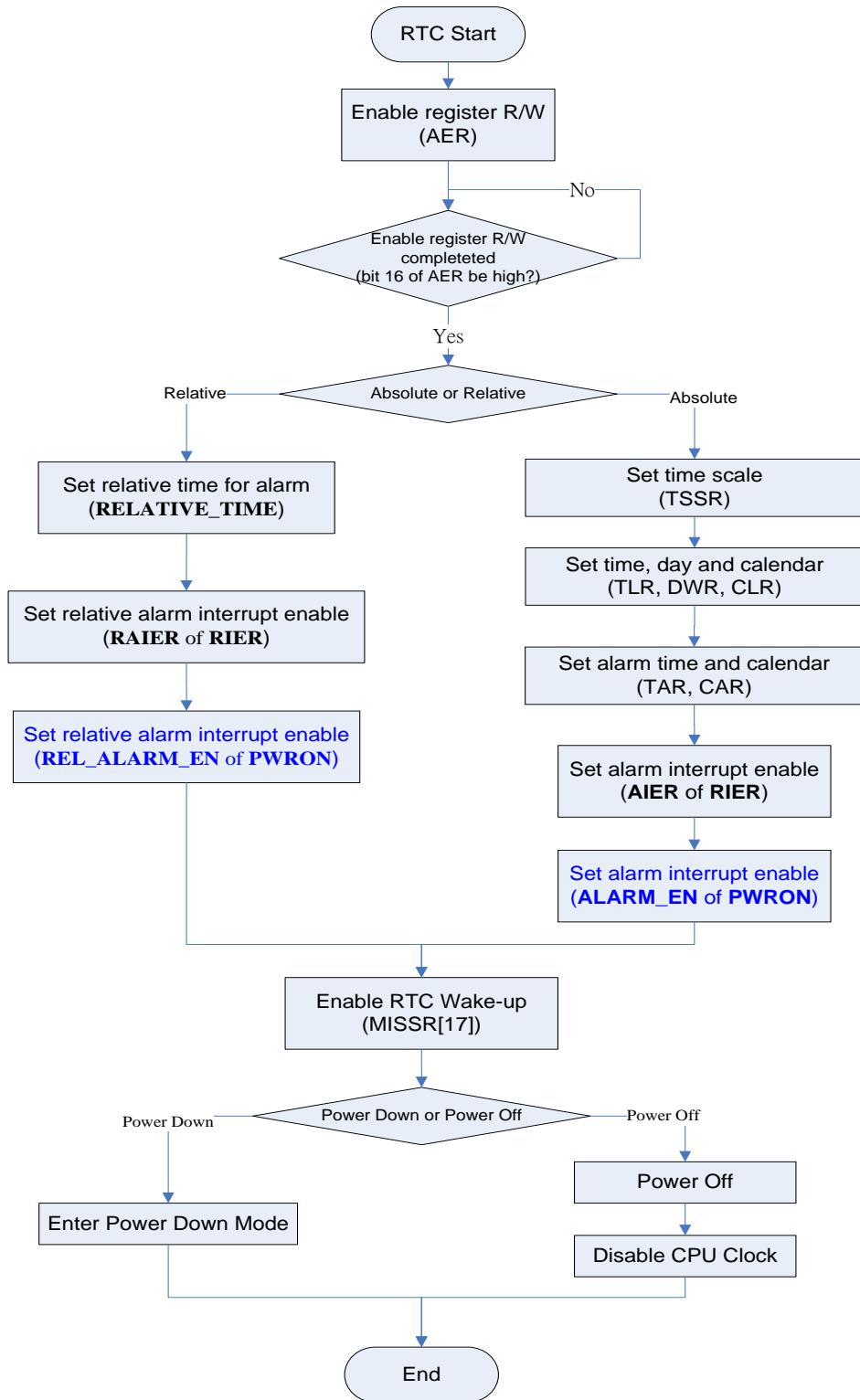
In normal mode, if RTC alarm occurs, the Alarm\_hold signal isn't set. Therefore, PWCE doesn't force to high by Alarm\_hold.



[Note1] Alarm\_hold is set only when PWR\_ON is low.

[Note2] If user doesn't want to power on by RTC Alarm, user must let the alarm condition can't be reached to avoid false wakeup. Ex, Clear the alarm time.

The Power down Wakeup from RTC alarm programming flow is as follows.



[Note1] If user won't enable wakeup function, please don't enable the alarm enable bit of **PWRON** (**REL\_ALARM\_EN** or **ALARM\_EN**)

### **Power On Source Judgment**

Because the interrupt status flag can't keep in Power-down Mode, user can't use interrupt flag to determine which Power on Source is. The way to confirm it is to check the RTC Alarm time and Current time right after IBR. If they aren't the same (or close), system is power-on from Power Key. The following table is all the cases of Power on Source.

Power Key Status	Time Match	Power on Source
Pressed	No	Power Key
Pressed	Yes	Power Key and RTC Alarm
<b>Released</b>	<b>No</b>	<b>Reset Key/WDT Reset/Power on Exception</b>
Released	Yes	RTC Alarm

### **Power On Exception**

When user removes N9H26's battery and put it back later, N9H26 will be power on automatically without Power Key pressed or RTC Alarm (like user presses the reset button). Using the above-mentioned condition, we can let N9H26 power off in the situation. And the way to differentiate between Reset and Power Exception (ex. Removing N9H26's battery) is to write a specific key word in DRAM or SRAM when N9H26 is power on and clear it when N9H26 is power off normally.(Data in SRAM or DRAM will be lost when N9H26's power is off)

	Specific Key word	Power on source	Software action
<b>Hardware or Software Power Off</b>	Unknown value	Power Key or RTC Alarm	Keep Power On
<b>Reset/WDT</b>	Not Changed	N/A	Keep Power On
<b>Power on Exception</b>	<b>Unknown value</b>	<b>N/A</b>	<b>Power off</b>

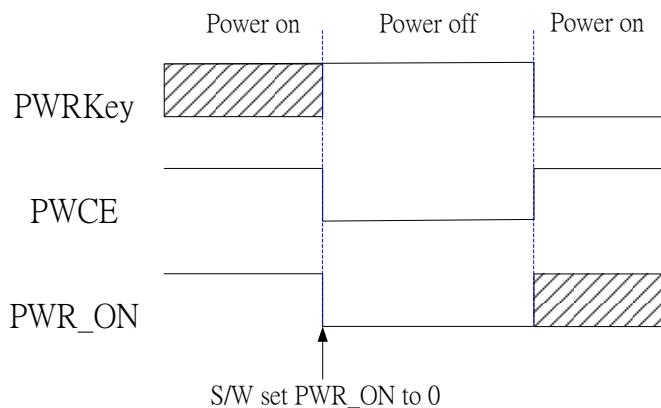
---

## **21.2.2. System Power Off Control Flow**

There are two modes to power off system.

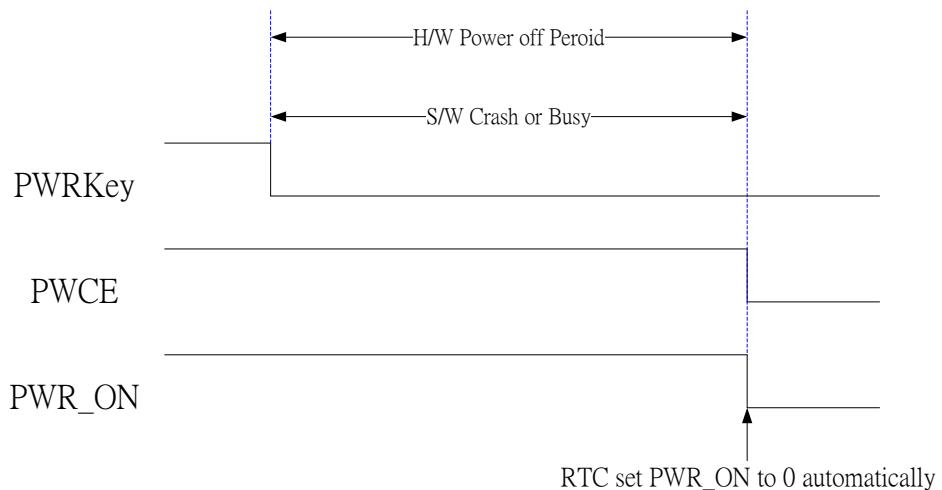
### **Software Power Off**

If user wants to power off system, ex. Specified time to power off or user press power key to power off system, user must set PWR\_ON to 0 to power off system any time he wants.



### **Hardware Force Power Off**

RTC provides a Hardware Force Power Off function to power off system even when software crash. After enabling the function (HW\_PCLR\_EN), if user holds the power key about the specified period, the system will be power off.




---

### **21.3. Power Control Flow Truth Table**

Here is the Power Control Flow Truth Table except the force mode.

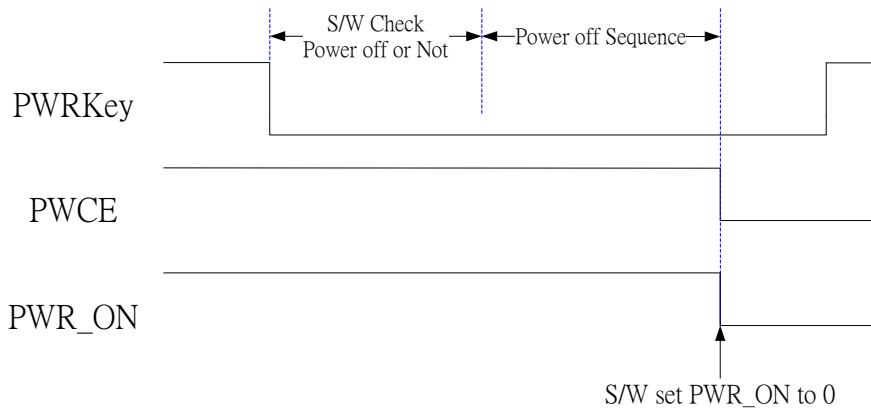
Step	Input					Output	Note
	X1	X2	X3	X4	Y		
PWRKey	PWR_ON	RST_	Alarm-hold	PWCE			

<b>Power key control flow</b>						
0	1	0	0	0	0	RTC powered only (Default state)
1	0	0	X	0	1	Press key, Power On
2	0	1	1	0	1	keep key pressed & S/W Set X2, Power On
3	1	1	1	0	1	Release key, Power keep On
4	0	1	1	0	1	Press key, get INT, intend to power Off
5	1	0	1	0	0	Release key & S/W clean X2, power Off Or S/W clean X2 , don't need press key, power off
<b>Reset</b>						
0	X	1	0	0	1	RST_ active, still keep power when X2=1
<b>Alarm flow</b>						
0	1	0	0	0	0	RTC powered only (Default state)
1	1	0	X	1	1	Alarm
2	1	1	1	0	1	IBR set X2
<b>PWCE (open drain output) = /PWRKEY + Alarm-hold + /PWR_ON when HW_PCLR_EN=0</b> X1, internal pull-up X2, it is R/W able X4, internal signal. Will be 1 after alarm if X2=0, and be cleared after X2 set. There is Interrupt from key be pressed						

## 21.4. Power off Flow Example

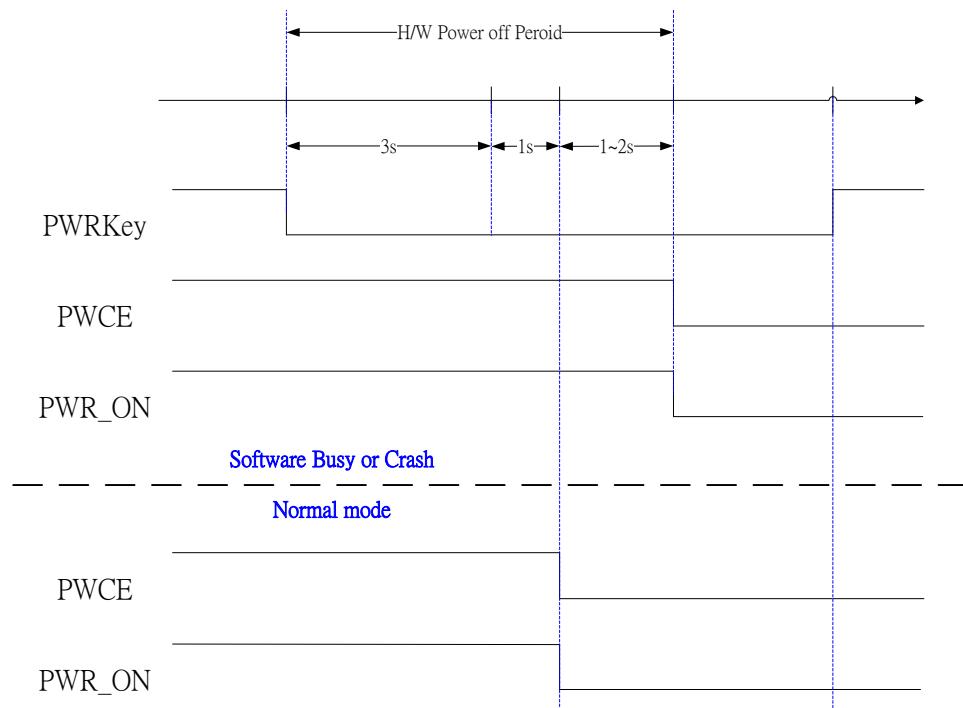
### Normal Power Off Flow

When user gets the Power Switch Interrupt, user can decide to do the power off sequence or not to do it. If the Power off sequence is done, user can set PWR\_ON to 0 to power off system right away.



### **Software Busy/Crash Power Off Flow**

- The example sets the Hardware Power off period as 6 seconds.
- It takes 3 seconds to decide to do the power off sequence or not to do it.
- It takes 1 second to do power off sequence.



# 22. SDIO Library Overview

N9H26 Non-OS library consists of a sets of libraries. These libraries are built to access those on-chip functions such as VPOST, APU, SIC, USBH, USBD, GPIO, I2C, SPI and UART, as well as File System (NVTFAT), TCP/IP protocol (lwip), USB MassStorage devices (UMAS) and NAND Flash devices (GNAND). This document describes the provided APIs of SDIO library. With these APIs, user can quickly build a binary target for SDIO library on N9H26 micro processor.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

---

## 22.1. SDIO Library

This library is designed to make user application access N9H26 SDIO (Secure-Digital Input / Output) controller more easily. This interface can directly connect to SD card.

The SDIO library has the following features:

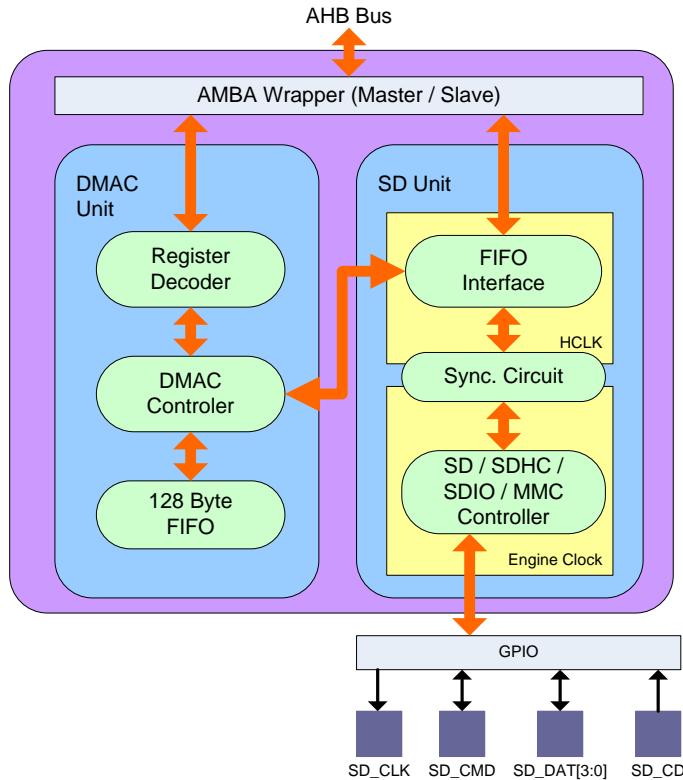
- Support single DMA channel and address in non-word boundary.
- Support SD/SDHC/SDIO/MMC card.

---

## 22.2. Programming Guide

### **System Overview**

The SDIO controller of N9H26 chip has DMAC unit and SD unit. The DMAC unit provides a DMA (Direct Memory Access) function for SD unit to exchange data between system memory (ex. SDRAM) and shared buffer (128 bytes), and the SD unit control the interface of SD/SDHC/SDIO/MMC. The SDIO controller can support SD/SDHC/SDIO/MMC card and the SD unit is cooperated with DMAC unit to provide a fast data transfer between system memory and cards. The block diagram of SDIO controller is shown as following:



## 22.3. SDIO API

### *sdioOpen*

#### Synopsis

```
void sdioOpen (VOID)
```

#### Description

`sdioOpen()` will initialize the SDIO and DMAC interface hardware. It configures GPIO to SDIO mode, and installs ISR. This function is board dependent. It probably needs some modifications before it can work properly on your target board.

#### Parameter

None

#### Return Value

None

#### Example

```
/* initialize SDIO mode */

sdioIcntl(SDIO_SET_CLOCK, 192000, 0, 0); /* clock from PLL */

sdioOpen();
```

***sdioClose*****Synopsis**

void sdioClose (VOID)

**Description**

sdioClose() will Close the SDIO and DMAC interface hardware. It configures GPIO to close DMAC and disable ISR for SDIO.

**Parameter**

None

**Return Value**

None

**Example**

```
sdioClose();
```

***sdioIcntl*****Synopsis**

VOID sdioIcntl (INT32 sdioFeature, INT32 sdioArg0, INT32 sdioArg1, INT32 sdioArg2)

**Description**

sdioIcntl() allows user set engine clock and callback functions, the support features and arguments listed in the table below.

Feature	Argument 0	Argument 1	Argument 2
SDIO_SET_CLOCK	AHB clock by KHz	None	None
SDIO_SET_CALLBACK	Card type (FMI_SDIO_CARD / FMI_SDIO1_CARD)	SD Card Remove callback function	SD Card Insert callback function
SDIO_GET_CARD_STATUS	Pointer to return value of SD card status	None	None

**Parameter**

sdioFeature	SDIO_SET_CLOCK, SDIO_SET_CALLBACK, SDIO_GET_CARD_STATUS
sdioArg0	Depend on feature setting
sdioArg1	Depend on feature setting
sdioArg2	Depend on feature setting

**Return Value**

For SDIO\_GET\_CARD\_STATUS, the card status assign to sdioArg0. The value TRUE means SD card inserted, FALSE means SD card removed.

**Example**

Refer to the example code of `sdioOpen()`.

## 22.4. SDIO API

### ***sdioSdOpen***

**Synopsis**

INT sdioSdOpen (void)	open SD card 0
INT sdioSdOpen0 (void)	open SD card 0
INT sdioSdOpen1 (void)	open SD card 1

**Description**

This function initialize the SDIO host interface and program the SD card from identify mode to stand-by mode.

**Parameter**

None

**Return Value**

>0	– Total sector number of SD card
Otherwise	– Refer error code defined in Error Code Table

**Example**

```
if (sdioSdOpen0() <= 0)      // Open SDIO port 0
{
    printf("Error in initializing SD card !! \n");
}
```

```
/* handle error status */  
}
```

## **sdioSdClose**

### **Synopsis**

void sdioSdClose (void)	close SD card 0
void sdioSdClose0 (void)	close SD card 0
void sdioSdClose1 (void)	close SD card 1

### **Description**

This function close the SDIO host interface.

### **Parameter**

None

### **Return Value**

None

### **Example**

```
sdioSdClose(); // Close SDIO port 0
```

## **sdioSdRead**

### **Synopsis**

INT sdioSdRead (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr)	
	for SD card 0
INT sdioSdRead0 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr)	
	for SD card 0
INT sdioSdRead1 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr)	
	for SD card 1

### **Description**

This function will read the data from SD card.

### **Parameter**

sdSectorNo	Sector No. to get the data from
sdSectorCount	Sector count of this access
sdTargetAddr	The address which data upload to SDRAM

### **Return Value**

0	- On success
FMISDIO_TIMEOUT	- Access timeout
FMISDIO_NO_SD_CARD	- Card removed
FMISDIO_SD_CRC7_ERROR	- Command/Response error
FMISDIO_SD_CRC16_ERROR	- Data transfer error

**Example**

```
#define FMI_TEST_SIZE (512*128)

__align(4096) UINT8 fmiReadBackBuffer[FMI_TEST_SIZE];

// read 128 sectors data from SD card sector address 300.

status = sdioSdRead(300, FMI_TEST_SIZE/512, (unsigned int)fmiReadBackBuffer);
```

**sdioSdWrite****Synopsis**

INT sdioSdWrite (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 0

INT sdioSdWrite0 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 0

INT sdioSdWrite1 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 1

**Description**

This function write the data into SD card.

**Parameter**

sdSectorNo	Sector No. to get the data from
sdSectorCount	Sector count of this access
sdSourceAddr	The address which download data from SDRAM

**Return Value**

0	- On success
FMISDIO_TIMEOUT	- Access timeout
FMISDIO_NO_SD_CARD	- Card removed
FMISDIO_SD_CRC7_ERROR	- Command/Response error
FMISDIO_SD_CRC_ERROR	- Data transfer error

**Example**

```
#define FMI_TEST_SIZE (512*128)
```

```
__align(4096) UINT8 fmiFlash_Buf[FMI_TEST_SIZE];  
  
// write 128 sectors data to SD card sector address 3000.  
  
status = sdioSdWrite(3000, FMI_TEST_SIZE/512, (unsigned int)fmiFlash_Buf);
```

---

## 22.5. Example code

The demo code test the SD card by read / write / compare please refer to the SDIO sample code of SDK Non-OS.

---

## 22.6. Error Code Table

Code Name	Value	Description
FMISDIO_TIMEOUT	0xFFFF0101	Access timeout
FMISDIO_NO_MEMORY	0xFFFF0102	No available memory
<b>Error Code for SD Card</b>		
FMISDIO_NO_SD_CARD	0xFFFF0110	No SD card insert
FMISDIO_ERR_DEVICE	0xFFFF0111	Unknown device type
FMISDIO_SD_SELECT_ERROR	0xFFFF0113	Select card from identify mode to stand-by mode error
FMISDIO_SD_INIT_ERROR	0xFFFF0115	SD Card initial and identify error
FMISDIO_SD_CRC7_ERROR	0xFFFF0116	Command/Response error
FMISDIO_SD_CRC16_ERROR	0xFFFF0117	Data reading error
FMISDIO_SD_CRC_ERROR	0xFFFF0118	Data writing error
FMISDIO_SD_CMD8_ERROR	0xFFFF0119	SD command 8 error

# 23. SIC Library Overview

N9H26 Non-OS library consists of a sets of libraries. These libraries are built to access those on-chip functions such as VPOST, APU, SIC, USBH, USBD, GPIO, I2C, SPI and UART, as well as File System (NVTFAT), TCP/IP protocol (lwip), USB MassStorage devices (UMAS) and NAND Flash devices (GNAND). This document describes the provided APIs of SIC library. With these APIs, user can quickly build a binary target for SIC library on N9H26 micro processor.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

---

## 23.1. Storage Interface Controller Library

This library is designed to make user application access N9H26 Storage Interface Controller (SIC) more easily. This interface can directly connect to SD and NAND Flash.

The SIC library has the following features:

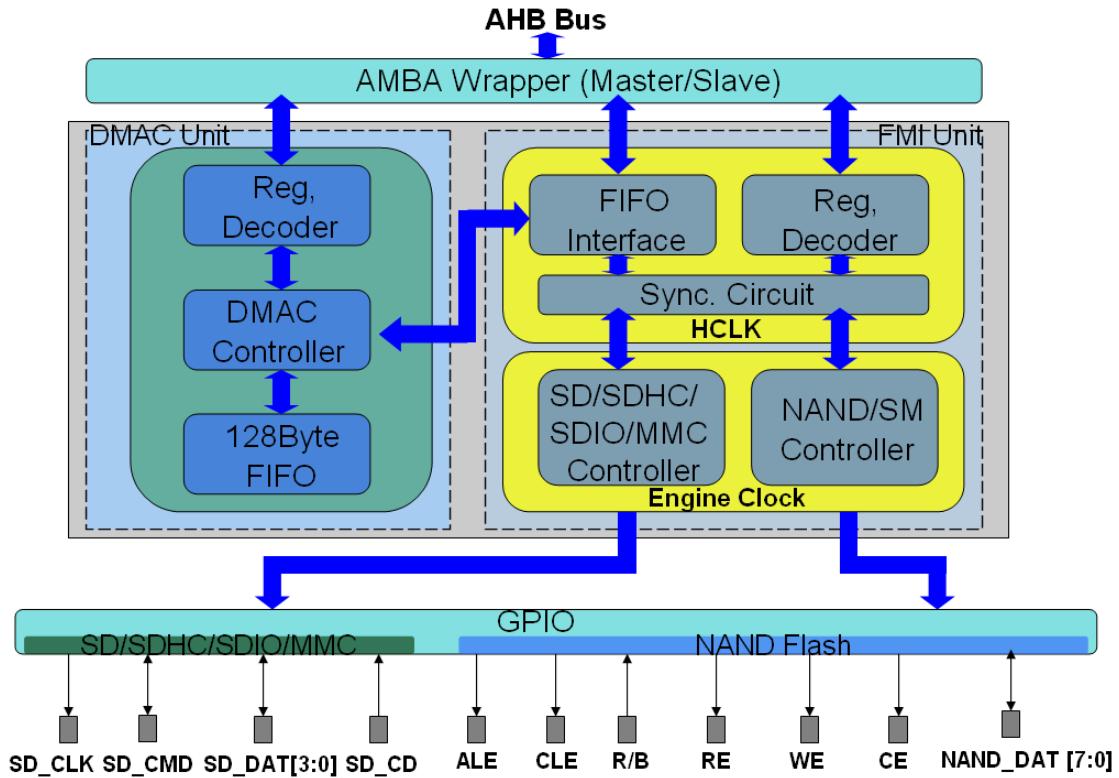
- Support single DMA channel and address in non-word boundary.
- Support SD/SDHC/SDIO/MMC card.
- Supports SLC and MLC NAND type Flash.
- Adjustable NAND page sizes. (512 / 2048 / 4096 / 8192 bytes + spare area)
- Support up to 4bit / 8bit / 12bit / 15bit / 24bit hardware ECC calculation circuit to protect data communication.
- Programmable NAND/SM timing cycle.

---

## 23.2. Programming Guide

### System Overview

The Storage Interface Controller (SIC) of N9H26 chip has SIC\_DMAC unit and SIC\_FMI unit. The SIC\_DMAC unit provides a DMA (Direct Memory Access) function for FMI to exchange data between system memory (ex. SDRAM) and shared buffer (128 bytes), and the SIC\_FMI unit control the interface of SD/SDHC/SDIO/MMC or NAND/SM. The storage interface controller can support SD/SDHC/SDIO/MMC card and NAND-type flash and the FMI is cooperated with DMAC to provide a fast data transfer between system memory and cards. The block diagram of SIC controller is shown as following:



### **NAND Driver and GNAND Library**

The SIC library provide NAND driver API to access NAND chip directly. However, the NAND driver don't support management features for NAND chip that don't guarantee all blocks are valid. The management features include bad block management, garbage collection, and wear-leveling. We provide GNAND library to support these management features and suggest to use GNAND library before using SIC NAND driver. Please refer to document "N9H26 Non-OS GNAND Library Reference Guide" for GNAND library detail information.

---

## **23.3. SIC API**

### ***sicOpen***

#### **Synopsis**

```
void sicOpen (VOID)
```

#### **Description**

sicOpen() will initialize the SIC and DMAC interface hardware. It configures GPIO to FMI mode, and installs ISR. This function is board dependent. It probably needs some modifications before it can work properly on your target board.

**Parameter**

None

**Return Value**

None

**Example**

```
/* initialize SIC to FMI (Flash Memory Interface controller) mode */

sicIoctl(SIC_SET_CLOCK, 192000, 0, 0); /* clock from PLL */

sicOpen();
```

***sicClose*****Synopsis**

void sicClose (VOID)

**Description**

sicClose() will Close the SIC and DMAC interface hardware. It configures GPIO to close DMAC and disable ISR for SIC.

**Parameter**

None

**Return Value**

None

**Example**

```
sicClose();
```

***sicIoctl*****Synopsis**

VOID sicIoctl (INT32 sicFeature, INT32 sicArg0, INT32 sicArg1, INT32 sicArg2)

**Description**

sicIoctl() allows user set engine clock and callback functions, the support features and arguments listed in the table below.

Feature	Argument 0	Argument 1	Argument 2
SIC_SET_CLOCK	AHB clock by KHz	None	None
SIC_SET_CALLBACK	Card type (FMI_SD_CA RD)	SD Card Remove callback function	SD Card Insert callback function
SIC_GET_CARD_STATUS	Pointer to return value of SD card status	None	None

**Parameter**

sicFeature      SIC\_SET\_CLOCK, SIC\_SET\_CALLBACK, SIC\_GET\_CARD\_STATUS  
 sicArg0          Depend on feature setting  
 sicArg1          Depend on feature setting  
 sicArg2          Depend on feature setting

**Return Value**

For SIC\_GET\_CARD\_STATUS, the card status assign to sicArg0. The value TRUE means SD card inserted, FALSE means SD card removed.

**Example**

Refer to the example code of sicOpen().

## 23.4. SIC / SD API

### ***sicSdOpen***

**Synopsis**

INT sicSdOpen (void)	open SD card 0
INT sicSdOpen0 (void)	open SD card 0
INT sicSdOpen1 (void)	open SD card 1
INT sicSdOpen2 (void)	open SD card 2

**Description**

This function initialize the SD host interface and program the SD card from identify mode to stand-by mode.

**Parameter**

None

**Return Value**

- >0 – Total sector number of SD card
- Otherwise – Refer error code defined in Error Code Table

#### Example

```
if (sicSdOpen0() <= 0) // Open SD port 0
{
    printf("Error in initializing SD card !! \n");
    /* handle error status */
}
```

## **sicSdClose**

#### Synopsis

void sicSdClose (void)	close SD card 0
void sicSdClose0 (void)	close SD card 0
void sicSdClose1 (void)	close SD card 1
void sicSdClose2 (void)	close SD card 2

#### Description

This function close the SD host interface.

#### Parameter

None

#### Return Value

None

#### Example

```
sicSdClose(); // Close SD port 0
```

## **sicSdRead**

#### Synopsis

INT sicSdRead (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr) for SD card 0
INT sicSdRead0 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr) for SD card 0
INT sicSdRead1 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr) for SD card 1

INT sicSdRead2 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdTargetAddr)  
for SD card 2

### Description

This function will read the data from SD card.

### Parameter

sdSectorNo	Sector No. to get the data from
sdSectorCount	Sector count of this access
sdTargetAddr	The address which data upload to SDRAM

### Return Value

0	- On success
FMI_TIMEOUT	- Access timeout
FMI_NO_SD_CARD	- Card removed
FMI_SD_CRC7_ERROR	- Command/Response error
FMI_SD_CRC16_ERROR	- Data transfer error

### Example

```
#define FMI_TEST_SIZE  (512*128)

__align(4096) UINT8 fmiReadBackBuffer[FMI_TEST_SIZE];

// read 128 sectors data from SD card sector address 3000.

status = sicSdRead(3000, FMI_TEST_SIZE/512, (unsigned int)fmiReadBackBuffer);
```

## sicSdWrite

### Synopsis

INT sicSdWrite (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 0

INT sicSdWrite0 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 0

INT sicSdWrite1 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 1

INT sicSdWrite2 (INT32 sdSectorNo, INT32 sdSectorCount, INT32 sdSourceAddr)  
for SD card 2

### Description

This function write the data into SD card.

### Parameter

sdSectorNo	Sector No. to get the data from
sdSectorCount	Sector count of this access
sdSourceAddr	The address which download data from SDRAM

**Return Value**

0	- On success
FMI_TIMEOUT	- Access timeout
FMI_NO_SD_CARD	- Card removed
FMI_SD_CRC7_ERROR	- Command/Response error
FMI_SD_CRC_ERROR	- Data transfer error

**Example**

```
#define FMI_TEST_SIZE (512*128)

__align(4096) UINT8 fmiFlash_Buf[FMI_TEST_SIZE];

// write 128 sectors data to SD card sector address 3000.

status = sicSdWrite(3000, FMI_TEST_SIZE/512, (unsigned int)fmiFlash_Buf);
```

---

## 23.5. SIC / NAND API

***nandInit0*****Synopsis**

INT nandInit0 (NDISK_T *NDISK_info)	for NAND chip 0
INT nandInit1 (NDISK_T *NDISK_info)	for NAND chip 1

**Description**

This function configures SIC register to initial DMAC and FMI to NAND mode. It also initial the internal data structure for future use. Since different NAND chip need different parameters, nandInit0() also read the product ID from NAND chip to try to configure correct parameters for it. This function is NAND chip dependent. It probably needs some modifications before it can work properly on your target NAND chip.

**Parameter**

NDISK_info	The internal data for NAND disk information. nandInit0() will initial it and return to caller.
------------	--

**Return Value**

0	- Success
---	-----------

Otherwise      - Refer error code defined in Error Code Table

#### Example

```
NDISK_T *ptMassNDisk;
NDISK_T MassNDisk;
ptMassNDisk = (NDISK_T *)&MassNDisk;
if (nandInit0(ptMassNDisk) < 0)
{
    printf("NAND initial fail !!\n");
    /* handle error status */
}
```

### ***nand\_ioctl***

#### Synopsis

INT nand\_ioctl (INT param1, INT param2, INT param3, INT param4)

#### Description

nand\_ioctl() is reserved for I/O control utility for NAND. It is empty now and could support new functions in the future.

#### Parameter

param1	Depend on feature setting
param2	Depend on feature setting
param3	Depend on feature setting
param4	Depend on feature setting

#### Return Value

0	- Success
Otherwise	- Refer error code defined in Error Code Table

#### Example

None

### ***nandpread0***

#### Synopsis

INT nandread0 (INT PBA, INT page, UINT8 *buff)	for NAND chip 0
INT nandread1 (INT PBA, INT page, UINT8 *buff)	for NAND chip 1

**Description**

This function read a page of data from NAND.

**Parameter**

PBA	physical block address of NAND that read data from.
page	page number in PBA block that read data from.
buff	the RAM address to store the reading data.

**Return Value**

0	- Success
Otherwise	- Refer error code defined in Error Code Table

**Example**

```
__align(32) UINT8 fmiFlash_Buf[PAGE_SIZE];

// read a page of data from NAND block 5 page 10 and store at fmiFlash_Buf

status = nandread0(5, 10, fmiFlash_Buf);

if (status < 0)

{

/* handle error status */

}
```

***nandpwrite0*****Synopsis**

INT nandpwrite0 (INT PBA, INT page, UINT8 *buff)	for NAND chip 0
INT nandpwrite1 (INT PBA, INT page, UINT8 *buff)	for NAND chip 1

**Description**

This function write a page of data to NAND.

**Parameter**

PBA	physical block address of NAND to write data.
page	page number in PBA block to write data.
buff	the RAM address to get the writing data.

**Return Value**

- |           |  |
|-----------|--|
| 0         | - Success                                      |
| Otherwise | - Refer error code defined in Error Code Table |

**Example**

```
__align(32) UINT8 fmiFlash_Buf[PAGE_SIZE];

// write a page of data from fmiFlash_Buf to NAND block 5 page 10

status = nandpwrite0(5, 10, fmiFlash_Buf);

if (status < 0)

{

/* handle error status */

}
```

***nand\_is\_page\_dirty0*****Synopsis**

- |   |                 |
|---|-----------------|
| INT nand_is_page_dirty0 (INT PBA, INT page) | for NAND chip 0 |
| INT nand_is_page_dirty1 (INT PBA, INT page) | for NAND chip 1 |

**Description**

This function check the redundancy area of the NAND page and return the dirty status to indicate whether a page is dirty or not. Dirty page means you cannot write data to it directly. You have to erase this block first to clean it.

**Parameter**

- |      |   |
|------|---|
| PBA  | physical block address of NAND to check the dirty status. |
| page | page number in PBA block to check the dirty status.       |

**Return Value**

- |   |  |
|---|--|
| 0 | - Clean page that can write data directly    |
| 1 | - Dirty page that cannot write data directly |

**Example**

```
/* check dirty status for NAND block 5 page 10 */

status = nand_is_page_dirty0(5, 10);

if (status == 0)

{
```

```
printf("This page is clean !! You can write data to it directly.\n");
}
else
{
    printf("This page is dirty !! You cannot write data to it directly.\n");
}
```

## ***nand\_is\_valid\_block0***

### **Synopsis**

INT nand_is_valid_block0 (INT PBA)	for NAND chip 0
INT nand_is_valid_block1 (INT PBA)	for NAND chip 1

### **Description**

This function check the redundancy area of the NAND block and return the valid status to indicate whether a block is valid or not. Valid block page means you can write data to it directly or indirectly (maybe need to erase block first). You cannot write data into a invalid block always since it could be a bad block.

### **Parameter**

PBA	physical block address of NAND to check the valid status.
-----	---

### **Return Value**

0	- Valid block that can write data into it directly or indirectly
1	- Invalid block that cannot write data into it always

### **Example**

```
/* check valid status for NAND block 5 */

status = nand_is_valid_block0(5);

if (status == 0)

{
    printf("This block is valid !! You can write data to it directly or indirectly.\n");
}

else

{
```

```
    printf("This block is invalid !! You cannot write data to it always.\n");
}
```

## ***nand\_block\_erase0***

### **Synopsis**

INT nand_block_erase0 (INT PBA)	for NAND chip 0
INT nand_block_erase1 (INT PBA)	for NAND chip 1

### **Description**

This function erase a block. You should call this API first if you want to write data into a dirty page.

### **Parameter**

PBA            physical block address of NAND to erase.

### **Return Value**

0	- Erase block successfully
Otherwise	- Refer error code defined in Error Code Table

### **Example**

```
/* erase NAND block 5 */

status = nand_block_erase0(5);

if (status == 0)

{

printf("This block is erased !!\n");

}

else

{

printf("This block erase fail !!\n");

}
```

## ***nand\_chip\_erase0***

### **Synopsis**

INT nand_chip_erase0 (VOID)	for NAND chip 0
-----------------------------	-----------------

INT nand\_chip\_erase1 (VOID) for NAND chip 1

#### Description

This function erase all blocks in NAND chip. All data in chip will lost that include information for GNAND library.

#### Parameter

None

#### Return Value

0	- Erase chip successfully
Otherwise	- Refer error code defined in Error Code Table

#### Example

```
/* erase whole NAND chip */

status = nand_chip_erase0();

if (status == 0)

{
    printf("This chip is erased !!\n");

}

else

{
    printf("This chip erase fail !!\n");
}
```

---

## 23.6. Example code

The demo code test the flash card by read / write / compare please refer to the SIC sample code of SDK Non-OS.

---

## 23.7. Error Code Table

Code Name	Value	Description
FMI_TIMEOUT	0xFFFF0101	Access timeout

FMI_NO_MEMORY	0xFFFF0102	No available memory
<b>Error Code for SD Card</b>		
FMI_NO_SD_CARD	0xFFFF0110	No SD card insert
FMI_ERR_DEVICE	0xFFFF0111	Unknown device type
FMI_SD_SELECT_ERROR	0xFFFF0113	Select card from identify mode to stand-by mode error
FMI_SD_INIT_ERROR	0xFFFF0115	SD Card initial and identify error
FMI_SD_CRC7_ERROR	0xFFFF0116	Command/Response error
FMI_SD_CRC16_ERROR	0xFFFF0117	Data reading error
FMI_SD_CRC_ERROR	0xFFFF0118	Data writing error
FMI_SD_CMD8_ERROR	0xFFFF0119	SD command 8 error
<b>Error Code for NAND</b>		
FMI_SM_INIT_ERROR	0xFFFF0120	NAND/SM card initial error
FMI_SM_RB_ERR	0xFFFF0121	NAND don't become ready from busy status
FMI_SM_STATE_ERROR	0xFFFF0122	NAND return fail for write command
FMI_SM_ECC_ERROR	0xFFFF0123	Read data error and uncorrectable by ECC
FMI_SM_STATUS_ERR	0xFFFF0124	NAND return fail for erase command
FMI_SM_ID_ERR	0xFFFF0125	NAND chip ID don't supported
FMI_SM_INVALID_BLOCK	0xFFFF0126	NAND block is invalid to erase or write
FMI_SM_MARK_BAD_BLOCK_ERR	0xFFFF0127	Fail to mark a block to bad
FMI_SM_REGION_PROTECT_ERR	0xFFFF0128	NAND return fail for write command because of region protect

# 24. SPI Library Description

This library provides APIs for programmers to access SPI device connecting with N9H26 SPI interfaces. The SPI library will get the APB clock frequency from system library, application must set the CPU clock before using SPI library.

## 24.1. API Functions

### *spiOpen*

#### Synopsis

```
INT32 spiOpen(SPI_INFO_T *pInfo)
```

#### Description

This function initialize the SPI interface.

#### Parameter

```
typedef struct _spi_info_t
{
    INT32 nPort;           /* select SPI0 (0) or SPI1 (1) */
    BOOL bIsSlaveMode;    /* set the interface mode - master mode or slave mode */
    BOOL bIsClockIdleHigh; /* set the clock idle state – high or low */
    BOOL bIsLSBFFirst;    /* set LSB transfer first or MSB first */
    BOOL bIsAutoSelect;   /* set automatically active / inactive CS pin */
    BOOL bIsActiveLow;    /* define the active level of device select signal */
    BOOL bIsTxNegative;  /* set the Tx signal changed on rising edge or
                           falling edge */
    BOOL bIsLevelTrigger; /* set the input slave select signal is edge-trigger
                           or level-trigger */
} SPI_INFO_T;
```

#### Return Value

= 0 Success

< 0 Fail

#### Example

```
spiOpen();
```

### **spiClose**

#### Synopsis

```
INT32 spiClose(UINT8 u8Port)
```

#### Description

This function disable SPI engine clock.

#### Parameter

u8Port      Select SPI0 (0) or SPI1 (1)

#### Return Value

= 0 Success

< 0 Fail

#### Example

```
spiClose (0);
```

### **spiloctl**

#### Synopsis

```
VOID spiIoctl(INT32 spiPort, INT32 spiFeature, INT32 spiArg0, INT32 spiArg1)
```

#### Description

This function allows programmers configure SPI interface.

#### Parameter

spiPort      Select SPI0 (0) or SPI1 (1)

spiFeature    SPI\_SET\_CLOCK

spiArg0      APB clock by MHz

spiArg1      Device output clock by kHz

#### Return Value

0      success

#### Example

```
/* apb clock is 48MHz, output clock is 10MHz */
```

```
spiIoctl(0, SPI_SET_CLOCK, 48, 10000);
```

## ***spiEnable***

### **Synopsis**

```
INT spiEnable(INT32 spiPort)
```

### **Description**

This function will active the SPI interface to access device (active CS#).

### **Parameter**

spiPort	Select SPI0 (0) or SPI1 (1)
---------	-----------------------------

### **Return Value**

0	success
---	---------

### **Example**

```
spiEnable(0);
```

## ***spiDisable***

### **Synopsis**

```
INT spiDisable(INT32 spiPort)
```

### **Description**

This function will inactive the SPI interface (inactive CS#).

### **Parameter**

spiPort	Select SPI0 (0) or SPI1 (1)
---------	-----------------------------

### **Return Value**

0	success
---	---------

### **Example**

```
spiDisable(0);
```

## ***spiRead***

### **Synopsis**

```
INT spiRead(INT port, INT RxBitLen, INT len, CHAR *pDst)
```

### **Description**

This function is used to read the data back from the SPI interface.

**Parameter**

port            select SPI0 (0) or SPI1 (1)  
 RxBitLen        set the receive bit length. *SPI\_8BIT, SPI\_16BIT, SPI\_32BIT*  
 len            data count. SPI\_8BIT is byte count; SPI\_16BIT is half-word count; SPI\_32BIT is word count.  
 pDst            Read back destination

**Return Value**

0            Success

**Example**

```
/* read 1 byte data from SPI device */

spiRead(0, SPI_8BIT, 1, (CHAR *)&rdata);
```

**spiWrite****Synopsis**

INT spiWrite(INT port, INT TxBitLen, INT len, CHAR \*pSrc)

**Description**

This function is used to write the data to the SPI interface.

**Parameter**

port            select SPI0 (0) or SPI1 (1)  
 TxBitLen        set the transmit bit length. *SPI\_8BIT, SPI\_16BIT, SPI\_32BIT*  
 len            data count. SPI\_8BIT is byte count; SPI\_16BIT is half-word count; SPI\_32BIT is word count  
 pSrc            data source address

**Return Value**

0            Success

**Example**

```
/* write 1 half-word to SPI device */

wdata = 0x80ff;

spiWrite(0, SPI_16BIT, 1, (CHAR *)&wdata);
```

**spiEnableInt****Synopsis**

VOID spiEnableInt(UINT8 u8Port)

#### Description

This function is used to enable the SPI interrupt.

#### Parameter

u8Port           Select SPI0 (0) or SPI1 (1)

#### Return Value

None

#### Example

```
spiEnableInt(0);
```

### ***spiDisableInt***

#### Synopsis

VOID spiDisableInt(UINT8 u8Port)

#### Description

This function is used to disable the SPI interrupt.

#### Parameter

u8Port           Select SPI0 (0) or SPI1 (1)

#### Return Value

None

#### Example

```
spiDisableInt(0);
```

### ***spiInstallCallBack***

#### Synopsis

ERRCODE spiInstallCallBack(UINT8 u8Port, PFN\_DRVSPI\_CALLBACK pfncallback,  
PFN\_DRVSPI\_CALLBACK \*pfnOldcallback)

#### Description

This function is used to install the specified SPI interrupt call back function.

#### Parameter

u8Port           Select SPI0 (0) or SPI1 (1)

pfncallback       The callback function pointer for specified SPI port.

pfnOldcallback      The previous callback function pointer for specified SPI port.

#### Return Value

0      Success

#### Example

```
spiInstallCallBack (0, SPIIRQHandler, &pfnOldcallback);
```

## **spiSetGo**

#### Synopsis

```
VOID spiSetGo(UINT8 u8Port)
```

#### Description

This function is used to set GO\_BUSY bit to trigger the SPI port.

#### Parameter

u8Port      Select SPI0 (0) or SPI1 (1)

#### Return Value

None

#### Example

```
spiSetGo(0);
```

## **spiSetByteEndin**

#### Synopsis

```
VOID spiSetByteEndin(UINT8 u8Port, E_DRVSPi_OPERATION eOP)
```

#### Description

This function is used to enable or disable the byte endin.

#### Parameter

u8Port      Select SPI0 (0) or SPI1 (1)

eOP      Select enable or disable the byte endin

#### Return Value

None

#### Example

```
spiSetByteEndin(0, eDRVSPi_DISABLE);;
```



# 25. SPU Library Description

This library provides APIs for programmers play PCM audio data from SPU engine. Except playing audio this library also provides 10-band equalizer APIs. SPU engine only plays audio, no record function is included.

## 25.1. API Functions

### *spuOpen*

#### Synopsis

```
VOID spuOpen(UINT32 u32SampleRate)
```

#### Description

This function will set the audio clock, play buffer address and install its interrupt.

#### Parameter

u32SampleRate Specific sampling rate

#### Return Value

None

#### Example

```
spuOpen();
```

### *spuStartPlay*

#### Synopsis

```
VOID spuStartPlay(PFN_DRVSPU_CB_FUNC *fnCallBack, UINT8 *data)
```

#### Description

After setting IO control to engine, this function will trigger SPU engine to start playing.

#### Parameter

fnCallBack Play call back function pointer

data      Source PCM audio data pointer

#### Return Value

None

#### Example

```
int playCallBack(UINT8 * pu8Buffer)
{
...
}
spuStartPlay((PFN_DRVSPU_CB_FUNC *) playCallBack, (UINT8 *)SPU_SOURCE);
```

## ***spuStopPlay***

#### Synopsis

VOID spuStopPlay (VOID)

#### Description

Stop play.

#### Parameter

None

#### Return Value

None

#### Example

spuStopPlay();

## ***spuClose***

#### Synopsis

VOID spuClose(VOID)

#### Description

This function disables SPU engine.

#### Parameter

None

#### Return Value

None

#### Example

```
spuClose();
```

### ***spuIoctl***

#### Synopsis

```
VOID spuIoctl(UINT32 cmd, UINT32 arg0, UINT32 arg1)
```

#### Description

This function allows programmers configure SPU engine, the supported command and arguments listed in the table below.

Command	Argument 0	Argument 1	Description
SPU_IOCTL_SET_VOLUME	Specifies left channel volume ranging from 0 (min.) to 0x3F (max.)	Specifies right channel volume ranging from 0 (min.) to 0x3F (max.)	Set SPU volume
SPU_IOCTL_SET_MONO	Not used	Not used	Set SPU to the mono mode
SPU_IOCTL_SET_STEREO	Not used	Not used	Set SPU to the stereo mode
SPU_IOCTL_GET_FRAG_SIZE	Fragment size	Not used	Get the fragment size from library

#### Parameter

- Cmd      Command
- arg0     First argument of the command
- arg1     Second argument of the command

#### Return Value

None

#### Example

```
spuIoctl(SPU_IOCTL_SET_VOLUME, 0x3f, 0x3f);
```

### ***spuDacOn***

#### Synopsis

```
VOID spuDacOn(UINT8 level)
```

#### Description

This function is used to enable DAC interface and must used before calling spuStartPlay().

**Parameter**

level delay time for de-pop noise

**Return Value**

None

**Example**

```
spuDacOn (1);
```

***spuDacOff*****Synopsis**

```
VOID spuDacOff(VOID)
```

**Description**

This function is used to disable DAC interface and must used after calling spuStopPlay().

**Parameter**

None

**Return Value**

None

**Example**

```
spuDacOff ();
```

***spuEqOpen*****Synopsis**

```
VOID spuEqOpen (E_DRVSPU_EQ_BAND eEqBand, E_DRVSPU_EQ_GAIN eEqGain)
```

**Description**

Open 10-band equalizer.

**Parameter**

eEqBand Equalizer band setting

eEqGain Equalizer gain setting for each band

**Return Value**

None

**Example**

```
spuEqOpen(eDRVSPU_EQBAND_2, eDRVSPU_EQGAIN_P7DB);
```

### ***spuEqClose***

#### **Synopsis**

```
VOID spuEqClose (VOID)
```

#### **Description**

Close Equalizer function.

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
spuEqClose ();
```

## **25.2. System Library Overview**

The System library provides a set of APIs to control on-chip functions such as Timers, UARTs, AIC, Cache and power management. With these APIs, user can quickly create a test program to run on demo board or evaluation board.

This library is created by using ARM Development Suite 1.2. Therefore, it only can be used in ADS environment.

## **25.3. Timer API Functions**

*Table 25-1 : Timer Channel*

Channel name	Value	Description
TIMER0	0	Timer 0
TIMER1	1	Timer 1
TIMER2	2	Timer 2
TIMER3	3	Timer 3
WDTIMER	4	Watch Dog Timer

*Table 25-2:Timer Mode*

<b>Timer Mode</b>	<b>Value</b>	<b>Description</b>
ONE_SHOT_MODE	0	One shot mode.
PERIODIC_MODE	1	Periodic mode.
TOGGLE_MODE	2	Toggle mode.
UNINTERRUPT_MODE	3	Uninterrupt mode.

*Table 25-3:Watch-Dog Interval Select*

<b>Interval Select</b>	<b>Value</b>	<b>Description</b>
WDT_14BITS	0	Timer 0
WDT_16BITS	1	Timer 1
WDT_18BITS	2	Timer 2
WDT_20BITS	3	Timer 3

## **sysSetTimerEvent**

### **Synopsis**

```
INT32 sysSetTimerEvent(UINT32 nTimeNo, UINT32 nTimeTick, PVOID pvFun);
```

### **Description**

This function is used to set the event of selected timer. *nTimeNo* is used to select timer 0 to timer 3. The event function which pointed by *pvFun* shall be executed after *nTimeTick* system timer tick. The function is useless for WDTIMER.

### **Parameter**

- |           |  |
|-----------|--|
| nTimeNo   | Timer 0 ~ timer 3. Please refer <a href="#">Table 25-1 : Timer Channel</a> . |
| nTimeTick | Tick count before event executed   |
| pvFun     | Event function pointer   |

### **Return Value**

Event number. Please remember the event number if you want to uninstall the timer event.

### **Example**

```
/* Set event function "hello" after 100 tick */

INT nEventNo;

VOID hello(VOID)
{
```

```
    sysPrintf("Hello World!\n");

}

nEventNo = sysSetTimerEvent (TIMER0, 100, (PVOID)hello);

....
```

sysClearTimerEvent (TIMER0, nEventNo);

## **sysClearTimerEvent**

### **Synopsis**

```
VOID sysClearTimerEvent(UINT32 nTimeNo, UINT32 uTimeEventNo);
```

### **Description**

This function is used to clear the event of selected timer. *nTimeNo* is used to select timer 0 ~ timer 3. The event function which indicated by *uTimeEventNo* shall be cleared. The function is useless for WDTIMER.

### **Parameter**

nTimeNo      TIMER0, TIMER1

uTimeEventNo   Event number which want to clear.

The event number is the return value of function-sysSetTimerEvent().

### **Return value**

None

### **Example**

```
/* clear event NO 5*/

sysClearTimerEvent (TIMER0, 5);
```

## **sysClearWatchDogTimerCount**

### **Synopsis**

```
VOID sysClearWatchDogTimerCount(VOID);
```

### **Description**

This function is used to clear watch dog timer reset counter. When interrupt occurred, the system will be reset after 1024 clock cycles. Clear the timer reset counter, the system will not be reset.

### **Parameter**

None

**Return value**

None

**Example**

```
sysClearWatchDogTimerCount();
```

**sysClearWatchDogTimerInterruptStatus****Synopsis**

```
VOID sysClearWatchDogTimerInterruptStatus(VOID);
```

**Description**

This function is used to clear watch dog timer interrupt status. When interrupt occurred, the watch dog timer interrupt flag will be set. Clear this flag, the interrupt will occur again.

**Parameter**

None

**Return value**

None

**Example**

```
sysClearWatchDogTimerInterruptStatus();
```

**sysDelay****Synopsis**

```
VOID sysDelay(UINT32 uTicks);
```

**Description**

This function is used to delay a specific period. *uTicks* is the length of delay time which unit is ten milliseconds. Please notice that the delay period has an extent of error which is less than ten milliseconds.

**Parameter**

uTicks delay period which unit is ten milliseconds

**Return value**

None

**Example**

```
/* delay 1s*/
```

```
sysDelay(100);
```

## ***sysDisableWatchDogTimer***

### **Synopsis**

```
VOID sysDisableWatchDogTimer(VOID);
```

### **Description**

This function is used to disable watch dog timer.

### **Parameter**

None

### **Return value**

None

### **Example**

```
sysDisableWatchDogTimer();
```

## ***sysDisableWatchDogTimerReset***

### **Synopsis**

```
VOID sysDisableWatchDogTimerReset(VOID);
```

### **Description**

This function is used to disable watch dog timer reset function.

### **Parameter**

None

### **Return value**

None

### **Example**

```
sysDisableWatchDogTimerReset();
```

## ***sysEnableWatchDogTimer***

### **Synopsis**

```
VOID sysEnableWatchDogTimer(VOID);
```

### **Description**

This function is used to enable watch dog timer.

**Parameter**

None

**Return value**

None

**Example**

```
sysEnableWatchDogTimer();
```

**sysEnableWatchDogTimerReset****Synopsis**

```
VOID sysEnableWatchDogTimerReset(VOID);
```

**Description**

This function is used to enable watch dog timer reset function. The system will be reset when this function is enabled.

**Parameter**

None

**Return value**

None

**Example**

```
sysEnableWatchDogTimerReset();
```

**sysGetCurrentTime****Synopsis**

```
VOID sysGetCurrentTime(DateTime_T *curTime);
```

**Description**

This function is used to get local time. *curTime* is a structure pointer which contains year, month, day, hour, minute, and second information.

**Parameter**

\**curTime*        structure pointer which contains the following information

typedef struct datetime\_t

{

```
    UINT32 year;  
    UINT32 mon;  
    UINT32 day;  
    UINT32 hour;  
    UINT32 min;  
    UINT32 sec;  
} DateTime_T;
```

**Return value**

None

**Example**

```
/* set local time*/  
  
DateTime_T     TimeInfo;  
  
sysGetCurrentTime(TimeInfo);
```

## **sysGetTicks**

**Synopsis**

```
UINT32 sysGetTicks(INT32 nTimeNo);
```

**Description**

This function gets the Timer 0 or Timer 1's current tick count.

**Parameter**

nTimeNo        TIMER0, TIMER1

**Return value**

The current selected timer tick count.

**Example**

```
/* Get current timer 0 tick count */  
  
UINT32 btime;  
  
btime = sysGetTicks(TIMER0);
```

## **sysInstallWatchDogTimerISR**

**Synopsis**

```
PVOID sysInstallWatchDogTimerISR(INT32 nIntTypeLevel, PVOID pvNewISR);
```

### Description

This function is used to set up own watch dog timer interrupt service routine. *nIntTypeLevel* is select interrupt to be FIQ or IRQ, and level group 0 ~ 7. *pvNewISR* is the own interrupt service routine's pointer.

### Parameter

*nIntTypeLevel* FIQ\_LEVEL\_0, IRQ\_LEVEL\_1 ~ IRQ\_LEVEL\_7  
*pvNewISR* the pointer of watch dog timer interrupt service routine

### Return value

a pointer which point to old ISR

### Example

```
/* Set watch dog timer interrupt to be IRQ and group level 1 */

PVOID oldVect;

oldVect = sysInstallWatchDogTimerISR(IRQ_LEVEL_1, myWatchDogISR);
```

## ***sysResetTicks***

### Synopsis

```
INT32 sysResetTicks(INT32 nTimeNo);
```

### Description

This function used to reset Timer 0 or Timer 1's global tick counter. The function is useless for WDTIMER.

### Parameter

*nTimeNo* TIMER0, TIMER1

### Return value

Successful

### Example

```
/* Reset timer 0 tick count */

INT32 status;

status = sysResetTicks(TIMER0);
```

## ***sysSetLocalTime***

### Synopsis

```
VOID sysSetLocalTime(DateTime_T ltime);
```

### Description

This function is used to set local time. *ltime* is a structure which contains year, month, day, hour, minute, and second information.

### Parameter

*ltime* structure which contains the following information

```
typedef struct datetime_t
{
    UINT32 year;
    UINT32 mon;
    UINT32 day;
    UINT32 hour;
    UINT32 min;
    UINT32 sec;
} DateTime_T;
```

### Return value

None

### Example

```
/* set local time*/

DateTime_T    TimeInfo;

TimeInfo.year = 2006;

TimeInfo.mon = 6;

TimeInfo.day = 12

TimeInfo.hour = 9;

TimeInfo.min = 0;

TimeInfo.sec = 30;

sysSetLocalTime(TimeInfo);
```

## ***sysSetTimerReferenceClock***

### Synopsis

```
INT32 sysSetTimerReferenceClock(UINT32 nTimeNo, UINT32 uClockRate);
```

**Description**

This function used to set the timer's reference clock. The default reference clock is system clock (15MHz). The function is useless for WDTIMER.

**Parameter**

nTimeNo	TIMER0, TIMER1
uClockRate	reference clock

**Return Value**

Successful

**Example**

```
/* Set 20MHz to be timer 0's reference clock */

INT32 status;

status = sysSetTimerReferenceClock(TIMER0, 20000000);
```

***sysSetWatchDogTimerInterval*****Synopsis**

```
INT32 sysSetWatchDogTimerInterval(INT32 nWdtInterval);
```

**Description**

This function is used to set the watch dog timer interval. The default is 0.5 minutes. You can select interval to be 0.5, 1, 2, and 4 minutes.

**Parameter**

nWdtInterval    WDT\_INTERVAL\_0, WDT\_INTERVAL\_1, WDT\_INTERVAL\_2,  
                  WDT\_INTERVAL\_3.

The watch dog timer interval is shown as follows bases on 12MHz.

nWdtInterval	Interrupt Timeout	Reset Timeout	Real Time Interval
WDT_INTERVAL_0	$2^{14}$ clocks	$2^{14} + 1024$ clocks	0.371 sec.
WDT_INTERVAL_1	$2^{16}$ clocks	$2^{16} + 1024$ clocks	1.419 sec.
WDT_INTERVAL_2	$2^{18}$ clocks	$2^{18} + 1024$ clocks	5.614 sec.
WDT_INTERVAL_3	$2^{20}$ clocks	$2^{20} + 1024$ clocks	22.391 sec.

**Return value**

Successful

**Example**

```
/* Set watch dog timer interval to WDT_INTERVAL_0 */

INT32 status;
```

```
status = sysSetWatchDogTimerInterval(WDT_INTERVAL_0);
```

## sysStartTimer

### Synopsis

```
INT32 sysStartTimer(INT32 nTimeNo, UINT32 uTicksPerSecond, INT32 nOpMode);
```

### Description

*nTimeNo* is used to select Timer 0, Timer 1, Timer 2, Timer3 or What-dog Timer. Because of the chip's timer has four operation modes, the *nOpMode* is used to set the operation mode. *uTicksPerSecond* indicates that how many ticks per second.

### Parameter

*nTimeNo*      TIMER0, TIMER1, TIMER2, TIMER3 or WDTIMER.

*nTickPerSecond* Tick number per second.

It is useless if WDTIMER

*nOpMode*      Working mode. Please refer the [Table 25-2:Timer Mode](#)

It is useless for WDTIMER

### Return Value

Successful

### Example

```
/* Start the timer 1, and set it to periodic mode and 100 ticks per second */

INT32 status;

status = sysStartTimer(TIMER1, 100, PERIODIC_MODE);
```

## sysStopTimer

### Synopsis

```
INT32 sysStopTimer(INT32 nTimeNo);
```

### Description

sysStopTimer will stop the specified timer channel. *nTimeNo* is used to select timer 0 ~ timer 3 or Watch Dog Timer. After disabling timer, this function will restore the interrupt service routine.

### Parameter

*nTimeNo*      TIMER0, TIMER1, TIMER2, TIMER3 or WDTIMER.

### Return Value

Successful

#### Example

```
/* Stop the timer 1 */
INT32 status;
status = sysStopTimer(TIMER1);
```

### ***sysUpdateTickCount***

#### Synopsis

```
INT32 sysUpdateTickCount(INT32 nTimeNo, UINT32 uCount);
```

#### Description

This function used to update Timer 0 or Timer 1's global tick counter.

#### Parameter

nTimeNo	TIMER0, TIMER1
uCount	tick counter value

#### Return Value

Successful

#### Example

```
/* update timer 0's tick counter as 3000 */
sysUpdateTickCount (TIMER0, 3000);
```

## 25.4. UART Function

*Table 25-4: UART Port*

Port Name	Value	Description
WB_UART_0	0	UART 0 – High Speed UART
WB_UART_1	1	UART 1 – Normal Speed UART

*Table 25-5: UART Data Bits*

Data Bits	Value	Description
WB_DATA_BITS_5	0	5 Data Bits
WB_DATA_BITS_6	1	6 Data Bits

WB_DATA_BITS_7	2	7 Data Bits
WB_DATA_BITS_8	3	8 Data Bits

*Table 25-6: UART Stop Bits*

Stop Bits	Value	Description
WB_STOP_BITS_1	0x0	1 Stop Bit
WB_STOP_BITS_2	0x4	2 Stop Bits

*Table 25-7: UART Parity Bits*

Parity Bits	Value	Description
WB_PARITY_NONE	0x0	Non Parity Bit
WB_PARITY_ODD	0x8	Odd Parity Bit
WB_PARITY_EVEN	0x18	Even Parity Bit

*Table 25-8: UART FIFO Threshold*

FIFO	Value	Description
LEVEL_1_BYTE	0x0	1 Byte FIFO
LEVEL_4_BYTES	0x1	4 Bytes FIFO
LEVEL_8_BYTES	0x2	8 Bytes FIFO
LEVEL_14_BYTES	0x3	14 Bytes FIFO
LEVEL_30_BYTE	0x4	30 Bytes FIFO (High Speed UART Only)
LEVEL_46_BYTES	0x5	46 Bytes FIFO (High Speed UART Only)
LEVEL_62_BYTES	0x6	62 Bytes FIFO (High Speed UART Only)

*Table 25-9: UART Interrupt type*

FIFO	Value	Description
UART_INT_RDA	0x0	UART Data Ready
UART_INT_RDTO	0x1	UART Time out
UART_INT_NONE	0xFF	Not to enable UART

## sysGetChar

### Synopsis

```
CHAR sysGetChar(VOID);
```

### Description

This function is user to obtain the next available character from the UART. Nothing is echoed. When no available characters are found, the function waits until a character from UART is found.

### Parameter

None

### Return Value

A character from UART

### Example

```
/* get user's input*/
CHAR cUserInput;
cUserInput = sysGetChar();
```

## sysInitializeUART

### Synopsis

```
INT32 sysInitializeUART(WB_UART *uart);
```

### Description

WB\_UART is the device initialization structure. The definition is as following:

```
typedef struct UART_INIT_STRUCT
{
    UINT32 uart_no;
    UINT32 freq;
    UINT32 baud_rate;
    UINT32 data_bits;
    UINT32 stop_bits;
    UINT32 parity;
    UINT32 rx_trigger_level;
} WB_UART;
```

*uart->uart\_no* is UART port to be initialized.

*uart->freq* is UART reference clock. Default is 15MHz. If user have different reference clock, used this parameter to change it.

*uart->baud\_rate* is used to set the COM port baud rate. The range is from 9600 to 230400.

The UART data bit can be 5, 6, 7, or 8. Use *uart->data\_bits* to set the suitable data bits.

The UART stop bit can be 1, or 2. Use *uart->stop\_bits* to set the suitable stop bits.

*uart->parity* is used to set the suitable parity check.

*uart->rx\_trigger\_level* is used to set the suitable trigger level.

### Parameter

*uart->uart\_no* WB\_UART\_0: High speed UART port.

WB\_UART\_1: Normal speed UART port.

*uart->data\_bits* WB\_DATA\_BITS\_5 ~ WB\_DATA\_BITS\_8

*uart->stop\_bits* WB\_STOP\_BITS\_1, WB\_STOP\_BITS\_2

*uart->parity* WB\_PARITY\_NONE, WB\_PARITY\_ODD, WB\_PARITY\_EVEN

*uart->rx\_trigger\_level* LEVEL\_1\_BYTE, LEVEL\_4\_BYTES, LEVEL\_8\_BYTES, LEVEL\_14\_BYTES are for normal/high speed UART. And LEVEL\_30\_BYTES, LEVEL\_46\_BYTES and LEVEL\_62\_BYTES are only for high speed UART. Normal speed UART means the baud rate less or equal to 115200 bps. And high speed UART means the baud rate up to 921600 bps.

### Return Value

Successful/ WB\_INVALID\_PARITY/ WB\_INVALID\_DATA\_BITS/  
WB\_INVALID\_STOP\_BITS/ WB\_INVALID\_BAUD

### Example

```
WB_UART_T uart;  
  
uart.uart_no = WB_UART_1 ;  
  
uart.uiFreq = APB_SYSTEM_CLOCK;  
  
uart.uiBaudrate = 115200;  
  
uart.uiDataBits = WB_DATA_BITS_8;  
  
uart.uiStopBits = WB_STOP_BITS_1;  
  
uart.uiParity = WB_PARITY_NONE;  
  
uart.uiRxTriggerLevel = LEVEL_1_BYTE;  
  
sysInitializeUART(&uart);    WB_UART_T uart;
```

## sysPrintf

### Synopsis

```
VOID sysPrintf(PCHAR pcStr, ...);
```

#### Description

The function sends the specified *str* to the terminal through the RS-232 interface by interrupt mode.

#### Parameter

pcStr pointer of string which want to display

#### Return Value

None

#### Example

```
sysPrintf("Hello World!\n");
```

## ***sysprintf***

#### Synopsis

```
VOID sysPrintf(PCHAR pcStr, ...);
```

#### Description

The function sends the specified *str* to the terminal through the RS-232 interface by polling mode.

#### Parameter

pcStr pointer of string which want to display

#### Return Value

None

#### Example

```
sysprintf("Hello World!\n");
```

## ***sysPutChar***

#### Name

sysPutChar – put a character out to UART

#### Synopsis

```
VOID sysPutChar(UCHAR ch);
```

#### Description

The function sends the specified *ch* to the UART.

**Parameter**

ch character which want to display

**Return Value**

None

**Example**

```
sysPutChar("A");
```

***sysUartInstallcallback*****Name**

sysUartInstallcallback – install callback function for high speed UART data ready event or data time out event processing.

**Synopsis**

```
void sysUartInstallcallback(UINT32 u32IntType,
                           PFN_SYS_UART_CALLBACK pfnCallback);
```

**Description**

The function is used to install the call back function for received data ready or received data time out. The call back function need following structure.

```
typedef void (*PFN_SYS_UART_CALLBACK)()
{
    UINT8* u8Buf,
    UINT32 u32Len);
}
```

u8Buf	Received data buffer pointer.
u32Len	Received data length.

**Parameter**

u32IntType interrupt type. Please refer [Table 25-9: UART Interrupt type](#)

pfnCallback a function pointer to process the received data ready and received data time out event.

**Return Value**

None

**Example**

```
sysUartInstallcallback(UART_INT_RDA, UartDataValid_Handler);
sysUartInstallcallback(UART_INT_RDTO, UartDataTimeOut_Handler);
```

## ***sysUartEnableInt***

### **Name**

sysUartEnableInt— enable UART interrupt type.

### **Synopsis**

```
VOID sysUartEnableInt(INT32 eIntType);
```

### **Description**

The function is used to enable UART interrupt.

### **Parameter**

eIntType      UART interrupt type. Please refer [Table 25-9: UART Interrupt type](#)

### **Return Value**

None

### **Example**

```
sysUartEnableInt(UART_INT_RDA);
sysUartEnableInt(UART_INT_RDTO);
.....
sysUartEnableInt(UART_INT_NONE);
```

## ***sysUartTransfer***

### **Name**

sysUartTransfer— Start up the UART transfer.

### **Synopsis**

```
VOID sysUartTransfer(char* pu8buf, UINT32 u32Len);
```

### **Description**

The function is used to transfer data.

### **Parameter**

pu8buf      Transfer data buffer pointer.  
u32Len      Transfer data length.

### **Return Value**

None

### **Example**

```
sysUartTransfer(pi8UartBuf, u32Count);
```

---

## 25.5. AIC Functions

*Table 25-10: Interrupt No.*

AIC Interrupt No	Value	Description
IRQ_WDT	1	Watch Dog Timer Interrupt
IRQ_EXTINT0	2	GPIO Group 0 interrupt
IRQ_EXTINT1	3	GPIO Group 1 interrupt
IRQ_EXTINT2	4	GPIO Group 2 interrupt
IRQ_EXTINT3	5	GPIO Group 3 interrupt
IRQ_IPSEC	6	AES Interrupt
IRQ_SPU	7	SPU Interrupt
IRQ_I2S	8	I2S Interrupt
IRQ_VPOST	9	VPOST Interrupt
IRQ_VIN	10	Video In 0 Interrupt
IRQ_MDCT	11	MDCT Interrupt
IRQ_BLT	12	BLT Interrupt
IRQ_VPE	13	VPE Interrupt
IRQ_HUART	14	High Speed UART Interrupt
IRQ_TMR0	15	Timer 0 Interrupt
IRQ_TMR1	16	Timer 1 Interrupt
IRQ_UDC	17	USB Device Controller Interrupt
IRQ_SIC	18	Storage Interrupt Controller Interrupt
IRQ_SDIO	19	Secure Digital Input / Output Control Interrupt
IRQ_UHC	20	USB Host Controller Interrupt
IRQ_EHCI	21	Enhanced Host Controller Interface Interrupt
IRQ_OHCI	22	Host Controller Interface Interrupt
IRQ_EDMA0	23	Enhanced DMA 0 Interrupt
IRQ_EDMA1	24	Enhanced DMA 1 Interrupt
IRQ_SPIMSO	25	SPI Master / Slave 0 Interrupt
IRQ_SPIMS1	26	SPI Master / Slave 1 Interrupt

IRQ_AUDIO	27	Audio Record Interrupt
IRQ_TOUCH	28	Touch Controller Interrupt
IRQ_RTC	29	RTC Interrupt
IRQ_UART	30	UART Interrupt
IRQ_PWM	31	PWM Interrupt
IRQ_JPG	32	JPEG Codec Interrupt
IRQ_VDE	33	H264 Decode Interrupt
IRQ_VEN	34	H264 Encode Interrupt
IRQ_SDIC	35	SDIC Interrupt
IRQ_EMCTX	36	EMC TX Interrupt
IRQ_EMCRX	37	EMC RX Interrupt
IRQ_I2C	38	I2C Interrupt
IRQ_KPI	39	Keypad Interrupt
IRQ_RSC	40	RS Codec Interrupt
IRQ_VTB	41	Convolution / Viterbi Codec Interrupt
IRQ_ROT	42	Convolution / Viterbi Codec Interrupt
IRQ_PWR	43	System Wake-Up Interrupt
IRQ_LVD	44	Low Voltage Detector Interrupt
IRQ_VIN1	45	Video In 1 Interrupt
IRQ_TMR2	46	Timer 2 Interrupt
IRQ_TMR3	47	Timer 3 Interrupt

*Table 25-11 : Interrupt Exception Type*

Exception Type	Value	Description
WB_SWI	0	Software Interrupt
WB_D_ABORT	1	Data Abort Interrupt
WB_I_ABORT	2	Instruction Abort Interrupt
WB_UNDEFINE	3	Undefined Interrupt

*Table 25-12: Interrupt Priority*

Interrupt Priority	Value	Description
FIQ_LEVEL_0	0	Highest Priority
IRQ_LEVEL_1	1	Level 1 Priority

IRQ_LEVEL_2	2	Level 2 Priority
IRQ_LEVEL_3	3	Level 3 Priority
IRQ_LEVEL_4	4	Level 4 Priority
IRQ_LEVEL_5	5	Level 5 Priority
IRQ_LEVEL_6	6	Level 6 Priority
IRQ_LEVEL_7	7	Lowest Priority

*Table 25-13: Local Interrupt Type*

Local Interrupt Type	Value	Description
ENABLE_IRQ	0x7F	Enable ARM Core's IRQ bit
ENABLE_FIQ	0xBF	Enable ARM Core's FIQ bit
ENABLE_FIQ_IRQ	0x3F	Enable ARM core's FIQ and IRQ bit
DISABLE_IRQ	0x80	Disable ARM Core's IRQ bit
DISABLE_FIQ	0x40	Disable ARM Core's FIQ bit
DISABLE_FIQ_IRQ	0xC0	Disable ARM core's FIQ and IRQ bit

*Table 25-14: Interrupt Trigger Type*

Interrupt Trigger Type	Value	Description
LOW_LEVEL_SENSITIVE	0x00	Low Level Trigger Type
HIGH_LEVEL_SENSITIVE	0x01	High Level Trigger Type
NEGATIVE_EDGE_TRIGGER	0x02	Falling Edge Trigger Type
POSITIVE_EDGE_TRIGGER	0x03	Rising Edge Trigger Type

***sysDisableInterrupt*****Name**

sysDisableInterrupt – disable interrupt source

**Synopsis**

INT32 sysDisableInterrupt(UINT32 intNo);

**Description**

This function is used to disable interrupt source.

**Parameter**

intNo interrupt source number. Please refer the [Table 25-10: Interrupt No.](#).

**Return Value**

Successful or Fail.

**Example**

```
/* Disable timer 0 interrupt (source number is 7) */

INT32 status;

status = sysDisableInterrupt(7);
```

***sysEnableInterrupt*****Synopsis**

```
INT32 sysEnableInterrupt(UINT32 intNo);
```

**Description**

This function is used to enable interrupt source. Please refer the [Table 25-10: Interrupt No.](#).

**Parameter**

intNo interrupt source number

**Return Value**

Successful or Fail.

**Example**

```
/* Enable timer 0 interrupt (source number is 7) */

INT32 status;

status = sysEnableInterrupt(7);
```

***sysGetIBitState*****Synopsis**

```
BOOL sysGetIBitState (VOID);
```

**Description**

This function is used to get the status of interrupt disable bit, I-bit, of CPSR register.

**Parameter**

None

**Return Value**

TRUE – I-bit is clear, FALSE – I-bit is set.

**Example**

```
BOOL int_status;  
  
Int_status = sysGetIBitState();
```

***sysGetInterruptEnableStatus*****Synopsis**

```
UINT32 sysGetInterruptEnableStatus(VOID);
```

**Description**

This function is used to get the enable/disable status of low channel interrupts which save in AIC\_IMR register.

**Parameter**

None

**Return Value**

value of AIC\_IMR register

**Example**

```
/* Set AIC as software mode */  
  
UINT32 uIMRValue;  
  
uIMRValue = sysGetInterruptEnableStatus();
```

***sysGetInterruptHighEnableStatus*****Synopsis**

```
UINT32 sysGetInterruptHighEnableStatus(VOID);
```

**Description**

This function is used to get the enable/disable status of high channel interrupts which save in AIC\_IMRH register.

**Parameter**

None

**Return Value**

value of AIC\_IMRH register

**Example**

```
/* Set AIC as software mode */

UINT32 uIMRValue;

uIMRValue = sysGetInterruptHighEnableStatus();
```

***sysInstallExceptionHandler*****Synopsis**

```
PVOID sysInstallExceptionHandler(INT32 exceptType, PVOID pNewHandler);
```

**Description**

This function is used to install *pNewHandler* into *exceptType* exception.

**Parameter**

exceptType      WB\_SWI, WB\_D\_ABORT, WB\_I\_ABORT, WB\_UNDEFINE

Please refer the [Table 25-11 : Interrupt Exception Type](#)

pNewHandler    pointer of the new handler

**Return Value**

a pointer which point to old handler

**Example**

```
/* Setup own software interrupt handler */

PVOID oldVect;

oldVect = sysInstallExceptionHandler(WB_SWI, pNewSWIHandler);
```

***sysInstallFiqHandler*****Synopsis**

```
PVOID sysInstallFiqHandler(PVOID pNewISR);
```

**Description**

Use this function to install FIQ handler into interrupt vector table.

**Parameter**

pNewISR      pointer of the new ISR handler

**Return Value**

a pointer which point to old ISR

**Example**

```
/* Setup own FIQ handler */

PVOID oldVect;

oldVect = sysInstallFiqHandler(pNewFiqISR);
```

## **sysInstallIrqHandler**

### **Synopsis**

```
PVOID sysInstallIrqHandler(PVOID pNewISR);
```

### **Description**

Use this function to install FIQ handler into interrupt vector table.

### **Parameter**

pNewISR      pointer of the new ISR handler

### **Return Value**

A pointer which point to old ISR

### **Example**

```
/* Setup own IRQ handler */

PVOID oldVect;

oldVect = sysInstallIrqHandler(pNewIrqISR);
```

## **sysInstallISR**

### **Synopsis**

```
PVOID sysInstallISR(INT32 intTypeLevel, INT_SOURCE_E intNo, PVOID pNewISR);
```

### **Description**

Interrupt priority level is 0 ~ 7. Level 0 is FIQ, and level 1 ~ 7 are IRQ. The highest priority is 0, and the lowest priority is 7. Use this function to set up interrupt source (*intNo*) *pNewISR* handler to AIC interrupt vector table.

### **Parameter**

intTypeLevel    FIQ\_LEVEL\_0, IRQ\_LEVEL\_1 ~ IRQ\_LEVEL\_7

Please refer the [Table 25-12: Interrupt Priority](#)

intNo          interrupt source number

Please refer the [Table 25-10: Interrupt No.](#)

pNewISR        Function pointer of new ISR.

**Return Value**

A function pointer which points to old ISR

**Example**

```
/* Setup timer 0 handler */

PVOID oldVect;

oldVect = sysInstallISR(IRQ_LEVEL_1, IRQ_TMR0, pTimerISR);
```

**sysSetAIC2SWMode****Synopsis**

```
INT32 sysSetAIC2SWMode(VOID);
```

**Description**

This function is used to set AIC as software mode. When the system AIC in software mode, the priority of each interrupt source shall be handled by software.

**Parameter**

```
intState ENABLE_IRQ, ENABLE_FIQ, ENABLE_FIQ_IRQ, DISABLE_IRQ,
DISABLE_FIQ, DISABLE_FIQ_IRQ
```

**Return Value**

Successful

**Example**

```
/* Set AIC as software mode */

sysSetAIC2SWMode();
```

**sysSetGlobalInterrupt****Synopsis**

```
INT32 sysSetGlobalInterrupt(INT32 intState);
```

**Description**

The function is used to enable or disable all interrupt sources.

**Parameter**

intState	ENABLE_ALL_INTERRUPTS or DISABLE_ALL_INTERRUPTS
----------	--

**Return Value**

Successful

#### Example

```
/* Disable all interrupt */

INT32 status;

status = sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
```

## ***sysSetInterruptPriorityLevel***

#### Synopsis

```
INT32 sysSetInterruptPriorityLevel(INT_SOURCE_E intNo, UINT32 intLevel);
```

#### Description

The interrupt has 8 group levels. The highest priority is 0, and the lowest priority is 7. Use this function can change the priority level after install ISR.

#### Parameter

intNo            interrupt source number

Please refer the [Table 25-10: Interrupt No.](#)

intLevel        Interrupt priority. Please refer [Table 25-12: Interrupt Priority](#).

#### Return Value

Successful or Fail.

#### Example

```
/* Change timer 0 priority to level 4 */

INT32 status;

status = sysSetInterruptPriorityLevel(7, 4);
```

## ***sysSetInterruptType***

#### Synopsis

```
INT32 sysSetInterruptType(INT_SOURCE_E intNo, UINT32 intSourceType);
```

#### Description

The interrupt has four kinds of interrupt source types. They are low level sensitive, high level sensitive, negative edge trigger, and positive edge trigger. The default is high level sensitive. This function is used to change the interrupt source type.

#### Parameter

intNo            interrupt source number

Please refer the [Table 25-10: Interrupt No.](#).

intSourceType    Interrupt trigger type.

Please refer the [Table 25-14: Interrupt Trigger Type](#)

#### Return Value

Successful or Fail.

#### Example

```
/* Change timer 0 source type to be positive edge trigger */

INT32 status;

status = sysSetInterruptType(IRQ_TMR0, POSITIVE_EDGE_TRIGGER);
```

## **sysSetLocalInterrupt**

#### Synopsis

```
INT32 sysSetLocalInterrupt(INT32 intState);
```

#### Description

The CPSR I bit and F bit need to be enabled or disabled, when using interrupt. This function is used to enable / disable I bit and F bit.

#### Parameter

intState        Enable or disable ARM core's F and I bit.

Please refer [Table 25-13: Local Interrupt Type](#)

#### Return Value

Successful

#### Example

```
/* Enable I bit of CPSR */

INT32 state;

state = sysSetLocalInterrupt(ENABLE_IRQ);
```

---

## 25.6. Cache Function

### ***sysDisableCache***

#### **Synopsis**

```
VOID sysDisableCache(VOID);
```

#### **Description**

This function is used to disable cache.

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
/* disabled cache */  
  
sysDisableCache();
```

### ***sysEnableCache***

#### **Synopsis**

```
VOID sysEnableCache(UINT32 uCacheOpMode);
```

#### **Description**

This function is used to enable cache.

#### **Parameter**

uCacheOpMode CACHE\_WRITE\_BACK, CACHE\_WRITE\_THROUGH

#### **Return Value**

None

#### **Example**

```
/* enable cache */  
  
sysEnableCache();
```

## **sysFlushCache**

### **Synopsis**

```
VOID sysFlushCache(INT32 cacheType);
```

### **Description**

This function is used to flush system cache. The parameter, cacheType is used to select cache which needs to be flushed.

### **Parameter**

cacheType      L\_CACHE, D\_CACHE, I\_D\_CACHE

### **Return Value**

None

### **Example**

```
/* flush cache */  
  
sysFlushCache(I_D_CACHE);
```

## **sysGetCacheState**

### **Synopsis**

```
VOID sysGetCacheState (VOID);
```

### **Description**

This function is used to get the enable/disable status of cache.

### **Parameter**

None

### **Return Value**

None

### **Example**

```
/* Read cache status */  
  
BOOL status;  
  
status = sysGetCacheState();
```

## **sysGetSdramSizebyMB**

### **Synopsis**

```
INT32 sysGetSdramSizebyMB(VOID);
```

**Description**

This function returns the size (in Mbytes) of total memory.

**Parameter**

None

**Return Value**

Memory size or Fail

**Example**

```
/* Get the memory size */

INT32 memsize;

memsize = sysGetSdramSizebyMB();

sysprintf("The total memory size is %dMbytes\n", memsize);
```

**sysInvalidCache****Synopsis**

```
VOID sysInvalidCache (VOID);
```

**Description**

This function is used to invalid both Instruction and Data cache contents.

**Parameter**

None

**Return Value**

None

**Example**

```
/* Invalid cache */

sysInvalidCache();
```

**sysSetCachePages****Synopsis**

```
INT32 sysSetCachePages(UINT32 addr, INT32 size, INT32 cache_mode);
```

**Description**

This function is used to change the cache mode of a memory area. Note that the starting address and the size must be 4Kbytes boundary.

#### Parameter

addr     The memory starting address.  
size     The memory size.  
cache\_mode     CACHE\_WRITE\_BACK / CACHE\_WRITE\_THROUGH /  
CACHE\_DISABLE.

#### Return Value

Successful or Fail

#### Example

```
/* enable cache to write-back mode */  
  
sysEnableCache(CACHE_WRITE_BACK);  
  
...  
  
sysFlushCache();  
  
/* Change the memory region 0x1000000 ~ 0x1001000 to be non-cacheable */  
  
sysSetCachePages(0x1000000, 4096, CACHE_DISABLE);
```

---

## 25.7. Clock Control function

### ***sysGetExternalClock***

#### Synopsis

```
UINT32 sysGetExternalClock(void);
```

#### Description

This function is used to get external clock setting. IBR only support 2 kinds of external clock frequency. 12MHz or 27MHz. So external clock will be 12MHz or 27MHz. The power on setting must meet the external clock.

#### Parameter

None

#### Return Value

External clock. Unit : Hz

**Example**

```
/* Read system clock setting */

UINT32 u32ExtFreq;

u32ExtFreq = sysGetExternalClock();
```

***sysSetSystemClock*****Synopsis**

```
UINT32 sysSetSystemClock(E_SYS_SRC_CLK eSrcClk,
                        UINT32 u32PllHz,
                        UINT32 u32SysHz);
```

**Description**

This function is used to write system clock setting includes PLL output frequency, System clock. The function gets the external clock automatically by power on setting.

**Parameter**

eSrcClk : System clock source.

It could be eSYS\_EXT, eSYS\_APLL and eSYS\_UPLL. They mean the system clock source come from external clock, APLL and UPLL respectively.

u32PllHz : Set the APLL or UPLL output frequency.

Unit : Hz.

u32SysHz : Set the system clock output frequency.

Unit : Hz. The system clock source can be external, APLL or UPLL.

There are some limitations in the clock function due to hardware's limitation.

1. These frqequence exist multiplication factor It means  $PLL \geq n * SYS$ , And HCLK clock is always equal to SYS clock/2.  
Where n is integer. And HCLK clock is SDR/DDR/DDR2 clock.
2. PLL clock must under or equal to 432MHz.
3. System clock must under or equal to the source clock.
4. HCLK clock depens on the layout and core power. Generally, it can up to 150MHz in core power 1.2V.

**Return Value**

Successful or Error code

**Example**

```
/* Write system clock setting */
```

```
sysSetSystemClock(eSYS_UPLL,           // system clock come from UPLL
                  288000000,        // UPLL = 288MHz
                  288000000)        // SYS = 288MHz
```

## **sysSetDramClock**

### **Synopsis**

```
UINT32 sysSetDramClock(E_SYS_SRC_CLK eSrcClk,
                       UINT32 u32PllHz,
                       UINT32 u32DdrHz);
```

### **Description**

This function is used to write memory clock setting includes PLL output frequency, memory clock. The function gets the external clock automatically by power on setting.

### **Parameter**

- eSrcClk : System clock source.  
It will be limited to eSYS\_MPLL.
- u32PllHz : MPLL output frequency.  
Unit : Hz.
- u32DdrHz : Set the memory clock output frequency.  
Unit : Hz. The system clock source can only be MPLL.

There are some limitations in the clock function due to hardware's limitation.

1. MCLK clock is equal to half of u32DdrHz. .
2. MCLK need great than HCLK1, HCLK2 and HCLK3.
3. Max MPLL output clock will be 360MHz.

### **Return Value**

Successful or Error code

### **Example**

```
/* Write memory clock setting */

sysSetDramClock(eSYS_MPLL,           // system clock come from MPLL
                  360000000,        // MPLL = 360MHz
                  360000000)        // DDR = 360MHz

sysSetSystemClock(eSYS_UPLL,          // system clock come from UPLL
```

```
288000000,      // UPLL = 288MHz  
288000000)      // SYS = 288MHz
```

## **sysSetCPUClock**

### **Synopsis**

```
UINT32 sysSetCPUClock(UINT32 u32CPUClock);
```

### **Description**

This function is used to set CPU clock.

### **Parameter**

u32CPUClock: CPU clock.

There are some limitations in the clock function due to hardware's limitation.

1. The CPU clock comes from SYS clock. It must less or equal to SYS clock.
2. The CPU divider only support even divider. It means CPU = SYS, SYS/2, SYS/4,... or SYS/16.
3. HCLK1 clock depends on CPU clock.
  - If CPU divider =1, HCLK1 = CPU/2.
  - If CPU divider !=1, HCLK1 = CPU.

### **Return Value**

Successful or Error code

### **Example**

```
/* Write system clock setting */  
  
sysSetSystemClock(eSYS_UPLL,           //E_SYS_SRC_CLK eSrcClk,  
                  192000000,          //UINT32 u32PllKHz,  
                  192000000);         //UINT32 u32SysKHz,  
  
sysSetCPUClock(192000000);
```

## **sysSetAPBClock**

### **Synopsis**

```
UINT32 sysSetAPBClock(UINT32 u32APBClock);
```

**Description**

This function is used to set APB clock.

**Parameter**

u32APBClock: APB clock.

There are some limitations in the clock function due to hardware's limitation.

1. The APB clock comes from HCLK1 clock. It must less or equal to HCLK1 clock.
2. Max APB divider is 8.

**Return Value**

Successful or Error code

**Example**

```
/* Write system clock setting */

sysSetSystemClock(eSYS_UPLL,           //E_SYS_SRC_CLK eSrcClk,
                  288000000,          //UINT32 u32PllKHz,
                  288000000);         //UINT32 u32SysKHz,
sysSetCPUClock(288000000);           //HCLK1 = 144MHz.
sysSetAPBClock(72000000);            //APB = 72MHz
```

***sysGetPLLOutputHz*****Synopsis**

```
UINT32 sysGetPLLOutputHz ( E_SYS_SRC_CLK eSysPll,
                           UINT32 u32FinHz);
```

**Description**

This function is used to read PLL output frequency.

**Parameter**

eSrcClk Specified PLL wants to know.

It could be eSYS\_APLL= 2 and eSYS\_UPLL = 3.

u32FinHz: External clock. Unit : Hz.

**Return Value**

Specified PLL output clock. Unit : Hz.

**Example**

```
u32ExtFreq = sysGetExternalClock();
```

```
u32PllOutHz = sysGetPLLOutputHz(eSYS_UPLL, u32ExtFreq);
```

## **sysGetSystemClock**

### **Synopsis**

```
UINT32 sysGetSystemClock(void);
```

### **Description**

This function is used to get system clock.

### **Parameter**

None

### **Return Value**

System clock. Unit: Hz.

### **Example**

```
/* Read system clock setting */

UINT32 u32SysFreq;

u32SysFreq = sysGetSystemClock();
```

## **sysGetDramClock**

### **Synopsis**

```
UINT32 sysGetDramClock(void);
```

### **Description**

This function is used to get DRAM clock.

### **Parameter**

None

### **Return Value**

DRAM clock. Unit: Hz.

### **Example**

```
/* Read DRAM clock setting */

UINT32 u32DramFreq;

u32DramFreq = sysGetDramClock();
```

## **sysGetCPUClock**

### **Synopsis**

```
UINT32 sysGetCPUClock(void);
```

### **Description**

This function is used to get CPU clock.

### **Parameter**

None

### **Return Value**

CPU clock. Unit: Hz.

### **Example**

```
/* Read CPU clock setting */

UINT32 u32CPUFreq;

u32CPUFreq = sysGetCPUClock();
```

## **sysGetHCLK1Clock**

### **Synopsis**

```
UINT32 sysGetHCLK1Clock(void);
```

### **Description**

This function is used to get HCLK1 clock.

### **Parameter**

None

### **Return Value**

HCLK1 clock. Unit: Hz.

### **Example**

```
/* Read HCLK1 clock setting */

UINT32 u32HCLK1Freq;

u32HCLK1Freq = sysGetHCLK1Clock();
```

## **sysGetAPBClock**

### **Synopsis**

```
UINT32 sysGetAPBClock(void);
```

### **Description**

This function is used to get APB clock.

### **Parameter**

None

### **Return Value**

APB clock. Unit: Hz.

### **Example**

```
/* Read HCLK1 clock setting */

UINT32 u32APBFreq;

u32APBFreq = sysGetAPBClock();
```

## **sysSetPllClock**

### **Synopsis**

```
UINT32 sysSetPllClock(E_SYS_SRC_CLK eSrcClk,
                      UINT32 u32TargetHz);
```

### **Description**

There are two PLL in the chip. User can assign one PLL as system clock source. The other one PLL can be assigned the output frequency through the function.

### **Parameter**

eSrcClk:        eSYS\_APLL = 2 or eSYS\_UPLL = 3.  
u32TargetHz:     Target PLL output frequency. Unit : Hz.

### **Return Value**

Specified PLL output frequency. Unit : Hz. The return value may not same as the specified value due to hardware's limitation. If not meet the hardware SPEC, library will auto to search the nearly frqeuncy.

### **Example**

```
/* Write system clock setting */

sysSetSystemClock(eSYS_UPLL,                    // system clock come from UPLL
```

```
    300000000,      // UPLL = 300MHz
    300000000);    // SYS = 300MHz

/*Specified APLL clock */

sysSetPllClock(eSYS_APLL,
    432000000);      // SYS = 432MHz
```

## ***sysCheckPllConstraint***

### **Synopsis**

```
UINT32 sysCheckPllConstraint (BOOL bIsCheck);
```

### **Description**

This function is used to enable or disable constraint checking for setting PLL clock.

### **Parameter**

None

### **Return Value**

None

### **Example**

```
/* Set PLL clock without constraint check */

sysCheckPllConstraint(FALSE);      //Disable constraint checking

sysSetSystemClock(eSYS_UPLL,        // system clock come from UPLL
    318000000,      // UPLL = 318MHz
    318000000);    // SYS = 318MHz

sysCheckPllConstraint(FALSE);      //Enable constraint checking for next time.
```

## ***sysSetSystemDivider***

### **Synopsis**

```
UINT32 sysSetSystemDivider (UINT32 u32Hclk,
                           UINT32 u32SysDiv);
```

### **Description**

This function is used to set system clock divider. It speeds up to slower down all of clocks after system clock for power consumption consider.

#### Parameter

- u32Hclk: HCLK clock. The clock is the system clock divide by 2.
- u32SysDiv: The system clock divider. The value should be from 0 ~ 7.

#### Return Value

Successful or Error code.

#### Example

```
sysSetSystemClock(eSYS_UPLL,           // system clock come from UPLL
                  288000000,        // UPLL = 288MHz
                  288000000);       // SYS = 288MHz

sysSetCPUClock(144000000)           // CPU = 144MHz. HCLK1 = 144MHz

sysSetAPBClock(72000000)            // APB = 72MHz

sysSetSystemDivider (288000000/4/2, 3); // System clock divider = (3+1) = 4.
                                         // HCLK = UPLL/8 = 36MHz.

                                         // CPU = 18MHz, APB = 9MHz
```

## 25.8. Power management Function

System can enter standby mode with DDR memory enter self refresh mode. The program code was kept in DDR memory and PLLs, CPU, system clock, AHB and APB were turned off. There are 10 wake-up channels to wake-up system if system in standby mode.

*Table 25-15: Wakeup Channels.*

Wake up channel	Value	Description
WE_EMAC	0x1	Wake up by magic packet
WE_UHC20	0x2	Wake up by device attached/deattached
WE_GPIO	0x100	Wake up by GPIO level change
WE_RTC	0x200	Wake up by RTC
WE_UART	0x800	Wake up by UART (RTS pin)
WE_UDC	0x1000	Wake up by attached/detached from USB host

WE_UHC	0x2000	Wake up by device attached/detached
WE_ADC	0x4000	Wake up by touch panel touch
WE_KPI	0x8000	Wake up by KPI pressing

**sysPowerDown****Synopsis**

```
INT sysPowerDown(UINT u32WakeUpSrc);
```

**Description**

This function was used to enter standby mode. The function also specified the wake-up channel to wake up system. Programmer need to disable the analog IPs such as TV DAC, ADC and LVD and so on before entry standby mode.

**Parameter**

u32WakeUpSrc Wakeup channels. Please reference [Table 25-15: Wakeup Channels](#).

**Return Value**

Successful

**Example**

```
sysPowerDown(WE_GPIO);
```

**25.9. Error Code Table**

Code Name	Value	Description
Successful	0	Successful
Fail	-1	Fail
WB_INVALID_PARITY	-1	Invalid parity
WB_INVALID_DATA_BITS	-2	Invalid data bits
WB_INVALID_STOP_BITS	-3	Invalid stop bits
WB_INVALID_BAUD	-4	Invalid baud rate
WB_PM_PD_IRQ_Fail	-1	Invalid power down IRQ
WB_PM_Type_Fail	-2	Invalid power manager type
WB_PM_INVALID_IRQ_NUM	-3	Invalid IRQ number
E_ERR_CLK	0xB0000001	Wrong clock setting

# 26. Touch ADC Library Overview

The N9H26 Touch ADC library provides a set of APIs to report the X and Y-axis coordinate, battery voltage and analog keypad. With these APIs, user can read the position that was touched in touch panel, get the current battery voltage and get the keypad scancode.

These libraries are created by using ARM Development Suite 1.2. Therefore, they only can be used in ADS environment.

## 26.1. Touch ADC Library APIs Specification

### *DrvADC\_Open*

#### **Synopsis**

```
INT32 DrvADC_Open (void);
```

#### **Description**

This function is used to open the Touch ADC library.

#### **Parameter**

None

#### **Return Value**

Successful

#### **Example**

```
/* Initialize Touch ADC library and enable IP clock*/
DrvADC_Open();
```

### *DrvADC\_Close*

#### **Synopsis**

```
INT32 DrvADC_Close(void)
```

#### **Description**

Close the ADC library.

#### Parameter

None

#### Return Value

Successful

#### Example

```
/* Close Touch ADC library*/
DrvADC_Close();
```

### ***DrvADC\_InstallCallback***

#### Synopsis

```
INT32 DrvADC_InstallCallback(E_ADC_INT_TYPE eIntType,
                               PFN_ADC_CALLBACK pfnCallback,
                               PFN_ADC_CALLBACK* pfnOldCallback);
```

#### Description

This function was used to install callback function that is used to notice the upper layer for event complete.

#### Parameter

eIntType      Interrupt event type.

pfnCallback    The callback function want to register

pfnOldCallback old callback function

Table 26-1: ADC read mode

Field name	Value	Description
eADC_KEY	0	(Unused in the driver)
eADC_TOUCH	1	(Unused in the driver)
eADC_AIN	2	Normal ADC conversion event done type.
eADC_POSITION	3	Converse position event done type
eADC_PRESSURE	4	Conversion pressure event done type

#### Return Value

-1 or Successful.

### Example

```
/* Read the x and y-axis coordinate */

static void Pressure_callback(UINT32 u32code)

{ /* The u32code[14:0] is parameterZ2. u32code[15] is Valid(1) or Invalid(0).

The u32code[30:16] is Z1 position. u32code[31] is Valid(1) or Invalid(0) */

    ...

}

static void Position_callback(UINT32 u32code)

{ /* The u32code[14:0] is Y position. u32code[15] is Valid(1) or Invalid(0).

The u32code[30:16] is X position. u32code[31] is Valid(1) or Invalid(0). */

    ...

}

DrvADC_Open();

DrvADC_InstallCallback(eADC_POSITION,

                      Position_callback,

                      &pfnOldCallback);

DrvADC_InstallCallback(eADC_PRESSURE,

                      Pressure_callback,

                      &pfnOldCallback);
```

## ***DrvADC\_PenDetection***

### Synopsis

```
INT32 DrvADC_PenDetection(BOOL bIs5Wire)
```

### Description

This function was used to read the touching position and pressure

### Parameter

bIs5Wire      5 wires or 4 wires touch panel.

1: 5 wires touch panel

0: 4 wires touch panel

**Return Value**

Error code

Name	Value	Description
E_ADC_BUSY	0xB800F001	Touch ADC is busy
E_TOUCH_UP	1	Pen state is up. (No touching event)
Successful	0	Pen state is down. Touch ADC is triggering.

**Example**

```
DrvADC_Open();

DrvADC_InstallCallback(eADC_POSITION,
    Position_callback,
    &pfnOldCallback);

DrvADC_InstallCallback(eADC_PRESSURE,
    Pressure_callback,
    &pfnOldCallback);

btime = sysGetTicks(TIMER0);

etime = btime;

while ((etime - btime) <= 300)

{

    while(TouchPanel_time==TRUE){

        TouchPanel_time = FALSE;

        do{

            ret = DrvADC_PenDetection(bIs5Wire);

            }while(ret != Successful);

    }

    etime = sysGetTicks(TIMER0);

}
```

```
sysClearTimerEvent(TIMER0, tmp);
```

## DrvADC\_KeyDetection

### Synopsis

```
INT32 DrvADC_KeyDetection(UINT32 u32Channel, UINT32* pu32KeyCode)
```

### Description

This function was used to read the scancode of keypad.

### Parameter

u32Channel Channel number. The value from 1 to 3.

pu32KeyCode Scancode.

### Return Value

Error code

Name	Value	Description
E_ADC_BUSY	0xB800F001	Touch ADC is busy
E_KEYPAD_UP	1	keypad state is up. (No keypad event)
Successful	0	keypad state is down.

### Example

```
DrvADC_Open();

btme = sysGetTicks(TIMER0);

etime = btime;

while ((etime - btime) <= 300){

    if(DrvADC_KeyDetection(u32Channel, &u32KeyCode)==Successful){ // ready

        if(u32KeyCode!=0)

            sysprintf("Key Scan code = 0x%x\n", u32KeyCode);

    }else

        if(u32KeyCode!=0)

            sysprintf("Key Scan code = 0x%x\n", u32KeyCode);

    etime = sysGetTicks(TIMER0);

}
```

```
sysClearTimerEvent(TIMER0, tmp);
```

## DrvADC\_VoltageDetection

### Synopsis

```
INT32 DrvADC_VoltageDetection (UINT32 u32Channel)
```

### Description

This function was used to read the conversion value.

### Parameter

u32Channel      Channel number. The value from 1 to 3.

### Return Value

Error code

Name	Value	Description
E_ADC_BUSY	0xB800F001	Touch ADC is busy
Successful	0	Touch ADC is triggering.

### Example

```
static void VoltageDetect_callback(UINT32 u32code)

{ /* u32code is the ADC value */

    bIsValidVoltageDet = TRUE;

    u32VoltageValue = u32code;

    u32Count = u32Count+1;

    if(u32code==0){

        sysprintf("Voltage Detect Value %d = %d\n", u32Count, u32code);

    }

    sysprintf("Voltage Detect Value %d = %d\n", u32Count, u32code);

}

DrvADC_Open();

DrvADC_InstallCallback(eADC_AIN,

                      VoltageDetect_callback,

                      &pfnOldCallback);
```

```
        btime = sysGetTicks(TIMER0);

        etime = btime;

while ((etime - btime) <= 1000)

{

    while(VoltageDetect_time==TRUE){

        VoltageDetect_time = FALSE;

        if(DrvADC_VoltageDetection(u32Channel)==Successful){//ready

            if(bIsValidVoltageDet==TRUE){ //Key code has been updated

                bIsValidVoltageDet = FALSE;

            }

        }

    }

    etime = sysGetTicks(TIMER0);

}

sysClearTimerEvent(TIMER0, tmp);
```

---

## 26.2. Error Code Table

Code Name	Value	Description
E_ADC_BUSY	0xB800F000	Touch ADC is busy
E_TOUCH_UP	1	Pen state is up
E_KEYPAD_UP	1	Keypad state is up
Successful	0	No error

# 27. UDC Library

This library is designed to make user application to use N9H26 UDC more easily.  
The UDC library has the following features:

- Support all Basic USB operations.
- Pass USB-IF Chapter 9.

SDK Non-OS provide two usb class libraries for the USB class reference sample. User can refer to the libraries to develop him own class libraries. Mass Storage Class device: mscd library.

- - Pass the USB-IF Mass Storage Class Test
  - Provide flash options to build MSC device as a Composite device with RAM disk, NAND Disk, and SD Card Reader.
- USB Video Class device : uvcd library.
  - Pass the USB-IF Video Class Test
  - Provide a video cam sample to send two test patterns to PC.

User can use UDC library to implement all USB basic operations (Send descriptors, Reset command and etc.), and a USB class library (like MSCD) to provide USB class functions.

MSC Device	UVC Device	Other Devices
MSC Library	UVC Library	Other Libraries
UDC Library		

## 27.1. Programming Guide

### System Overview

The USB device controller interfaces the AHB bus and the UTMI bus. The USB controller contains both the AHB master interface and AHB slave interface. CPU programs the USB controller registers through the AHB slave interface. For IN or OUT transfer, the USB device controller needs to write data to memory or read data from memory through the AHB master interface. The USB device controller is complaint with USB 2.0 specification and it contains four configurable endpoints in addition to control endpoint. These endpoints could be configured to BULK, INTERRUPT or ISO. The USB device controller has a built-in DMA to relieve the load of CPU.

## Features

- USB Specification version 2.0 compliant.
- Interfaces between USB 2.0 bus and the AHB bus.
- Supports 16-bit UTMI Interface to USB2.0 Transceiver.
- Support direct register addressing for all registers from the AHB bus.
- Software control for device remote-wakeup.
- AHB bus facilitates connection to common micro controllers and DMA controllers.
- Supports 4 configurable endpoints in addition to Control Endpoint
- Each of these endpoints can be Isochronous, Bulk or Interrupt and they can be either of IN or OUT direction.
- Three different modes of operation of an in-endpoint (Auto validation mode, manual validation mode, Fly mode.)
- DP RAM is used as end point buffer.
- DMA operation is carried out by AHB master
- Supports Endpoint Maximum Packet Size up to 1024 bytes.

## ***UDC Library Property Definition***

The UDC library provides property structure to set UDC property more easily.

USBD\_INFO\_T (The fields for internal used are not in the table)

Name	Description
<b>Descriptor pointer</b>	
<i>pu32DevDescriptor</i>	Device Descriptor pointer
<i>pu32QulDescriptor</i>	Device Qualifier Descriptor pointer
<i>pu32HSConfDescriptor</i>	Standard Configuration Descriptor pointer for High speed
<i>pu32FSConfDescriptor</i>	Standard Configuration Descriptor pointer for Full speed
<i>pu32HOSConfDescriptor</i>	Other Speed Configuration Descriptor pointer for High speed
<i>pu32FOSConfDescriptor</i>	Other Speed Configuration Descriptor pointer for Full speed
<i>pu32StringDescriptor[5]</i>	String Descriptor pointer
<b>Descriptor length</b>	
<i>u32DevDescriptorLen</i>	Device Descriptor Length
<i>u32QulDescriptorLen</i>	Device Qualifier Descriptor pointer Length
<i>u32HSConfDescriptorLen</i>	Standard Configuration Descriptor Length for High speed
<i>u32FSConfDescriptorLen</i>	Standard Configuration Descriptor Length for Full speed
<i>u32HOSConfDescriptorLen</i>	Other Speed Configuration Descriptor Length for High speed
<i>u32FOSConfDescriptorLen</i>	Other Speed Configuration Descriptor Length for Full speed
<i>u32StringDescriptorLen[5]</i>	String Descriptor Length
<b>USBD Init</b>	

<i>pfnHighSpeedInit</i>	High speed USB Device Initialization function
<i>pfnFullSpeedInit</i>	Full speed USB Device Initialization function
<b>Endpoint Number</b>	
<i>i32EPA_Num</i>	Endpoint Number for EPA (-1 : Not used)
<i>i32EPB_Num</i>	Endpoint Number for EPB (-1 : Not used)
<i>i32EPC_Num</i>	Endpoint Number for EPC (-1 : Not used)
<i>i32EPD_Num</i>	Endpoint Number for EPD (-1 : Not used)
<b>Endpoint Call Back</b>	
<i>pfnEPACallBack</i>	Callback function pointer for Endpoint A Interrupt
<i>pfnEPBCallBack</i>	Callback function pointer for Endpoint B Interrupt
<i>pfnEPCCallBack</i>	Callback function pointer for Endpoint C Interrupt
<i>pfnEPDCallBack</i>	Callback function pointer for Endpoint D Interrupt
<b>Class Call Back</b>	
<i>pfnClassDataINCallBack</i>	Callback function pointer for Class Data IN
<i>pfnClassDataOUTCallBack</i>	Callback function pointer for Class Data OUT
<i>pfnDMACompletion</i>	Callback function pointer for DMA Complete
<i>pfnReset</i>	Callback function pointer for USB Reset Interrupt
<i>pfnSOF</i>	Callback function pointer for USB SOF Interrupt
<i>pfnPlug</i>	Callback function pointer for USB Plug Interrupt
<i>pfnUnplug</i>	Callback function pointer for USB Un-Plug Interrupt
<b>VBus status</b>	
<i>u32VbusStatus</i>	VBus Status

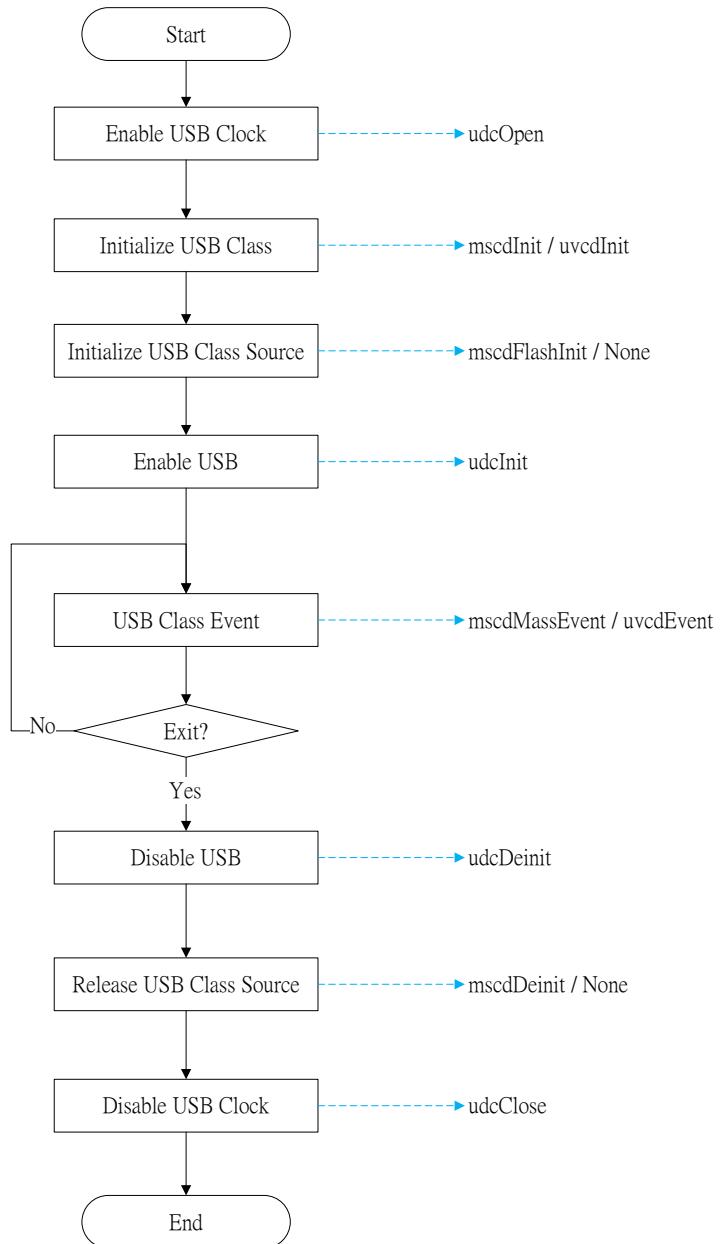
The USB Device initial function initializes the basic setting of USB device controller including endpoints buffer allocate, endpoint number, endpoint type, speed mode, and interrupt, etc. User can modify the function to change USB speed and endpoint properties.

- *pfnHighSpeedInit*
  - *mscdHighSpeedInit*
  - *uvcdHighSpeedInit*
- *pfnFullSpeedInit*
  - *mscdFullSpeedInit*
  - *uvcdFullSpeedInit*

PC classifies USB device according to the descriptors. With Non-OS SDK structure, the descriptors are initialized in the class Init functions. The functions set proper descriptors and the callback functions.

- *mscdInit*
- *uvcdInit*

## *Programming Flow*



---

## 27.2. USB Device (UDC) API

### *udcOpen*

#### Synopsis

```
VOID udcOpen(VOID)
```

#### Description

This function enables the engine clock.

#### Parameter

None

#### Return Value

None

#### Example

```
udcOpen();
```

### *udcClose*

#### Synopsis

```
VOID udcClose (VOID)
```

#### Description

This function disables the engine clock.

#### Parameter

None

#### Return Value

None

#### Example

```
udcClose();
```

### *udcInit*

#### Synopsis

```
VOID udcInit(VOID)
```

**Description**

This function initializes the software resource, enables its interrupt, and set VBus detect function.

**Parameter**

None

**Return Value**

None

**Example**

```
udcInit();
```

***udcDeinit*****Synopsis**

```
VOID udcDeinit(VOID)
```

**Description**

Disable VBus detect function

**Parameter**

None

**Return Value**

None

**Example**

```
udcDeinit();
```

***udcIsAttached*****Synopsis**

```
BOOL udcIsAttached(VOID)
```

**Description**

This function can get USB attach status.

**Parameter**

None

**Return Value**

TRUE	- USB is attached.
------	--------------------

FALSE - USB isn't attached.

#### Example

```
/* Check USB attach status */

if(udcIsAttached ())

    sysprintf("USB is attached\n");

else

    sysprintf("USB isn't attached\n");
```

### ***udcIsAttachedToHost***

#### Synopsis

BOOL udcIsAttachedToHost (VOID)

#### Description

This function can get USB current attach device status.

#### Parameter

None

#### Return Value

TRUE - USB is attached to Host now.

FALSE - USB doesn't get any command from Host now.

#### Example

```
/* Check USB HOST attach status */

if(udcIsAttachedToHost ())

    sysprintf("USB is attached to Host now\n");

else

    sysprintf("USB doesn't get any command from Host \n");
```

#### Note

It takes time for Host to sent command to device. So user may set a timeout time to check the status, i.e., user needs to polling the status during the timeout time.

---

## 27.3. Mass Storage Class (MSCD) API

### ***mscdInit***

#### **Synopsis**

```
VOID mscdInit(VOID)
```

#### **Description**

This function initializes software source (descriptors, callback functions, buffer configuration)

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
mscdInit();
```

### ***mscdDeinit***

#### **Synopsis**

```
VOID mscdDeinit (VOID)
```

#### **Description**

This function release software source (allocated by mscdInit)

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

```
mscdDeinit();
```

### ***mscdFlashInit***

#### **Synopsis**

```
UINT8 mscdFlashInit (NDISK_T *pDisk, INT SDsector);
```

**Description**

Initial the Flash capacity for usb device controller use.(One chip selector NAND flash and one port SD)

**Parameter**

pDisk            The internal data for NAND disk information.  
SDsector        Total sector for SD disk.

**Return Value**

0                - Fail  
1                - Success

**Example**

```
NDISK_T MassNDisk;  
  
INT SDsector;  
  
SDsector = sicSdOpen0();  
  
mscdFlashInit(&MassNDisk, SDsector);
```

**Note**

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable.(Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by **mscdSdEnable**. (Default is Port0 if user doesn't use mscdSdEnable)
3. The API can only single port SD and single CS NAND by mscdSdEnable or mscdNandEnable.
4. If user wants to export only SD, please link N9H26\_MSC\_SD.a
5. If user wants to export only NAND, please link N9H26\_MSC\_NAND.a
6. If user wants to export both SD and NAND, please link N9H26\_MSC\_NAND\_SD.a

***mscdFlashInitNAND*****Synopsis**

```
UINT8  
  
mscdFlashInitNAND (   
    NDISK_T *pDisk,  
    NDISK_T *pDisk1,  
    NDISK_T *pDisk2,  
    INT SDsector
```

);

**Description**

Initial the Flash capacity for usb device controller use (thress chip selector NAND flash and one port SD) .

**Parameter**

pDisk	The internal data for NAND disk information for CS0.
pDisk1	The internal data for NAND disk information for CS1.
pDisk2	The internal data for NAND disk information for CS2.
SDsector	Total sector for SD disk.

**Return Value**

0	- Fail
1	- Success

**Example**

```
NDISK_T MassNDisk, MassNDisk1, MassNDisk2;

INT SDsector;

SDsector = sicSdOpen0();

mscdFlashInitNAND (&MassNDisk, &MassNDisk1, &MassNDisk2, SDsector);
```

**Note**

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable.(Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. The API can only single port SD by mscdSdEnable.
4. If user wants to export only SD, please link N9H26\_MSC\_SD.a
5. If user wants to export only NAND, please link N9H26\_MSC\_NAND.a
6. If user wants to export both SD and NAND, please link N9H26\_MSC\_NAND\_SD.a

***mscdFlashInitExtend*****Synopsis**

```
UINT8
mscdFlashInitExtend (
    NDISK_T *pDisk,
    NDISK_T *pDisk1,
```

```

NDISK_T *pDisk2,
INT SDsector0,
INT SDsector1,
INT SDsector2,
INT RamSize
);

```

**Description**

Initial the Flash capacity for usb device controller use (thress chip selector NAND flash and three ports SD).

**Parameter**

pDisk	The internal data for NAND disk information for CS0.
pDisk1	The internal data for NAND disk information for CS1.
pDisk2	The internal data for NAND disk information for CS2.
SDsector0	Total sector for SD0 disk.
SDsector1	Total sector for SD1 disk.
SDsector2	Total sector for SD2 disk.
RamSize	MSC_RAMDISK_1M~ MSC_RAMDISK_64M

**Return Value**

0	- Fail
1	- Success

**Example**

```

NDISK_T MassNDisk, MassNDisk1, MassNDisk2;

INT SDsector;

SDsector = sicSdOpen0();

mscdFlashInitNAND (&MassNDisk, &MassNDisk1, &MassNDisk2, SDsector);

```

**Note**

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable.(Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. If user wants to export only SD, please link N9H26\_MSC\_SD.a
4. If user wants to export only NAND, please link N9H26\_MSC\_NAND.a
5. If user wants to export both SD and NAND, please link N9H26\_MSC\_NAND\_SD.a

6. If user wants to export only all flash, please link N9H26\_MSC\_All.a

### ***mscdSdEnable***

#### **Synopsis**

```
VOID mscdSdEnable (UINT32 u32Enable)
```

#### **Description**

This function enables the SD port for MSC.

#### **Parameter**

u32Enable	MSC_SD_MP_PORT0~ MSC_SD_MP_PORT2
	MSC_SD_PORT0~ MSC_SD_PORT2

#### **Return Value**

None

#### **Example**

```
/* Export two SD ports (Multiple partition) */

mscdSdEnable (MSC_SD_MP_PORT0| MSC_SD_MP_PORT1);

/* Export one SD port (Single partition) */

mscdSdEnable (MSC_SD_ PORT0);
```

### ***mscdNandEnable***

#### **Synopsis**

```
VOID mscdNandEnable (UINT32 u32Enable)
```

#### **Description**

This function enables the NAND CS for MSC.

#### **Parameter**

u32Enable	MSC_NAND_CS0~ MSC_NAND_CS2
-----------	----------------------------

#### **Return Value**

None

#### **Example**

```
/* Export two NAND CS */

mscdSdEnable (MSC_NAND_CS0| MSC_NAND_CS2);
```

```
/* Export one NAND CS */  
  
mscdSdEnable (MSC_NAND_CS0);
```

### ***mscdSdUserWriteProtectPin***

#### **Synopsis**

```
VOID mscdSdUserWriteProtectPin (  
    UINT32      u32SdPort,  
    BOOL        bEnable,  
    UINT32      u32GpioPort,  
    UINT32      u32GpioPin  
)
```

#### **Description**

This function enables/disables the SD write protect function and SD write protect pin for MSC.

#### **Parameter**

u32SdPort	MSC_SD_PORT0~ MSC_SD_PORT2
bEnable	Enable or Disable Write Protection function (TRUE/FALSE)
u32GpioPort	MSC_SD_GPIO_PORTA~ MSC_SD_GPIO_PORTE, MSC_SD_GPIO_PORTG, MSC_SD_GPIO_PORTH
u32GpioPin	GPIO pin number : 0~15

#### **Return Value**

None

#### **Example**

```
/* Set GPIOA Pin 2 for SD port0 Write Protect Pin */  
  
mscdSdUserWriteProtectPin (MSC_SD_PORT0, TRUE, MSC_SD_GPIO_PORTA, 2);  
  
/* Disable SD Port 0 Write Protect Pin function*/  
  
mscdSdUserWriteProtectPin (MSC_SD_PORT0, FALSE, 0, 0);
```

#### **Note**

Only SD Port0 has default Write Protection pin and Write Protection function is default enable and use the default pin (GPA0).

***mscdSdUserCardDetectPin*****Synopsis**

```
VOID mscdSdUserCardDetectPin (
    UINT32      u32SdPort,
    BOOL        bEnable,
    UINT32      u32GpioPort,
    UINT32      u32GpioPin
)
```

**Description**

This function enables/disables the SD card detection funvyion for MSC.

**Parameter**

u32SdPort	MSC_SD_PORT0~ MSC_SD_PORT2
bEnable	Enable or Disable card detection (TRUE/FALSE)
u32GpioPort	MSC_SD_GPIO_PORTA~ MSC_SD_GPIO_PORTE, MSC_SD_GPIO_PORTG, MSC_SD_GPIO_PORTH
u32GpioPin	GPIO pin number : 0~15

**Return Value**

None

**Example**

```
/* Set GPIOA Pin 2 for SD port0 Card detect Pin */

mscdSdUserCardDetectPin (MSC_SD_PORT0, TRUE, MSC_SD_GPIO_PORTA, 2);

/* Disable SD Port 0 Card detect function*/

mscdSdUserCardDetectPin (MSC_SD_PORT0, FALSE, 0, 0);
```

**Note**

1. Only SD Port0/2 has default Card detect pin and Card detect function is default enable and use the default pin (GPA1 for Port 0 and GPE11for Port2).
2. If user disable the Card detect function, MSC will consider that the SD card is always exist.

***mscdMassEvent*****Synopsis**

```
VOID mscdMassEvent (PFN_USBD_EXIT_CALLBACK* callback_fun)
```

**Description**

This function processes all the mass storage class commands such as read, write, inquiry, etc. The function has loop in it and it exits the loop according to the return value of the callback function.

**Parameter**

callback\_fun                  The callback function for the Mass Event Exit condition. If it returns FALSE, the mass event service is disabled.

**Return Value**

None

**Example**

```
mscdMassEvent(udcIsAttached);
```

**Note**

The API must be called when all APIs about MSC is completed.

---

## 27.4. USB Video Class (UVCD) API

### *uvcdInit*

**Synopsis**

```
VOID uvcdInit(PFN_UVCD_PUCONTROL_CALLBACK* callback_func)
```

**Description**

This function initializes software source and install the Process Unit Callback function.

**Parameter**

callback\_func                  Process Uint Call back function pointer

**Return Value**

None

**Example**

```
/* Initial UVC and install Process Uint Call back function */

uvcdInit(ProcessUnitControl);

/* Process Uint Call back function */
```

```
UINT32 ProcessUnitControl(UINT32 u32ItemSelect,UINT32 u32Value)

{

    switch(u32ItemSelect)

    {

        case PU_BACKLIGHT_COMPENSATION_CONTROL:

            sysprintf("Set Backlight -> %d\n",u32Value);

            break;

        case PU_BRIGHTNESS_CONTROL:

            sysprintf("Set Brightness -> %d\n",u32Value);

            break;

        case PU_CONTRAST_CONTROL:

            sysprintf("Set Contrast -> %d\n",u32Value);

            break;

        case PU_HUE_CONTROL:

            sysprintf("Set Hue -> %d\n",u32Value);

            break;

        case PU_SATURATION_CONTROL:

            sysprintf("Set Saturation -> %d\n",u32Value);

            break;

        case PU_SHARPNESS_CONTROL:

            sysprintf("Set Sharpness -> %d\n",u32Value);

            break;

        case PU_GAMMA_CONTROL:

            sysprintf("Set Gamma -> %d\n",u32Value);

            break;

        case PU_POWER_LINE_FREQUENCY_CONTROL:

            sysprintf("Set Power Line Frequency -> %d\n",u32Value);

            break;
    }
}
```

```
    }  
  
    return 0;  
  
}
```

## ***uvcdSendImage***

### **Synopsis**

```
BOOL uvcdSendImage(UINT32 u32Addr, UINT32 u32transferSize, BOOL bStillImage)
```

### **Description**

This function is to send preview or snapshot image to USB Host.

### **Parameter**

u32Addr	Image data address
u32transferSize	Image data size
bStillImage	TRUE (Snapshot) / FALSE (Preview)

### **Return Value**

None

### **Example**

```
/* Send Image */  
  
uvcdSendImage(u32Addr, u32transferSize, uvcStatus.StillImage);
```

## ***uvcdIsReady***

### **Synopsis**

```
BOOL uvcdIsReady(VOID)
```

### **Description**

This function is to check UVC is ready to send image or not.

### **Parameter**

None

### **Return Value**

TRUE	Ready
FALSE	Busy

### **Example**

```
/* Wait for Complete */  
while(!uvcdIsReady());
```

---

## 27.5. Example code

This demo code has sample code for MSC (Mass Storage Class) and UVC (USB Video Class)  
Please refer to the mass\_storage & video\_class sample codes of SDK Non-OS.

# 28. USB Core Library Overview

## 28.1. USB Core Library Overview

The USB Core library is composed of four major parts, which are OHCI driver, EHCI driver, USB driver, and USB hub device driver. Each of these four drivers also represents one of the three-layered USB driver layers. Figure 1-1 presents the driver layers of the USB library.

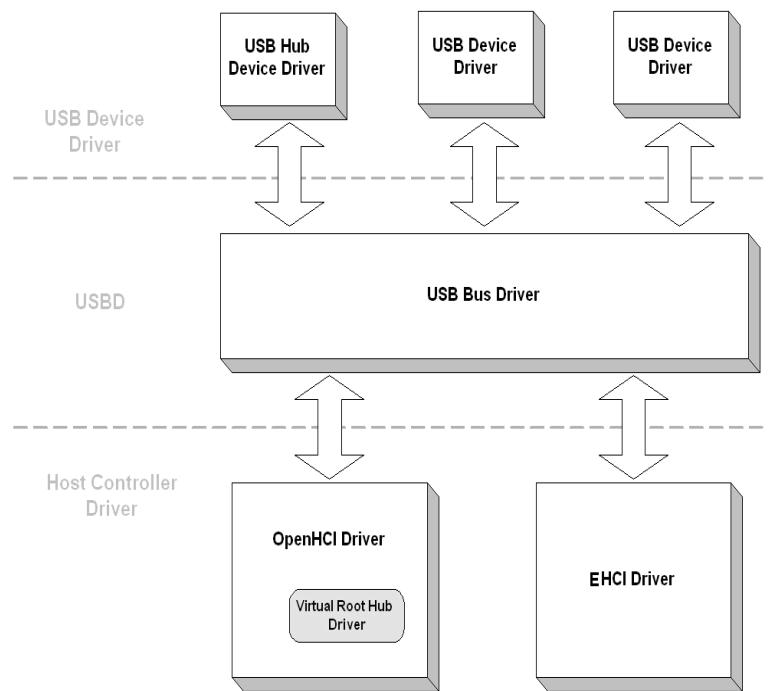


Figure 28-1 USB driver layer of USB library

## 28.2. Data Structures

The USB Core library has included many complicated data structures to describe a USB bus, a device, a driver, various descriptors, and so on. To realize these data structures may be necessary for a USB device driver designer. In the following sections, we will introduce all data structures you may need. These data structures are all defined in header file <usb.h>.

### ***USB\_DEV\_T***

***USB\_DEV\_T*** is the data structure used to represent a device instance. Once the host finds that a device presented on a USB bus, the USB system software is notified. The USB system software resets and enables the hub port to reset the device. It then creates a ***USB\_DEV\_T*** for the newly detected device. For each USB device presented on the bus, even the same device type, USB system software will create a ***USB\_DEV\_T*** to represent it as an instance.

The contents of all members of ***USB\_DEV\_T*** are automatically assigned by USB system software. The USB system software will assign a unique device number, read device descriptor and configuration descriptors, and create parent/child relationships. The definition of ***USB\_DEV\_T*** is listed below, and the detailed descriptions can be found in [Table 28-1: Members of \*USB\\_DEV\\_T\*](#)

```
typedef struct usb_device
{
    INT     devnum;
    INT     slow;
    enum
    {
        USB_SPEED_UNKNOWN = ,
        USB_SPEED_LOW,
        USB_SPEED_FULL,
        USB_SPEED_HIGH
    }     speed;
    struct usb_tt *tt;
    INT     ttport;
    INT     refcnt;
    UINT32 toggle[2];
```

```

    UINT32  halted[2];

    INT      epmaxpacketin[16];

    INT      epmaxpacketout[16];

    struct usb_device *parent;

    INT      hub_port;

    USB_BUS_T *bus;

    USB_DEV_DESC_T descriptor;

    USB_CONFIG_DESC_T *config;

    USB_CONFIG_DESC_T *actconfig;

    CHAR     **rawdescriptors;

    INT      have_langid;

    INT      string_langid;

    VOID     *hcpriv;

    INT      maxchild;

    struct usb_device *children[USB_MAXCHILDREN];

} USB_DEV_T;

```

Table 28-1: Members of **USB\_DEV\_T**

<b>Member</b>	<b>Description</b>
devnum	Device number on USB bus; each device instance has a unique device number
slow	Is low speed device speed ? (1: yes; 0: no)
speed	Device speed
refcnt	Reference count (to count the number of users using the device)
toggle[2]	Data toggle; one bit for each endpoint ( [0] = IN, [1] = OUT )
halted[2]	Endpoint halts; one bit for each endpoint ( [0] = IN, [1] = OUT )
epmaxpacketin[16]	IN endpoints specific maximum packet size (each entry represents for an IN endpoint of this device)
epmaxpacketout[16]	OUT endpoints specific maximum packet size (each entry represents for an OUT endpoint of this device)

	an OUT endpoint of this device)
parent	Parent device in the bus topology (generally, it should be a hub)
bus	The bus on which this device was presented
descriptor	Device descriptor
config	All of the configuration descriptors
actconfig	The descriptor of the active configuration
rawdescriptors	Raw descriptors for each configuration descriptor (driver can find class specific or vendor specific descriptors from the <i>rawdescriptors</i> )
have_langid	Whether string_langid is valid yet
string_langid	Language ID for strings
hcpriv	Host controller private data
maxchild	Number of ports if this is a hub device
children[ ]	Link to the downstream port device if this is a hub device

## 28.3. Descriptor Structures

In the USB\_DEV\_T structure, device descriptor, configuration descriptors, and raw descriptor are included. The USB Driver will acquire these descriptors from device automatically while the device is probed. The USB Driver issues GET\_DESCRIPTOR standard device request to acquire the configuration descriptors. It also parses the returned descriptors to create configuration-interface-endpoint descriptor links. Client software can obtain any configuration, interface, or endpoint descriptors by tracing the descriptor link started from USB\_DEV\_T. As USB Driver cannot understand class-specific and vendor-specific descriptors, it does not create link for these descriptors. If the client software wants to obtain any class-specific or vendor-specific descriptors, it can parse the descriptors stored in raw descriptor, which is the original descriptors list returned from the device. Table2-2, Table 2-3, Table 2-4, and Table 2-5 describe the structures defined for device descriptors, configuration descriptors, interface descriptors, and endpoint descriptors, respectively.

Figure 2-1 presents an overview on the relationship of these data structures. From USB\_DEV\_T (device instance structure), USB\_DEV\_DEC\_T (device descriptor structure) and USB\_CONFIG\_DEC\_T (configuration descriptor structure), USB\_IF\_DESC\_T (interface descriptor structure), to USB\_EP\_DESC\_T (endpoint descriptor structure), all structure entries are linked in top-down order.

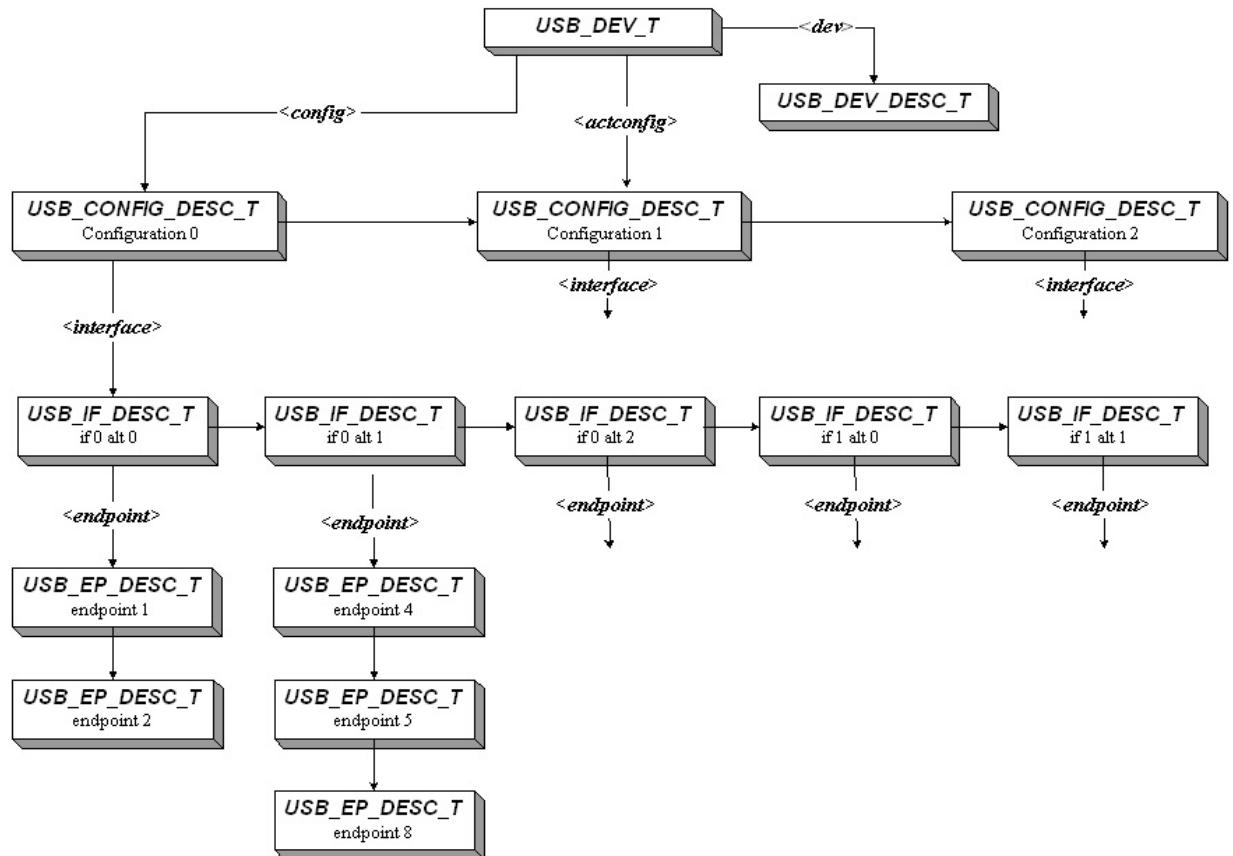


Figure 28-2:Descriptors relationship

```

/* Device descriptor */

typedef struct usb_device_descriptor
{
    __packed UINT8 bLength;
    __packed UINT8 bDescriptorType;
    __packed UINT16 bcdUSB;
    __packed UINT8 bDeviceClass;
    __packed UINT8 bDeviceSubClass;
    __packed UINT8 bDeviceProtocol;
    __packed UINT8 bMaxPacketSize0;
    __packed UINT16 idVendor;
    __packed UINT16 idProduct;
}
  
```

```

__packed UINT16 bcdDevice;
__packed UINT8 iManufacturer;
__packed UINT8 iProduct;
__packed UINT8 iSerialNumber;
__packed UINT8 bNumConfigurations;
} USB_DEV_DESC_T;

```

*Table 28-2: Members of **USB\_DEV\_DESC\_T***

Member	Description
bLength	Size of the descriptor in bytes
bDescriptorType	DEVICE descriptor type (0x01)
bcdUSB	USB specification release number in BCD format
bDeviceClass	Device class code
bDeviceSubclass	Device subclass code
bDeviceProtocol	Protocol code
bMaxPacketSize0	Maximum packet size for endpoint zero
idVendor	Vendor ID
idProduct	Product ID
iManufacturer	Device release number in BCD format
iProduct	Index of string descriptor describing product
iSerialNumber	Index of string descriptor describing the serial number
bNumConfigurations	Number of possible configurations

You may have found that the definition of **USB\_DEV\_DESC\_T** is fully compliant to the definition of device descriptor defined in USB 1.1 specification. In fact, the USB Driver acquires the device descriptor and fills it into this structure without making any modifications.

```

/* Configuration descriptor information.. */
typedef struct usb_config_descriptor
{
    __packed UINT8 bLength;

```

```

__packed UINT8    bDescriptorType;
__packed UINT16   wTotalLength;
__packed UINT8    bNumInterfaces;
__packed UINT8    bConfigurationValue;
__packed UINT8    iConfiguration;
__packed UINT8    bmAttributes;
__packed UINT8    MaxPower;
USB_IF_T  *interface;
UINT8   *extra;
INT     extralen;
} USB_CONFIG_DESC_T;

```

*Table 28-3: Members of USB\_CONFIG\_DESC\_T*

Member	Description
bLength	Size of the descriptor in bytes
bDescriptorType	CONFIGURATION descriptor type (0x02)
wTotalLength	The total length of data returned for this descriptor
bNumInterfaces	Number of interface supported by this configuration
bConfigurationValue	Value to use as an argument to the SetConfiguration() request to select the active configuration
iConfiguration	Index of string descriptor describing this configuration
bmAttributes	Bitmap describing the configuration characteristics
MaxPower	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational (in mA)
interface	Refer to the interface descriptor list (recorded in USB_IF_DESC_T structure format) returned by this configuration
extra	Refer to the memory buffer preserve the raw data of this configuration descriptor itself
extralen	The length of the <extra> memory buffer

The `dev->config` refers to a list of configurations supported by this device. Client software can access any configuration by indexing the configuration, for example, `dev->config[0]` is referred to the first configuration of this device. While `<config>` of `USB_DEV_T` refers to the configuration list, `<actconfig>` refers to the currently activated configuration. There can be only one configuration activated at the same time.

The structure members from `<bLength>` to `<MaxPower>` are fully compliant to that defined in USB 1.1 specification. The `<interface>` refers to a list of interfaces supported by this configuration. In addition, USB Driver keeps the interface descriptor itself in a dynamically allocated memory buffer, which is referred to by `<extra>`, and the length of this memory buffer is `<extralen>`.

An interface may contain several alternate settings. Each alternate setting has its own set of endpoints. USB Driver creates a single `USB_IF_DESC_T` structure for each alternate interface setting and links them in the order that they presented in the returned data of a configuration descriptor.

```
/* Interface descriptor */

typedef struct usb_interface_descriptor

{

    __packed UINT8 bLength;

    __packed UINT8 bDescriptorType;

    __packed UINT8 bInterfaceNumber;

    __packed UINT8 bAlternateSetting;

    __packed UINT8 bNumEndpoints;

    __packed UINT8 bInterfaceClass;

    __packed UINT8 bInterfaceSubClass;

    __packed UINT8 bInterfaceProtocol;

    __packed UINT8 iInterface;

    USB_EP_DESC_T *endpoint;

    UINT8 *extra;

    INT extralen;

} USB_IF_DESC_T;
```

Table 28-4: Members of USB\_IF\_DESC\_T

Member	Description
bLength	Size of the descriptor in bytes
bDescriptorType	INTERFACE descriptor type (0x04)
blInterfaceNumber	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
bAlternateSetting	Value used to select alternate setting for this interface
bNumEndpoints	Number of endpoints used by this interface (excluding endpoint zero)
blInterfaceClass	Class code
blInterfaceSubClass	Subclass code
blInterfaceProtocol	Protocol code
iInterface	Index of string descriptor describing this interface
endpoint	Refer to the endpoint descriptor list (recorded in USB_EP_DESC_T structure format) of this interface returned by this configuration
extra	Refer to the memory buffer preserve the raw data of this interface descriptor itself
extralen	The length of the <extra> memory buffer

The `dev->config[n]->interface` refers to a list of interfaces supported by configuration n. The structure members from `<bLength>` to `<iInterface>` are fully compliant to that defined in USB 1.1 specification. The `<endpoint>` refers to a list of endpoints supported by this interface. In addition, USB Driver keeps the interface descriptor itself in a dynamically allocated memory buffer, which is referred to by `<extra>`, and the length of this memory buffer is `<extralen>`.

```
/* Endpoint descriptor */

typedef struct usb_endpoint_descriptor

{
    __packed UINT8 bLength;
    __packed UINT8 bDescriptorType;
    __packed UINT8 bEndpointAddress;
    __packed UINT8 bmAttributes;
    __packed UINT16 wMaxPacketSize;
    __packed UINT8 bInterval;
```

```

__packed UINT8 bRefresh;
__packed UINT8 bSynchAddress;
UINT8 *extra;
INT extralen;
} USB_EP_DESC_T;

```

Table 28-5: Members of *USB\_EP\_DESC\_T*

Member	Description
bLength	Size of the descriptor in bytes
bDescriptorType	ENDPOINT descriptor type (0x05)
bEndpointAddress	The address of this endpoint
bmAttributes	Transfer type of this endpoint
wMaxPacketSize	The maximum packet size this endpoint is capable of sending or receiving
bInterval	Interval for polling endpoint for data transfers (in milliseconds)
bRefresh	Audio extensions to the endpoint descriptor
bSynchAddress	Audio extensions to the endpoint descriptor
extra	Refer to the memory buffer preserve the raw data of this endpoint descriptor itself
extralen	The length of the <extra> memory buffer

### ***DEV\_REQ\_T***

***DEV\_REQ\_T*** is used to represent the eight bytes device request in a control transfer. All device requests, including standard device requests, class-specific device requests, and vendor-specific device requests, are written in the ***DEV\_REQ\_T*** structure, which is also a member of a URB, and transferred to device through the control pipe.

```

typedef struct
{
__packed UINT8 requesttype;
__packed UINT8 request;
__packed UINT16 value;
__packed UINT16 index;
}

```

```

__packed UINT16 length;
} DEV_REQ_T;

```

Table 28-6: Members of *DEV\_REQ\_T*

Member	Description
requesttype	Characteristics of request
request	Specific request
value	Word-sized field that varies according to request
index	Word-sized field that varies according to request
length	Number of bytes to transfer if there is a DATA stage

### ***USB\_DEV\_ID\_T***

When the USB System Software detects a device being attached, it must find out the corresponding device driver for each of its interface from the registered driver list. It can try to invoke the *probe()* routine of each registered device driver for each device interface, but this is not efficient and time-consuming. If the USB System Software can make some simple judgment before trying invoking a device driver, it will be better. This is the purpose of ***USB\_DEV\_ID\_T***. The USB Library employ device ID to identify the appropriate device drivers.

When a device driver is registered to USB Driver, it may provide a device ID table, which is structured in ***USB\_DEV\_ID\_T*** format. In the device ID table, driver can specify the characteristics of the USB device interface that the driver would serve. If a driver does not provide a device ID table, then the USB Driver will always try to invoke it when a new device is detected.

The device driver can use device ID table to specify several checks of characteristics, including vendor ID, device ID, release number, device class, device subclass, device protocol, interface class, interface subclass, and interface protocol. The device driver can specify one or more checks. The more checks are specified, the more specific device interface can be identified. Table 2-7 lists the entries of device ID table.

```

typedef struct usb_device_id
{
    UINT16 match_flags;
    UINT16 idVendor;
    UINT16 idProduct;
    UINT16 bcdDevice_lo;
    UINT16 bcdDevice_hi;
    UINT8 bDeviceClass;
}

```

```

    UINT8 bDeviceSubClass;
    UINT8 bDeviceProtocol;
    UINT8 bInterfaceClass;
    UINT8 bInterfaceSubClass;
    UINT8 bInterfaceProtocol;
    UINT32 driver_info;
} USB_DEV_ID_T;

```

Table 28-7: Members of DEV\_REQ\_T

Member	Description
matchflag	A bitmask of flags, used to determine which of the following items are to be used for matching
idVendor	Used to compare the vendor ID recorded in device descriptor
idProduct	Used to compare the product ID recorded in device descriptor
bcdDevice_lo	Specify the low limit of device release number
bcdDevice_hi	Specify the high limit of device release number
bDeviceClass	Used to compare the class code in device descriptor
bDeviceSubClass	Used to compare the subclass code in device descriptor
bDeviceProtocol	Used to compare the protocol code in device descriptor
bInterfaceClass	Used to compare the class code in interface descriptor
bInterfaceSubClass	Used to compare the subclass code in interface descriptor
bInterfaceProtocol	Used to compare the protocol code in interface descriptor

There are 10 check items can be used to identify a specific type of device. To select which of these check items should be used to identify a device type is controlled by the <**matchflag**> member, which is a 16bits bit-mask flag. Each bit of <**matchflag**> is corresponding to one of these check items. The bit-map definition of <**matchflag**> is defined as the followings:

```

#define USB_DEVICE_ID_MATCH_VENDOR          0x0001
#define USB_DEVICE_ID_MATCH_PRODUCT         0x0002
#define USB_DEVICE_ID_MATCH_DEV_LO          0x0004
#define USB_DEVICE_ID_MATCH_DEV_HI          0x0008

```

```
#define USB_DEVICE_ID_MATCH_DEV_CLASS      0x0010
#define USB_DEVICE_ID_MATCH_DEV_SUBCLASS    0x0020
#define USB_DEVICE_ID_MATCH_DEV_PROTOCOL    0x0040
#define USB_DEVICE_ID_MATCH_INT_CLASS      0x0080
#define USB_DEVICE_ID_MATCH_INT_SUBCLASS   0x0100
#define USB_DEVICE_ID_MATCH_INT_PROTOCOL   0x0200
```

For convenience of driver implementation, the USB library also provides some useful macros that facilitate the development of device driver. These macros are all listed in the followings, you can also define your own macros:

```
/* Some useful macros */

#define USB_DEVICE(vend,prod) \
{ USB_DEVICE_ID_MATCH_DEVICE, vend, prod, 0, 0, \
  0, 0, 0, 0, 0, 0, 0 }

#define USB_DEVICE_VER(vend,prod,lo,hi) \
{ USB_DEVICE_ID_MATCH_DEVICE_AND_VERSION, vend, \
  prod, lo, hi, 0, 0, 0, 0, 0, 0, 0 }

#define USB_DEVICE_INFO(cl,sc,pr) \
{ USB_DEVICE_ID_MATCH_DEV_INFO, 0, 0, 0, 0, cl, \
  sc, pr, 0, 0, 0 }

#define USB_INTERFACE_INFO(cl,sc,pr) \
{ USB_DEVICE_ID_MATCH_INT_INFO, 0, 0, 0, 0, 0, \
  0, 0, cl, sc, pr, 0 }
```

## ***USB\_DRIVER\_T***

The USB library has defined a generalized structure for all USB device drivers. To implement a USB device driver based on this library, you must create such a structure and register it to the USB Driver. Once you have registered your device driver, the USB Driver can determine whether to launch your driver when a new device is attached.

As we will give detail introduction to the implementation of USB device driver, we only briefly describe the members of ***USB\_DRIVER\_T*** as following:

```
typedef struct usb_device_id
{
    UINT16  match_flags;
    UINT16  idVendor;
    UINT16  idProduct;
    UINT16  bcdDevice_lo;
    UINT16  bcdDevice_hi;
    UINT8   bDeviceClass;
    UINT8   bDeviceSubClass;
    UINT8   bDeviceProtocol;
    UINT8   bInterfaceClass;
    UINT8   bInterfaceSubClass;
    UINT8   bInterfaceProtocol;
    UINT32  driver_info;
} USB_DEV_ID_T;
```

*Table 28-8: Members of DEV\_REQ\_T*

<b>Member</b>	<b>Description</b>
matchflag	A bitmask of flags, used to determine which of the following items are to be used for matching
idVendor	Used to compare the vendor ID recorded in device descriptor
idProduct	Used to compare the product ID recorded in device descriptor
bcdDevice_lo	Specify the low limit of device release number
bcdDevice_hi	Specify the high limit of device release number
bDeviceClass	Used to compare the class code in device descriptor

bDeviceSubClass	Used to compare the subclass code in device descriptor
bDeviceProtocol	Used to compare the protocol code in device descriptor
bInterfaceClass	Used to compare the class code in interface descriptor
bInterfaceSubClass	Used to compare the subclass code in interface descriptor
bInterfaceProtocol	Used to compare the protocol code in interface descriptor

### URB\_T

USB specification has defined four transfer type: control, bulk, interrupt, and isochronous. In the USB library, all these four transfer types are accomplished by URB (USB Request Block).

Please refer to Chapter 3 for details about the implementation of each transfer type by using URB.

---

## 28.4. Data Transfer

USB specification defines four transfer types, control, bulk, interrupt, and isochronous. The USB device driver performs data transfers by preparing an URB and transfer it to the underlying USB system software. The URBs are designed to be accommodated with all four transfer types. By configuring the URB, USB device driver can specify the destination device interface and endpoint, the data buffer and data length to be transferred, the callback routine on completion, and other detail information. USB device driver passed the URB to the underlying USB system software, which will interpret the URB and accomplish the data transfers by initiating USB transactions between W90X900 Host Controller and the target device endpoint.

URB has been designed to be accommodated with all four USB data transfer types. Due to the characteristics of different transfer types, various requirements must be satisfied to fulfill the transfer. For example, URB contains *<setup\_packet>* for control transfer, *<interval>* for interval transfer, *<start\_frame>* and *<number\_of\_packets>* for isochronous transfer, and *<transfer\_buffer>* for all transfers. To implement a USB device driver, the programmers use URBs to accomplish all data transfers to all of the various endpoints.

For a specific endpoint, after delivering a URB to the underlying USB system software, the USB device driver must not deliver another URB to the same endpoint until the current transfer was done by the USB system software. That is, the driver must be blocked in waiting completion of the URB. URB includes a *<complete>* function pointer to solve the block waiting issue. The USB device driver provided a callback function and have *<complete>* pointer being referred to the callback function. On completion of this URB, the USB system software will invoke the callback function. Thus, the USB device driver was notified with the completion event, and can stop waiting. Note that the callback functions are invoked from an HISR, the execution time must be as short as possible.

## 28.5. Pipe Control

Before delivering an URB, the USB device driver must determine which device and which endpoint the URB will operate on. This destination device and endpoint is determined by <pipe> of URB.

<pipe> is actually a 32-bits unsigned integer. The USB library defined pipe structure with a 32-bits unsigned integer. The USB library has defined several useful macros for pipe control. The pipe is defined as the followings:

<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Pipe Type</b>		<b>Reserved</b>			<b>Speed</b>	<b>Reserved</b>	
<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Reserved</b>				<b>Data0/1</b>	<b>Endpoint</b>		
<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
	<b>Device</b>						
<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Direction</b>	<b>Reserved</b>					<b>Max Size</b>	

Table 28-9: Members of Pipe Control

Member	Description
Max Size [1 .. 0]	The maximum packet size. This field has been obsoleted. Now the maximum packet size is recorded in <epmaxpacketin> and <epmaxpacketout> fields of <i>USB_DEV_T</i> .
Direction[7]	Direction of data transfer. 0 = Host-to-Device [out]; 1 = Device-to-Host [in]
Device[8 .. 14]	Device number. This is the unique device address, which is assigned by Host Controller driver by <b>SET_ADDRESS</b> standard request. With this unique device number, the USB device driver can correctly locate the target device.
Endpoint[15 .. 18]	Endpoint number. This is the endpoint number on the target device, that the pipe is created with. By definition, a pipe corresponds to a unique endpoint on a unique device. By determining the device number and endpoint number, USB device driver can uniquely identify a specific endpoint of a specific device.

Data0/1[19]	Data toggle Data0/Data1. This bit is used to record the current data toggle condition.
Speed[26]	Endpoint transfer speed. 1 = Low speed; 0 = Full speed.
Pipe Type[30 .. 31]	Transfer type. 00 = isochronous; 01 = interrupt; 10 = control; 11 = bulk.

The USB library has provided a lot of macros facilities for USB device driver designer. The device driver can use the facilities to rescuer the trouble of managing bit fields. These macros are listed in the followings:

### Transfer Type

```
#define PIPE_ISOCHRONOUS          0
#define PIPE_INTERRUPT             1
#define PIPE_CONTROL               2
#define PIPE_BULK                  3

#define usb_pipetype(pipe)    (((pipe) >> 30) & 3)
#define usb_pipecontrol(pipe)  (usb_pipetype((pipe)) == PIPE_CONTROL)
#define usb_pipebulk(pipe)     (usb_pipetype((pipe)) == PIPE_BULK)
#define usb_pipeint(pipe)      (usb_pipetype((pipe)) == PIPE_INTERRUPT) \
#define usb_pipeisoc(pipe)    (usb_pipetype((pipe)) == PIPE_ISOCHRONOUS)
```

### Maximun Packet Size

```
#define usb_maxpacket(dev, pipe, out)  (out           \
? (dev)->epmaxpacketout[usb_pipeendpoint(pipe)]  \
: (dev)->epmaxpacketin [usb_pipeendpoint(pipe)] )
```

### Direction

```
#define usb_packetid(pipe) (((pipe) & USB_DIR_IN) ?           \
                           USB_PID_IN : USB_PID_OUT)
#define usb_pipeout(pipe)   (((pipe) >> 7) & 1) ^ 1
#define usb_pipein(pipe)   (((pipe) >> 7) & 1)
```

**Device Number**

```
#define usb_pipedevice(pipe)    (((pipe) >> 8) & 0x7f)
#define usb_pipe_endpdev(pipe)   (((pipe) >> 8) & 0x7ff)
```

**Endpoint Number**

```
#define usb_pipe_endpdev(pipe)  (((pipe) >> 8) & 0x7ff)
#define usb_pipeendpoint(pipe)   (((pipe) >> 15) & 0xf)
```

**Data Toggle**

```
#define usb_pipedata(pipe)      (((pipe) >> 19) & 1)

#define usb_gettoggle(dev, ep, out)           \
    (((dev)->toggle[out] >> ep) & 1)

#define usb_dotoggle(dev, ep, out)           \
    ((dev)->toggle[out] ^= (1 << ep))

#define usb_settoggle(dev, ep, out, bit)       \
    ((dev)->toggle[out] =           \
     ((dev)->toggle[out] & ~(1 << ep)) | \
     ((bit) << ep))
```

**Speed**

```
#define usb_pipeslow(pipe)    (((pipe) >> 26) & 1)
```

**Pipe Creation**

```
static __inline UINT32 __create_pipe(USB_DEV_T *dev, UINT32 endpoint)
{
    return (dev->devnum << 8) | (endpoint << 15) | (dev->slow << 26);

}

static __inline UINT32 __default_pipe(USB_DEV_T *dev)
```

```
{  
    return (dev->slow << 26);  
  
}  
  
/* Create various pipes... */  
  
#define usb_sndctrlpipe(dev,endpoint)      \  
    (0x80000000 | __create_pipe(dev,endpoint))  
  
#define usb_rcvctrlpipe(dev,endpoint)       \  
    (0x80000000 | __create_pipe(dev,endpoint) | USB_DIR_IN)  
  
#define usb_sndisocpipe(dev,endpoint)        \  
    (0x00000000 | __create_pipe(dev,endpoint))  
  
#define usb_rcvisocpipe(dev,endpoint)        \  
    (0x00000000 | __create_pipe(dev,endpoint) | USB_DIR_IN)  
  
#define usb_sndbulkpipe(dev,endpoint)         \  
    (0xC0000000 | __create_pipe(dev,endpoint))  
  
#define usb_rcvbulkpipe(dev,endpoint)         \  
    (0xC0000000 | __create_pipe(dev,endpoint) | USB_DIR_IN)  
  
#define usb_sndintpipe(dev,endpoint)          \  
    (0x40000000 | __create_pipe(dev,endpoint))  
  
#define usb_rcvintpipe(dev,endpoint)          \  
    (0x40000000 | __create_pipe(dev,endpoint) | USB_DIR_IN)  
  
#define usb_snddefctrl(dev)                  \  
    (0x80000000 | __default_pipe(dev))  
  
#define usb_rcvdefctrl(dev)                  \  
    (0x80000000 | __default_pipe(dev) | USB_DIR_IN)
```

## 28.6. Control Transfer

IN this section, we will introduce how to make control transfers by URBs. A control transfer is accomplished by sending a device request to the control endpoint of the target device. Depend on the request sent to device, there may be data stage or not.

The URB provided a *<setup\_packet>* field to accommodate the device request command. The USB device driver must have the *<setup\_packet>* of its URB being referred to an *<unsigned char>* array, which contained the device request command to be transferred. Note that *<setup\_packet>* is designed to be used with control transfer.

If a device request included data stage, the data to be transferred must be referred to by the *<transfer\_buffer>* pointer of URB. If the device request required data to be sent from Host to Device, the USB device driver must prepare a DMA buffer (non-cacheable) and fill the data to be transferred into this buffer. Then, the USB device driver have *<transfer\_buffer>* pointer refer to this buffer, and specify the length of the buffer with *<transfer\_buffer\_length>* of the URB. If the device request requires data to be sent from Device to Host, the USB device driver must prepare a DMA buffer to receive the data from Device. Again, the USB device driver used *<transfer\_buffer>* and *<Transfer\_buffer\_length>* to describe its DMA buffer. The *<actual\_length>* is written by USB system software to tell the device driver how many bytes are actually transferred.

The USB device driver also has to prepare a callback function to be invoked by the USB system software. The callback function will be invoked on the completion of URB, in spite of success or fail. Generally, the callback function is responsible for waking up the task that delivered the URB. The callback function may also check the status of the URB to determine the transfer is successful or not. The following is an example of control transfer.

```
static VOID ctrl_callback(URB_T *urb)
{
    PEGASUS_T *pegasus = urb->context;

    switch (urb->status)
    {
        case USB_ST_NOERROR:
            if (pegasus->flags & ETH_REGS_CHANGE)
            {
                pegasus->flags &= ~ETH_REGS_CHANGE;
                pegasus->flags |= ETH_REGS_CHANGED;
                update_eth_regs_async(pegasus);
            }
    }
}
```

```
        return;

    }

    break;

case USB_ST_URB_PENDING:

    return;

case USB_ST_URB_KILLED:

    break;

default:

    printf("Warning - status %d\n", urb->status);

}

pegasus->flags &= ~ETH_REGS_CHANGED;

if (pegasus->flags & CTRL_URB_SLEEP)

{

    pegasus->flags &= ~CTRL_URB_SLEEP;

    NU_Set_Events(&pegasus->events, 1, NU_OR); /* set event */

}

}

static INT get_registers(PEGASUS_T *pegasus, UINT16 indx, UINT16 size, VOID *data)

{

    INT      ret;

    UINT8   *dma_data;

    while (pegasus->flags & ETH_REGS_CHANGED)

    {

        pegasus->flags |= CTRL_URB_SLEEP;

        USB_printf("ETH_REGS_CHANGED waiting...\n");

        NU_Retrieve_Events(&pegasus->events, 1, NU_AND,
```

```
        (unsigned long *)&ret, NU_SUSPEND);

}

dma_data = (UINT8 *)USB_malloc(size, BOUNDARY_WORD);

if (!dma_data)

    return -ENOMEM;

pegasus->dr->requesttype = PEGASUS_REQT_READ;

pegasus->dr->request = PEGASUS_REQ_GET_REGS;

#ifndef LITTLE_ENDIAN

    pegasus->dr->value = 0;

    pegasus->dr->index = idx;

    pegasus->dr->length = size;

#else

    pegasus->dr->value = USB_SWAP16(0);

    pegasus->dr->index = USB_SWAP16(idx);

    pegasus->dr->length = USB_SWAP16(size);

#endif

    pegasus->ctrl_urb.transfer_buffer_length = size;

FILL_CONTROL_URB(&pegasus->ctrl_urb, pegasus->usb,

                  usb_rcvctrlpipe(pegasus->usb,0),

                  (UINT8 *)pegasus->dr,

                  dma_data, size, ctrl_callback, pegasus );

pegasus->flags |= CTRL_URB_SLEEP;

NU_Set_Events(&pegasus->events, 0, NU_AND); /* clear event */

USB_SubmitUrb(&pegasus->ctrl_urb);
```

```

NU_Retrieve_Events(&pegasus->events, 1, NU_AND,
                    (unsigned long *)&ret, NU_SUSPEND);

        memcpy(data, dma_data, size);

out:
    USB_free(dma_data);

    return ret;
}

```

In the above example, the device driver first prepare the device request command in `<pegasus->dr>`, which was later referred to by `<urb->setup_packet>`. It request a buffer for DMA transfer by `USB_malloc()`. Note that `USB_malloc()` will allocate a non-cacheable memory buffer. It then created a Control-In pipe by using `usb_rcvctrlpipe` macro, and the endpoint number is 0. The device driver the use the `FILL_CONTROL_URB` macro facility to fill the URB. The callback function is `ctrl_callback()`, which is provided by the device driver itself. After submitting the URB, the caller task suspend on waiting the `<pegasus->events>` event set. On completion of this URB, the USB system software will invoke `ctrl_callback()`, and `ctrl_callback()` will set the `<pegasus->events>` event to wake up the caller task.

---

## 28.7. Bulk Transfer

IN this section, we will introduce how to make bulk transfers by URBs. The URB provided `<transfer_buffer>` and `<transfer_buffer_length>` to accommodate data to be transferred to or from device. The direction of transfer is determined by the direction bit of bulk pipe. The transfer length is unlimited. If you are familiar with OpenHCI specification, you may understand that the maximum transfer size of a bulk transfer is 4096 bytes. If the transfer length of your URB exceeds 4096 bytes, the USB system software will split it into several transfer units smaller than 4096 bytes. Thus, you can specify unlimited transfer buffer length, only the physical memory can limit the size.

The transfer buffer must be non-cacheable. A designer can use `USB_malloc()` to acquire a block of non-cacheable memory.

The USB device driver also has to prepare a callback function to be invoked by the USB system software. The callback function will be invoked on the completion of URB, in spite of success or fail. Generally, the callback function is responsible for waking up the task that delivered the URB. The callback function may also check the status of the URB to determine the transfer is successful or not. The following is an example of bulk transfer

```

/* In Host Controller HISR context */

static VOID write_bulk_callback(URB_T *urb)
{

```

```
PEGASUS_T          *pegasus = urb->context;
STATUS             previous_int_value;
DV_DEVICE_ENTRY   *device;

_PegasusDevice->tx_ready = 1;

/* Get a pointer to the device. */
device = DEV_Get_Dev_By_Name("Pegasus");

/* Lock out interrupts. */
previous_int_value = NU_Control_Interrupts(NU_DISABLE_INTERRUPTS);

DEV_Recover_TX_Buffers(device);

/* If there is another item on the list, transmit it. */
if (device->dev_transq.head)
{
    /* Re-enable interrupts */
    NU_Control_Interrupts(previous_int_value);

    /* Transmit the next packet. */
    PegasusTransmit(device, device->dev_transq.head);

}

/* Re-enable interrupts. */
NU_Control_Interrupts(previous_int_value);

if (urb->status)
    USB_printf("write_bulk_callback - TX error status: %d\n",
               urb->status);
}
```

```
STATUS PegasusTransmit(DV_DEVICE_ENTRY *dev, NET_BUFFER *netBuffer)

{
    INT      ret, wait=0;
    UINT8   *buf_ptr;
    INT      totalLength = 0;

    while (!_PegasusDevice->tx_ready)
    {
        NU_Sleep(1);           /* wait on any outgoing Tx */
        if (wait++ > NU_PLUS_Ticks_Per_Second)
        {
            USB_Printf("Can't transmit packet!\n");
            return NU_IO_ERROR;
        }
    }

    buf_ptr = _PegasusDevice->tx_buff + 2;
    do
    {
        memcpy(buf_ptr, netBuffer->data_ptr, netBuffer->data_len);
        totalLength += netBuffer->data_len;
        buf_ptr += netBuffer->data_len;

        /* Move on to the next buffer. */
        netBuffer = netBuffer->next_buffer;
    } while (netBuffer != 0);
}
```

```
/* The first two bytes record the packet length. */

buf_ptr = _PegasusDevice->tx_buff;

buf_ptr[0] = totalLength & 0xff;

buf_ptr[1] = (totalLength >> 8) & 0xff;

FILL_BULK_URB(&_PegasusDevice->tx_urb, _PegasusDevice->usb,
               usb_sndbulkpipe(_PegasusDevice->usb, 2),
               (CHAR *)buf_ptr, PEGASUS_MAX_MTU,
               write_bulk_callback, _PegasusDevice);

_PegasusDevice->tx_urb.transfer_buffer_length =
    ((totalLength+2) & 0x3f) ? totalLength+2 : totalLength+3;

_PegasusDevice->tx_ready = 0;

USB_SubmitUrb(&_PegasusDevice->tx_urb);

return NU_SUCCESS;

}
```

---

## 28.8. Interrupt Transfer

IN this section, we will introduce how to make interrupt transfers by URBs. The URB provided *<transfer\_buffer>* and *<transfer\_buffer\_length>* to accommodate data to be transferred to or from device, and *<interval>* to specify polling interval of the interrupt transfer. The direction of transfer is determined by the direction bit of interrupt pipe. The transfer length is dependent on target interrupt endpoint.

The transfer buffer must be non-cacheable. A designer can use **USB\_malloc()** to acquire a block of non-cacheable memory.

The USB device driver also has to prepare a callback function to be invoked by the USB system software. The callback function will be invoked if there's data received in one of the interrupt interval. In the callback function, USB device driver can read *<transfer\_buffer>* to retrieve received interrupt data. The USB device driver have not to modify URB or resend URB. The USB library will resend the

interrupt URB after callback. The interrupt URB will not stop until hardware failure or explicitly deleted by the USB device driver.

```
static VOID intr_callback(URB_T *urb)

{

    PEGASUS_T *pegasus = urb->context;

    UINT8     *d;

    if (!pegasus)

        return;

    switch (urb->status)

    {

        case USB_ST_NOERROR:

            break;

        case USB_ST_URB_KILLED:

            return;

        default:

            break;

    }

    d = urb->transfer_buffer;

    if (d[2] & 0x1)

        UART_printf("Rx error - overflow!!\n");

}

FILL_INT_URB(&_PegasusDevice->intr_urb, _PegasusDevice->usb,
             usb_rcvintpipe(_PegasusDevice->usb, 3),
```

```

(CHAR *)&_PegasusDevice->intr_buff[0], 8,
intr_callback, _PegasusDevice,
_PegasusDevice->intr_interval);

res = USB_SubmitUrb(&_PegasusDevice->intr_urb);

if (res)

UART_printf("pegasus_open - failed intr_urb %d\n", res);

```

---

## 28.9. USB Core Library API

### ***USB\_PortInit***

#### **Synopsis**

INT USB\_PortInit (UINT32 u32PortType);

#### **Description**

The function is used to specified USB host port type.

#### **Parameter**

u32PortType:

*Table 28-10: Members of Pipe Control*

u32PortType	Description
HOST_LIKE_PORT0	USB host output from GPIOB[1:0]. It is a host like port
HOST_LIKE_PORT1	USB host output from GPIOA[4:3]. It is a host like port
HOST_NORMAL_PORT0_ONLY	USB host output from normal USB transceiver port 0.
HOST_NORMAL_TWO_PORT	USB host output from normal USB transceiver port 0 and port 1.

#### **Return Value**

None

#### **Example**

```
/* In/out through host like port 0 */
USB_PortInit(HOST_LIKE_PORT0);
USB_PortDisable(FALSE, TRUE);
InitUsbSystem();
UMAS_InitUmasDriver();
```

## ***USB\_PortDisable***

### **Synopsis**

```
VOID USB_PortDisable(BOOL bIsDisPort0, BOOL bIsDisPort1);
```

### **Description**

The function is used to disable USB hoost ports if the port is useless.

### **Parameter**

bIsDisPort0	TRUE to disable port 0. FALSE to enable port 0
bIsDisPort1	TRUE to disable port 1. FALSE to enable port 1

### **Return Value**

None

### **Example**

```
/* In/out through host like port 0 and diable port 1 */

USB_PortInit(HOST_LIKE_PORT0);

USB_PortDisable(FALSE, TRUE);

InitUsbSystem();

UMAS_InitUmasDriver();
```

## ***InitUsbSystem***

### **Synopsis**

```
INT InitUsbSystem (VOID)
```

### **Description**

Initialize the USB hardware and USB core library. This function must be invoked before any other functions. The USB library will scan device at this time, but the device will not be activated until the corresponding device driver was registered by `USB_RegisterDriver()`.

**Parameter**

None

**Return Value**

0 – Success

Otherwise – Failure

**Example**

```
/*
Initialize NVTFAT FAT file system, USB core system, and USB mass storage driver
*/
fsInitFileSystem();

USB_PortInit(HOST_LIKE_PORT0);

USB_PortDisable(FALSE, TRUE);

InitUsbSystem();

UMAS_InitUmasDriver();
```

***DeInitUsbSystem*****Synopsis**

INT DeInitUsbSystem(VOID)

**Description**

De-Initialize the USB hardware and USB core library.

**Parameter**

None

**Return Value**

0 – Success

**Example**

```
/*
Initialize NVTFAT FAT file system, USB core system, and USB mass storage driver
*/
fsInitFileSystem();
```

```
USB_PortInit(HOST_LIKE_PORT0);

USB_PortDisable(FALSE, TRUE);

InitUsbSystem();

UMAS_InitUmasDriver();

....;

/* De-Initialize USB core library */

DeInitUsbSystem();
```

## ***UMAS\_InitUmasDriver***

### **Synopsis**

```
INT UMAS_InitUmasDriver (VOID)
```

### **Description**

Initialize the USB mass storage driver. fsInitFileSystem() and InitUsbSystem() must be called prior to this API. Once an USB mass storage device detected, USB core library will initialize it and mount it to NVTFAT file system automatically.

### **Parameter**

None

### **Return Value**

0 – Success

Otherwise – Failure

### **Example**

```
/*
Initialize NVTFAT FAT file system, USB core system, and USB mass storage driver
*/
fsInitFileSystem();

USB_PortInit(HOST_LIKE_PORT0);

USB_PortDisable(FALSE, TRUE);

InitUsbSystem();

UMAS_InitUmasDriver();
```

## ***USB\_RegisterDriver***

### **Synopsis**

```
INT USB_RegisterDriver (USB_DRIVER_T *driver)
```

### **Description**

Register a device driver with the USB library. In this function, USB library will also try to associate the newly registered device driver with all connected USB devices that have no device driver associated with it. Note that a connected USB device can be detected by USB library but may not work until it was associated with its corresponding device driver.

### **Parameter**

driver               The USB device driver to be registered with USB core library

### **Return Value**

0 – Success

Otherwise – Failure

### **Example**

```
static USB_DRIVER_T usblp_driver =  
{  
    "usblp",  
    usblp_probe,  
    usblp_disconnect,  
    {NULL,NULL},  
    {0},  
    NULL,  
    usblp_ids,  
    NULL,  
    NULL  
};  
  
INT UsbPrinter_Init() {
```

```
    if (USB_RegisterDriver(&usblp_driver)) return -1;  
  
    return 0;  
}
```

## ***USB\_DeregisterDriver***

### **Synopsis**

```
VOID USB_DeregisterDriver(USB_DRIVER_T *driver)
```

### **Description**

Deregister a device driver.

### **Parameter**

driver            The device driver to be deregistered

### **Return Value**

0 – Success

Otherwise – Failure

### **Example**

```
VOID UsbPrinter_Exit()  
  
{  
  
    USB_DeregisterDriver(&usblp_driver);  
  
}
```

## ***USB\_AllocateUrb***

### **Synopsis**

```
URB_T *USB_AllocateUrb(INT iso_packets)
```

### **Description**

Creates an urb for the USB driver to use and returns a pointer to it. The driver should call `USB_FreeUrb()` when it is finished with the urb

### **Parameter**

iso\_packets    The number of isochronous frames in a single URB.  
For other transfer types, this value must be zero.

### **Return Value**

- NULL - Failure
- Otherwise - A pointer to the newly allocated URB

#### **Example**

```
_W99683_Camera->sbuf[i].urb = USB_AllocateUrb(FRAMES_PER_DESC);

if (_W99683_Camera->sbuf[i].urb == NULL)

{

    UART_printf("%s - USB_AllocateUrb(%d.) failed.\n", proc,

                FRAMES_PER_DESC);

    Return -1;

};
```

## **USB\_FreeUrb**

#### **Synopsis**

VOID USB\_FreeUrb(URB\_T \*urb)

#### **Description**

Frees the memory used by a urb.

#### **Parameter**

None

#### **Return Value**

None

#### **Example**

None

## **USB\_SubmitUrb**

#### **Synopsis**

INT USB\_SubmitUrb(URB\_T \*urb)

#### **Description**

Submit a URB for executing data transfer

#### **Parameter**

urb Pointer to the URB to be serviced.

**Return Value**

0 – Success

Otherwise – Failure

**Example**

```
/* prepare URB */

FILL_BULK_URB(&_PegasusDevice->tx_urb, _PegasusDevice->usb,
               usb_sndbulkpipe(_PegasusDevice->usb, 2),(CHAR *)buf_ptr,
               PEGASUS_MAX_MTU,
               write_bulk_callback, _PegasusDevice);

/* set the data length to be transferred */

_PegasusDevice->tx_urb.transfer_buffer_length =
    ((totalLength+2) & 0x3f) ? totalLength+2 : totalLength+3;

_PegasusDevice->tx_ready = 0;

/* submit URB */

if (USB_SubmitUrb(&_PegasusDevice->tx_urb) != 0)
{
    UART_printf("Warning - failed tx_urb %d\n", ret);
    return NU_IO_ERROR;
}
```

**USB\_UnlinkUrb****Synopsis**

INT USB\_UnlinkUrb(URB\_T \*urb)

**Description**

Unlink a URB which has been submitted but not finished

**Parameter**

urb pointer to the URB to be unlinked

**Return Value**

0 – Success

Otherwise – Failure

#### Example

```
INT PegasusClose()

{
    _PegasusDevice->flags &= ~PEGASUS_RUNNING;

    if (!(_PegasusDevice->flags & PEGASUS_UNPLUG))
        disable_net_traffic(_PegasusDevice);

    USB_UnlinkUrb(&_PegasusDevice->rx_urb);
    USB_UnlinkUrb(&_PegasusDevice->tx_urb);
    USB_UnlinkUrb(&_PegasusDevice->ctrl_urb);

#ifdef PEGASUS_USE_INTR
    USB_UnlinkUrb( &_PegasusDevice->intr_urb );
#endif

    return 0;
}
```

## **USB\_SendBulkMessage**

### Synopsis

```
INT USB_SendBulkMessage(USB_DEV_T *dev,
                        UINT32 pipe,
                        VOID *data,
                        INT len,
                        INT *actual_length,
                        INT timeout)
```

### Description

Builds a bulk urb, sends it off and waits for completion. This function sends a simple bulk message to a specified endpoint and waits for the message to complete, or timeout. Don't use this function from within an interrupt context.

#### Parameter

dev	pointer to the usb device to send the message to
pipe	endpoint "pipe" to send the message to
data	pointer to the data to send
len	length in bytes of the data to send
actual_length	pointer to a location to put the actual length transferred in bytes
timeout	time to wait for the message to complete before timing out (if 0 the wait is forever)

#### Return Value

0 – Success

Otherwise – Failure

#### Example

```
if (!pb->pipe)

    pipe = usb_rcvbulkpipe (s->usbdev, 2);

else

    pipe = usb_sndbulkpipe (s->usbdev, 2);

    ret = USB_SendBulkMessage(s->usbdev, pipe, pb->data, pb->size,
&actual_length, 100);

    if (ret<0) {

        err("dabusb: usb_bulk_msg failed(%d)",ret);

        if (usb_set_interface (s->usbdev, _DABUSB_IF, 1) < 0) {

            err("set_interface failed");

            return -EINVAL;

        }

    }

USB_malloc
```

#### Synopsis

```
VOID *USB_malloc(INT wanted_size,
```

INT boundary)

### Description

Allocate a non-cacheable memory block started from assigned boundary. The total size of the USB library managed memory block is 256KB.

### Parameter

wanted_size	The wanted size of non-cacheable memory block
boundary	The start address boundary of the memory block. It can be BOUNDARY_BYTE, BOUNDARY_HALF_WORD, BOUNDARY_WORD, BOUNDARY32, BOUNDARY64, BOUNDARY128, BOUNDARY256, BOUNDARY512, BOUNDARY1024, BOUNDARY2048, BOUNDARY4096.

### Return Value

NULL	Failed, there is not enough memory or USB library is not started
Otherwise	pointer to the newly allocated memory block

### Example

```
UINT8 *dma_data;

dma_data = USB_malloc(len, BOUNDARY_WORD);

if (dma_data == NULL) {

    NU_printf("usblp_ctrl_msg - Memory not enough!\n");

    return -1;

}

retval = USB_SendControlMessage(usblp->dev,

    dir ? usb_rcvctrlpipe(usblp->dev, 0) : usb_sndctrlpipe(usblp->dev, 0),

    request, USB_TYPE_CLASS | dir | recip, value, usblp->ifnum, dma_data,

    len, HZ * 5);

    memcpy(buf, dma_data, len);

USB_free(dma_data);
```

## USB\_free

### Synopsis

```
VOID  USB_free(VOID *alloc_addr)
```

**Description**

Free the memory block allocated by USB\_malloc().

**Parameter**

alloc\_addr      pointer to the USB\_malloc() allocated memory block to be freed.

**Return Value**

None

**Example**

Same as USB\_malloc()

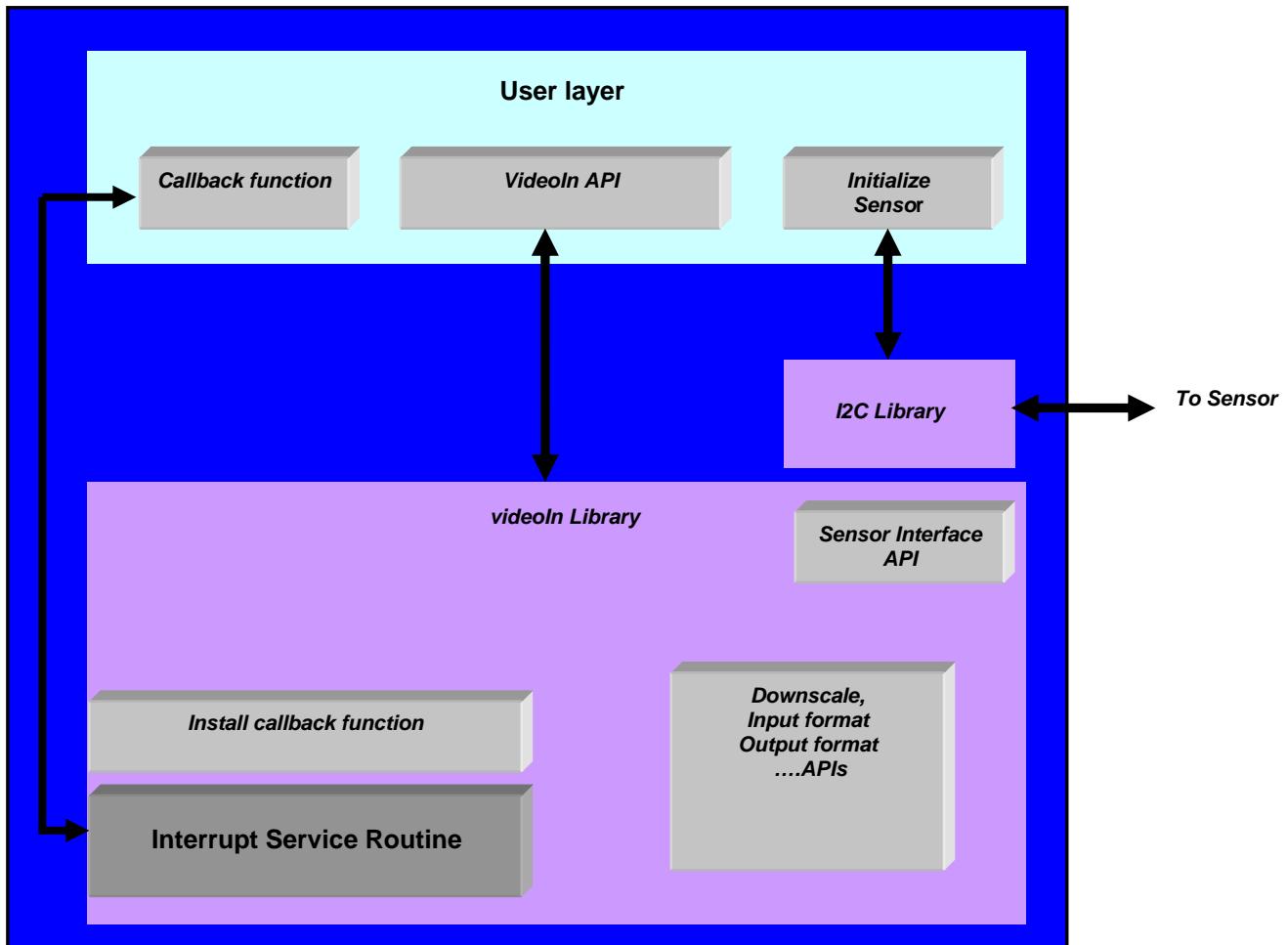
# 29. VIDEOIN Library Overview

## 29.1. Features

The VIDEOIN Library has the following features:

- Programmable sensor input format YCbCr422/RGB565.
- Programmable output packet format YCbCr422/RGB565/RGB555/Y-only to frame buffer.
- Programmable output planar format YUV422/YUV420/macro block YUV420 to frame buffer
- Support to enable or disable planar and packet pipes for encode and preview respectively.
- Programmable different downscale factor for planar and packet pipes.
- Support cropping image.
- Programmable CCIR601 and CCIR656 input interface.
- Programmable input polarity of pixel clock, h-sync and v-sync.
- Support indoor motion detection.
- Max support 2 sensor ports.

## 29.2. VIDEOIN Library Description



To initialize this sensor, just call the I2C library. However, programmer has to enable sensor clock before initialize sensor through I2C bus.

## 29.3. VIDEOIN Library API

### *register\_vin\_device*

#### Synopsis

```
INT32 register_vin_device(UINT32 u32port, VINDEV_T* pVinDev );
```

**Description**

The function has to be called before calling other videoIn APIs. It gets back a instant base on the specified port. All of the API functions will be operation base on the instant.

**Parameter**

u32Port	1: Capture port 1
	2: Capture port 2

**Return Value**

-1 or Successful

**Example**

```
VINDEV_T Vin;
VINDEV_T* pVin;
INT32 i32ErrCode;
i32ErrCode = register_vin_device(1, &Vin); /* Register capture 1 */
if(i32ErrCode<0){
    sysprintf("Register vin 0 device fail\n");
    return -1;
}
pVin = &Vin;
pVin->Init(TRUE, eSYS_UPPLL, 24000, eVIDEOIN_SNR_CCIR601);
```

***Init*****Synopsis**

```
void (*Init)(BOOL bIsEnableSnrClock,
            IDEOIN_SNR_SRC eSnrSrc,
            NT32 u32SensorFreqKHz,
            VIDEOIN_DEV_TYPE eDevType)
```

**Description**

The function has to be called before calling other videoIn APIs exception function-**register\_vin\_device()**. It enables sensor clock. So before initialize sensor, the sensor clock has to be also enabled. It is also specified the multiple pin function for the specified device type.

**Parameter**

bIsEnableSnrClock	TRUE to enable sensor clock. FALSE to disable sensor clock.
eSnrSrc	eSYS_UPLL or eSYS_ALL
u32SensorFreq	Specified the sensor clock to be initialized by I2C. Unit: KHz
eDevType function pins	Input device type. Please refer <a href="#">Table 29-1: Input device multiple function pins</a>

*Table 29-1: Input device multiple function pins*

eDevType	Value	Description
eVIDEOIN_SNRL_CCIR656	0	Sensor input CCIR656 format. The device type is only used if capture port 1. Sensor interface is through GPIOB port.
eVIDEOIN_SNRL_CCIR601	1	Sensor input CCIR601 format. The device type is only used if capture port 1. Sensor interface is through GPIOB port.
eVIDEOIN_TVD_CCIR656	2	TV decoder input CCIR656. The device type is only used if capture port 1. Sensor interface is through GPIOB port.
eVIDEOIN_TVD_CCIR601	3	TV decoder input CCIR601. The device type is only used if capture port 1. Sensor interface is through GPIOB port.
eVIDEOIN_2ND_SNRL_CCIR656	4	Sensor input CCIR656 format. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPE port.
eVIDEOIN_2ND_SNRL_CCIR601	5	Sensor input CCIR601 format. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPE port.
eVIDEOIN_2ND_TVD_CCIR656	6	TV decoder input CCIR656. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPE port.
eVIDEOIN_2ND_TVD_CCIR601	7	TV decoder input CCIR601. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPE port..
eVIDEOIN_3RD_SNRL_CCIR656	8	Sensor input CCIR656 format. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPD port.
eVIDEOIN_3RD_SNRL_CCIR601	9	Sensor input CCIR601 format. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPD port.

eVIDEOIN_3RD_TVD_CCIR656	10	TV decoder input CCIR656. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPD port.
eVIDEOIN_3RD_TVD_CCIR601	11	TV decoder input CCIR601. The device type is only used if capture port 2. Sensor interface is through GPIOC/GPA/GPD port.

**Return Value**

None

**Example**

```
VINDEV_T Vin;
VINDEV_T* pVin;
INT32 i32ErrCode;

i32ErrCode = register_vin_device(1, &Vin); /* Register capture 1 */

if(i32ErrCode<0){

    sysprintf("Register vin 0 device fail\n");

    return -1;

}

pVin = &Vin;

pVin->Init(TRUE, eSYS_UPPLL, 24000, eVIDEOIN_SNR_CCIR601);
```

**Open****Synopsis**

```
INT32 (*Open)(UINT32 u32EngFreqKHz,
               UINT32 u32SensorFreqKHz)
```

**Description**

Open videoin device.

**Parameter**

u32EngFreqKHz      It is useless.

u32SensorFreq      Sensor works frequency. Unit: KHz

**Return Value**

0 – Success

**Example**

```
VINDEV_T Vin;  
  
VINDEV_T* pVin;  
  
INT32 i32ErrCode;  
  
i32ErrCode = register_vin_device(1, &Vin); /* Register capture 1 */  
  
if(i32ErrCode<0){  
  
    sysprintf("Register vin 0 device fail\n");  
  
    return -1;  
  
}  
  
pVin = &Vin;  
  
pVin->Init(TRUE, eSYS_UPPLL, 24000, eVIDEOIN_SNR_CCIR601);  
  
pVin->Open(48000, 24000); /* Sensor clock 24MHz */
```

**Close****Synopsis**

```
void (*Close)(void)
```

**Description**

Close videoIn device.

**Parameter**

None

**Return Value**

None

**Example**

```
VINDEV_T Vin;  
  
VINDEV_T* pVin;  
  
INT32 i32ErrCode;  
  
i32ErrCode = register_vin_device(1, &Vin); /* Register capture 1 */
```

```

if(i32ErrCode<0){

    sysprintf("Register vin 0 device fail\n");

    return -1;

}

pVin = &Vin;

pVin->Init(TRUE, eSYS_UPPLL, 24000, eVIDEOIN_SNR_CCIR601);

pVin->Open(48000, 24000); /* Sensor clock 24MHz */

.....

pVin->Close();

```

## ***SetPipeEnable***

### **Synopsis**

void (\*SetPipeEnable)(BOOL bEngEnable, E\_VIDEOIN\_PIPE ePipeEnable)

### **Description**

Enable or disable engine and specified which pipes is enabled or disabled

### **Parameter**

bEngEnable     TRUE: Enable VideoIn engine

                  FALSE: Disable VideoIn engine

ePipeEnable    Pipes enable or disable. Please refer Table 29-2: Pipes Type

*Table 29-2: Pipes Type*

eDevType	Value	Description
eVIDEOIN_BOTH_PIPE_DISABLE	0	Both planar and packet pipes are disabled
eVIDEOIN_PLANAR	1	Enable planar pipe only
eVIDEOIN_PACKET	2	Enable packet pipe only
eVIDEOIN_BOTH_PIPE_ENABLE	3	Both planar and packet pipes are enabled

### **Return Value**

None

### **Example**

```

VINDEV_T Vin;

VINDEV_T* pVin;

INT32 i32ErrCode;

i32ErrCode = register_vin_device(1, &Vin); /* Register capture 1 */

if(i32ErrCode<0){

    sysprintf("Register vin 0 device fail\n");

    return -1;

}

pVin = &Vin;

pVin->Init(TRUE, eSYS_UPLL, 24000, eVIDEOIN_SNR_CCIR601);

pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->SetPipeEnable(TRUE, /* Engine enable */

                     eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

.....

pVin->Close();

```

## **InstallCallback**

### **Synopsis**

```

INT32 (*InstallCallback)(E_VIDEOIN_INT_TYPE eIntType,
                        PFN_VIDEOIN_CALLBACK pfnCallback,
                        PFN_VIDEOIN_CALLBACK *pfnOldCallback)

```

### **Description**

Install call back function for user layer. The function let the videoIn library call back to upper lay to inform user the frame end event. And pass some information to user layer.

### **Parameter**

eIntType      Interrupt type. Please refer [Table 29-3: Interrupt type](#)

*Table 29-3: Interrupt type*

eIntType	Value	Description
eVIDEOIN_MDINT	0x100000	Motion Detection Interrupt
eVIDEOIN_ADDRMINT	0x80000	Address match interrupt. It is only

		support packet pip
eVIDEOIN_MEINT	0x20000	Memory Error.
eVIDEOIN_VINT	0x10000	Frame end interrupt

pfnCallback      Function pointer for callback function.

pfnOldCallback    Old callback function.

#### Return Value

Successful or E\_VIDEOIN\_INVALID\_INT.

#### Example

```

/* Install call back function for frame end */

void VideoIn_InterruptHandler(UINT8 u8PacketBufID,
                               UINT8 u8PlanarBufID,
                               UINT8 u8FrameRate,
                               UINT8 u8Filed)

{ //Frame end

    ...

}

.....



pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
                        &pfnOldCallback); /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT); /* Enable frame end interrupt */

pVin->SetPipeEnable(TRUE, /* Engine enable */

eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

.....



pVin->Close();

```

## **EnableInt**

## Synopsis

INT32 (\*EnableInt)(E\_VIDEOIN\_INT\_TYPE eIntType)

## Description

Enable specified interrupt type.

## Parameter

eIntType      Reference    Please refer Table 29-3: Interrupt type

Table 29-3: Interrupt type

### Return Value

Successful or E VIDEOIN INVALID INT

## Example

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,

&pfnOldCallback      ); /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT); /* Enable frame end interrupt */

pVin->SetPipeEnable(TRUE, /* Engine enable */

eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

.....

pVin->Close();
```

### **DisableInt**

## Synopsis

INT32 (\*DisableInt)(E\_VIDEOIN\_INT\_TYPE eIntType)

## Description

Disable specified interrupt type.

## Parameter

eIntType      Reference    Please refer Table 29-3: Interrupt type

Table 29-3: Interrupt type

**Return Value**

Successful or E\_VIDEOIN\_INVALID\_INT

**Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VVideoIn_InterruptHandler,
&pfnOldCallback      /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT);    /* Enable frame end interrupt */

pVin->EnableInt(eVIDEOIN_MDINT);   /* Enable motion detection interrupt */

pVin->SetPipeEnable(TRUE, /* Engine enable */

eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

.....

pVin->DisableInt(eVIDEOIN_MDINT); /* Enable motion detection interrupt */
```

***SetInputType*****Synopsis**

```
void (*SetInputType)(UINT32 u32FieldEnable,
                     E_VIDEOIN_TYPE eInputType,
                     BOOL bFieldSwap)
```

**Description**

Specified the input device type. It also specified which fields enable and filesd swap if need in TV decoder input type.

**Parameter**

u32FieldEnable 0, 1, 2 or 3. It is only useful if TV decoder.

0: both field disable. 1: field one enable.

2: field two enable. 3: noth fields enable.

eInputType Input device type. Please refer [Table 29-4: Inpu Device Type](#)

bFieldSwap Both field swap. It is only useful if TV decoder.

TRUE: swap

FALSE: unswap

*Table 29-4: Inpu Device Type*

eIntType	Value	Description
eVIDEOIN_TYPE_CCIR601	0	Input device is CCIR601
eVIDEOIN_TYPE_CCIR656	1	Input device is CCIR656

**Return Value**

None

**Example**

```

pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
&pfnOldCallback      /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT);    /* Enable frame end interrupt */

pVin->EnableInt(eVIDEOIN_MDINT);   /* Enable motion detection interrupt */

pVin->SetInputType(3,
eVIDEOIN_TYPE_CCIR656,
FALSE);

pVin->SetPlanarFormat(eVIDEOIN_PLANAR_YUV422); /* planar YUV422 */

pVin->SetPipeEnable(TRUE, /* Engine enable */

eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

```

**SetSensorPolarity****Synopsis**

```

void (*SetSensorPolarity)(BOOL bVsync,
                         BOOL bHsync,
                         BOOL bPixelClk)

```

**Description**

Specified the polarity for vertical synchronisation, horizontal synchronisation and pixel clock signals.

**Parameter**

bVsync	TRUE: Vertical synchronisation period is high level FALSE: Vertical synchronisation period is low level
bHsync	TRUE: Horizontal synchronisation period is high level FALSE: Horizontal synchronisation period is low level
bPixelClk	TRUE: Latch data in rising edge FALSE: Latch data in falling edge

**Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->SetSensorPolarity(FALSE,
                         FALSE,
                         TRUE);

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */
                       (PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
                       &pfnOldCallback      /* Installed callback */
```

***SetDataFormatAndOrder*****Synopsis**

```
void (*SetDataFormatAndOrder)(E_VIDEOIN_ORDER eInputOrder,
                           E_VIDEOIN_IN_FORMAT eInputFormat,
                           E_VIDEOIN_OUT_FORMAT eOutputFormat);
```

**Description**

Specified the sensor input format and input order. And specified the packet output format

**Parameter**

- eInputOrder Sensor data input order. Please refer [Table 29-5:Input Order](#)
- eInputFormat Sensor data format. Please refer [Table 29-6:Input Format](#)
- eOutputFormat Packet pipe output format. Please refer [Table 29-7:Packet Output Format](#)

*Table 29-5:Input Order*

eInputOrder	Value	Description
eVIDEOIN_IN_UYVY	0	Input order is UYVYUYVY...

eVIDEOIN_IN_YUYV	1	Input order is YUYVYUYV...
eVIDEOIN_IN_VYUY	2	Input order is VYUYVYUY...
eVIDEOIN_IN_YVYU	3	Input order is YVYUYVYU...

*Table 29-6:Input Format*

eInputOrder	Value	Description
eVIDEOIN_IN_YUV422	0	YUV422 format
eVIDEOIN_IN_RGB565	1	RGB565 format

*Table 29-7:Packet Output Format*

eInputOrder	Value	Description
eVIDEOIN_OUT_YUV422	0	Packet YUV422
eVIDEOIN_OUT_ONLY_Y	1	Packet Y only
eVIDEOIN_OUT_RGB555	2	Packet RGB555
eVIDEOIN_OUT_RGB565	3	Packet RGB565

**Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->SetSensorPolarity(FALSE,
                         FALSE,
                         TRUE);

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */
                       (PFN_VIDEOIN_CALLBACK)VvideoIn_InterruptHandler,
                       &pfnOldCallback); /* Installed callback */
```

**SetCropWinSize****Synopsis**

```
void (*SetCropWinSize)(UINT32 u32height, UINT32 u32width)
```

**Description**

Specified the cropping size

**Parameter**

- |           |   |
|-----------|---|
| u32height | The height of cropping window.<br>The values should less than or equal to height of sensor dimension. |
| u32width  | The width of cropping window.<br>The values should less than or equal to width of sensor dimension.   |

**Return Value**

None

**Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */
                      (PFN_VIDEOIN_CALLBACK)VVideoIn_InterruptHandler,
                      &pfnOldCallback); /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT); /* Enable frame end interrupt */

pVin->EnableInt(eVIDEOIN_MDINT); /* Enable motion detection interrupt */

pVin->SetCropWinSize(480,
                     640);

pVin->SetCropWinStartAddr(0, /* VerticalStart start position*/
                          0); /* HorizontalStart start position*/
```

***SetCropWinStartAddr*****Synopsis**

```
void (*SetCropWinStartAddr)(UINT32 u32VerticalStart,
                           UINT32 u32HorizontalStart)
```

**Description**

Specified the cropping start position

**Parameter**

- |                    |                               |
|--------------------|-------------------------------|
| u32VerticalStart   | The start position of Y axis. |
| u32HorizontalStart | The start position of X axis. |

**Return Value**

None

**Example**

```

pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
&pfnOldCallback      /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT);    /* Enable frame end interrupt */

pVin->EnableInt(eVIDEOIN_MDINT);   /* Enable motion detection interrupt */

pVin->SetCropWinSize(480,
640);

pVin->SetCropWinStartAddr(0, /* VerticalStart start position*/
0);      /* HorizontalStart start position*/

```

## **PreviewPipeSize**

### **Synopsis**

```
void (*PreviewPipeSize)( UINT16 u16height,
                        UINT16 u16width)
```

### **Description**

Specified the packet pipe dimension for preview

### **Parameter**

u16height	The height of packet pipe.
u16width	The width of packet pipe.

### **Return Value**

None

### **Example**

```

pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
&pfnOldCallback      /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT);    /* Enable frame end interrupt */

pVin->EnableInt(eVIDEOIN_MDINT);   /* Enable motion detection interrupt */

```

```
pVin->SetCropWinSize(960,  
                      1280);  
  
pVin->SetCropWinStartAddr(0, /* VerticalStart start position*/  
                           0);      /* HorizontalStart start position*/  
  
pVin->PreviewPipeSize(480,  
                       640);  
  
pVin->EncodePipeSize(960,  
                      1280);
```

## EncodePipeSize

### Synopsis

```
void (*EncodePipeSize)( UINT16 u16height,  
                        UINT16 u16width)
```

### Description

Specified the planar pipe dimension for encoding.

### Parameter

u16height	The height of planar pipe.
u16width	The width of planar pipe.

### Return Value

None

### Example

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */  
  
pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */  
                      (PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,  
                      &pfnOldCallback      /* Installed callback */)  
  
pVin->EnableInt(eVIDEOIN_VINT);    /* Enable frame end interrupt */  
pVin->EnableInt(eVIDEOIN_MDINT);   /* Enable motion detection interrupt */  
  
pVin->SetCropWinSize(960,  
                      1280);  
  
pVin->SetCropWinStartAddr(0,        /* VerticalStart start position*/  
                           0);      /* HorizontalStart start  
position*/
```

```
pVin->PreviewPipeSize(480,
                      640);

pVin->EncodePipeSize(960,
                      1280);
```

## ***SetStride***

### **Synopsis**

```
void (*SetStride)(UINT32 u32packetstride,
                  UINT32 u32planarstride)
```

### **Description**

Specified the stride of packet and planar pipes.

### **Parameter**

u32packetstride	The stride of packet.
u32planarstride	The stride of planar.

### **Return Value**

None

### **Example**

```
pVin->SetCropWinSize(960,
                      1280);

pVin->SetCropWinStartAddr(0,           /* VerticalStart start position*/
                           0);      /* HorizontalStart start position*/

pVin->PreviewPipeSize(480,
                      640);

pVin->EncodePipeSize(960,
                      1280);

pVin->SetStride(640, 1280);
```

## ***GetStride***

### **Synopsis**

```
void (*GetStride)(PUINT32 pu32packetstride,
                  PUINT32 pu32planarstride)
```

### **Description**

Get the stride of packet and planar pipes.

#### Parameter

pu32packetstride	The stride of packet.
pu32planarstride	The stride of planar.

#### Return Value

None

#### Example

```
UINT32 u32PacketStride, u32PlanarStride;

pVin->SetCropWinSize(960,
                      1280);

pVin->SetCropWinStartAddr(0,           /* VerticalStart start position*/
                           0);       /* HorizontalStart start position*/

pVin->PreviewPipeSize(480,
                       640);

pVin->EncodePipeSize(960,
                      1280);

pVin->GetStride(&u32PacketStride, &u32PlanarStride);

pVin->SetStride(u32PacketStride, 1280);
```

## ***SetPlanarFormat***

#### Synopsis

void (\*SetPlanarFormat)(E\_VIDEOIN\_PLANAR\_FORMAT ePlanarFmt)

#### Description

Specified the planar format.

#### Parameter

ePlanarFmt	Planar format. Please reference Table 29-8 Planar Format
------------	--

*Table 29-8 Planar Format*

eIntType	Value	Description
eVIDEOIN_PLANAR_YUV422	0	Planar YUV422
eVIDEOIN_PLANAR_YUV420	1	Planar YUV420
eVIDEOIN_MACRO_PLANAR_YUV420	2	Planar macro block YUV420

**Return Value**

None

**Example**

```
/* Disable frame end interrupt */

pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

(PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
&pfnOldCallback      /* Installed callback */

pVin->EnableInt(eVIDEOIN_VINT);    /* Enable frame end interrupt */

pVin->EnableInt(eVIDEOIN_MDINT);/*Enable motion detection interrupt */

pVin->SetPlanarFormat(eVIDEOIN_PLANAR_YUV422); /* planar YUV422 */

pVin->SetPipeEnable(TRUE, /* Engine enable */

eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

.....

pVin->DisableInt(eVIDEOIN_MDINT); /*Enable motion detection interrupt*/
```

***SetBaseStartAddress*****Synopsis**

```
INT32 (*SetBaseStartAddress)(E_VIDEOIN_PIPE ePipe,
                           E_VIDEOIN_BUFFER eBuf,
                           UINT32 u32BaseStartAddr)
```

**Description**

Specified planar and packet buffer base address.

**Parameter**

ePipe                   eVIDEOIN\_PLANAR or eVIDEOIN\_PACKET.

Please refer Table 29-2: Pipes Type

eBuf                   Buffer number. Please refer [Table 29-9:Buffer Number](#)

u32BaseStartAddr      Buffer base address.

*Table 29-9:Buffer Number*

eIntType	Value	Description
eVIDEOIN_BUF0	0	Planar Y Buffer
eVIDEOIN_BUF1	1	Planar U Buffer
eVIDEOIN_BUF2	2	Planar V Buffer

### Example

```

/* Disable frame end interrupt */

    pVin->Open(48000, 24000); /* Sensor clock 24MHz */

    pVin->InstallCallback(eVIDEOIN_VINT,           /* Frame end interrupt */

                           (PFN_VIDEOIN_CALLBACK)VideoIn_InterruptHandler,
                           &pfnOldCallback); /* Installed callback */

    pVin->EnableInt(eVIDEOIN_VINT); /* Enable frame end interrupt */

    pVin->EnableInt(eVIDEOIN_MDINT); /* Enable motion detection interrupt */

    pVin->SetPlanarFormat(eVIDEOIN_PLANAR_YUV422); /* planar YUV422 */

    pVin->SetPipeEnable(TRUE, /* Engine enable */

                         eVIDEOIN_BOTH_PIPE_ENABLE); /* Both pipes enable */

/* Specified Packet Buffer */

    pVin->SetBaseStartAddress(eVIDEOIN_PACKET,
                               0,
                               (UINT32)((UINT32)pu8FrameBuffer0);

/* Specified Planar Y/U/V Buffer */

    pVin->SetBaseStartAddress(eVIDEOIN_PLANAR,
                               0, /* 0 means Y Buffer */
                               (UINT32)u8PlanarFrameBuffer);

    pVin->SetBaseStartAddress(eVIDEOIN_PLANAR,
                               1, /* 1 means U Buffer */

```

```
(UINT32)u8PlanarFrameBuffer+
OPT_ENCODE_WIDTH*OPT_ENCODE_HEIGHT);

pVin->SetPlanarFormat(eVIDEOIN_PLANAR_YUV422);

pVin->SetBaseStartAddress(eVIDEOIN_PLANAR,
2, /* 2 means V Buffer */
(UINT32)u8PlanarFrameBuffer+
OPT_ENCODE_WIDTH*OPT_ENCODE_HEIGHT+
OPT_ENCODE_WIDTH*OPT_ENCODE_HEIGHT/2);
```

## ***SetOperationMode***

### **Synopsis**

```
void (*SetOperationMode)( BOOL bIsOneSutterMode)
```

### **Description**

VideoIn engine works in one shutter mode or continuous mode.

### **Parameter**

bIsOneSutterMode	VideoIn engine operation mode
TRUE:	One shutter mode.
FALSE:	Continuous mode.

### **Return Value**

None

### **Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

.....

pVin->SetOperationMode(TRUE); /* Take one frame */

while(pVin->SetOperationMode()==TRUE);/* VideoIn engine still keeps working*/
/* VideoIn engine is stopped here */

/* Feed frame buffer to JPEG codec engine*/

.....
```

```
/* Restart VideoIn engine */

pVin->SetPipeEnable(TRUE, eVIDEOIN_BOTH_PIPE_ENABLE);
```

## **GetOperationMode**

### **Synopsis**

BOOL (\*GetOperationMode)( void)

### **Description**

Check VideoIn engine is stopped after set it into one shutter mode. It is only valid after set VideoIn engine into one shutter mode. If VideoIn engine operation in continuous mode, it is invalid.

### **Parameter**

None

### **Return Value**

TRUE: VideoIn is still keeping working

FALSE: VideoIn engine is stopped after one shutter.

### **Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

.....

pVin->SetOperationMode(TRUE); /* Take one frame */

while(pVin->SetOperationMode() == TRUE); /* VideoIn engine still keeps working*/
    /* VideoIn engine is stopped here */

/* Feed frame buffer to JPEG codec engine*/

.....

/* Restart VideoIn engine */

pVin->SetPipeEnable(TRUE, eVIDEOIN_BOTH_PIPE_ENABLE);
```

## **SetColorEffect**

### **Synopsis**

INT32 (\*SetColorEffect)( E\_VIDEOIN\_CEF eColorMode);

### **Description**

Set color effect mode.

**Parameter**

eColorMode      Color effect mode

**Return Value**

Successful

**Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->SetColorEffect(eVIDEOIN_CEF_POSTERIZE);

pVin->SetColorEffectParameter(0xFC, 0xFC, 0xFC);
```

***SetColorEffectParameter*****Synopsis**

```
INT32 (*SetColorEffectParameter)(UINT8 u8YComp,
                                UINT8 u8UComp,
                                UINT8 u8VComp);
```

**Description**

Set color effect parameter.

**Parameter**

u8YComp      Y component  
u8UComp      U Component  
u8VComp      V Component

**Return Value**

Successful or E\_VIDEOIN\_WRONG\_COLOR\_PARAMETER

**Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

pVin->SetColorEffect(eVIDEOIN_CEF_POSTERIZE);

pVin->SetColorEffectParameter(0xFC, 0xFC, 0xFC);
```

***SetMotionDet*****Synopsis**

```
INT32 (*SetMotionDet)( BOOL bEnable,
                       BOOL bBlockSize,
```

BOOL bSaveMode)

### Description

Enable motion detection and set relate parameter.

### Parameter

bEnable	Enable or disable motion detection TRUE: Enable FALSE: Disable
bBlockSize	Block size TRUE: Block size 8x8 FALSE: Block size 16x16
bSaveMode	Save format for motion detection TRUE: 1 bit DIFF + 7 Y differential FALSE: 1 bit Diff only

### Return Value

None

### Example

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */

/* Enable motion detection with block size 8x8 and 1 bit DIFF+7 Y
differential */

pVin-> SetMotionDet(TRUE, TRUE, TRUE);

/* Threshold = 0x20 */

/* Output DIFF buffer in address 2MB */

/* Temp Y buffer in address 2.5MB */

pVin-> SetMotionDetEx(0x20, 0x200000, 0x280000);
```

## ***SetMotionDetEx***

### Synopsis

```
INT32 (*SetMotionDetEx)( UINT32 u32Threshold,
                         UINT32 u32OutBuffer
                         UINT32 u32LumBuffer)
```

### Description

Set motion detection relate parameter.

#### Parameter

u32Threshold Threshold between 2 motion detection frames  
u32OutBuffer DIFF output buffer address  
u32LumBuffer Motion detection temporary Y component output buffer address

#### Return Value

None

#### Example

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */  
  
/* Enable motion detection with block size 8x8 and 1 bit DIFF+7 Y  
differential */  
  
pVin->SetMotionDet(TRUE, TRUE, TRUE);  
  
/* Threshold = 0x20 */  
  
/* Output DIFF buffer in address 2MB */  
  
/* Temporary Y buffer in address 2.5MB */  
  
pVin->SetMotionDetEx(0x20, 0x200000, 0x280000);
```

## **SetStandardCCIR656**

#### Synopsis

```
void (*SetStandardCCIR656)( (BOOL bIsStandard)
```

#### Description

The function is only used for Hinix HI-702 sensor CCIR656 mode. Because it is not standard CCIR656.

#### Parameter

bIsStandard Input device support standard CCIR656 mode or not.  
TRUE: Support standard CCIR656 mode  
FALSE: Support Hi-702 CCIR656 mode

#### Return Value

None

#### Example

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */
```

```
/* Support standard CCIR656 */  
pVin-> SetStandardCCIR656 (TRUE);
```

## **SetShadowRegister**

### **Synopsis**

```
void (*SetShadowRegister)( void)
```

### **Description**

Some register can not be updated in capturing image such as downscale, cropping start position and cropping window size and buffer start address. For the reason, there are some shadow register for those register as mention above. After update those register as above, set shadow register bit will update those registers in frame end.

### **Parameter**

None

### **Return Value**

None

### **Example**

```
pVin->Open(48000, 24000); /* Sensor clock 24MHz */  
  
pVin->InstallCallback(eVIDEOIN_VINT, /* Frame end interrupt */  
  
(PFN_VIDEOIN_CALLBACK)VVideoIn_InterruptHandler,  
    &pfnOldCallback /* Installed callback */;  
  
pVin->EnableInt(eVIDEOIN_VINT); /* Enable frame end interrupt */  
pVin->EnableInt(eVIDEOIN_MDINT); /*Enable motion detection interrupt*/  
  
pVin->SetCropWinSize(960,  
    1280);  
  
pVin->SetCropWinStartAddr(0, /* VerticalStart start position*/  
    0); /* HorizontalStart start position*/  
  
pVin->PreviewPipeSize(480,  
    640);  
  
pVin->EncodePipeSize(960,  
    1280);
```

```
pVin->SetShadowRegister();
```

## 29.4. Error Code Table

Code Name	Value	Description
E_VIDEOIN_INVALID_INT	0xFFFF1001	Invalid interrupt channel
E_VIDEOIN_INVALID_BUF	0xFFFF1002	Invalid buffer
E_VIDEOIN_INVALID_PIPE	0xFFFF1003	Invalid pipe
E_VIDEOIN_INVALID_COLOR_MODE	0xFFFF1004	Invalid color mode
E_VIDEOIN_WRONG_COLOR_PARAMETER	0xFFFF1005	Invalid color parameter

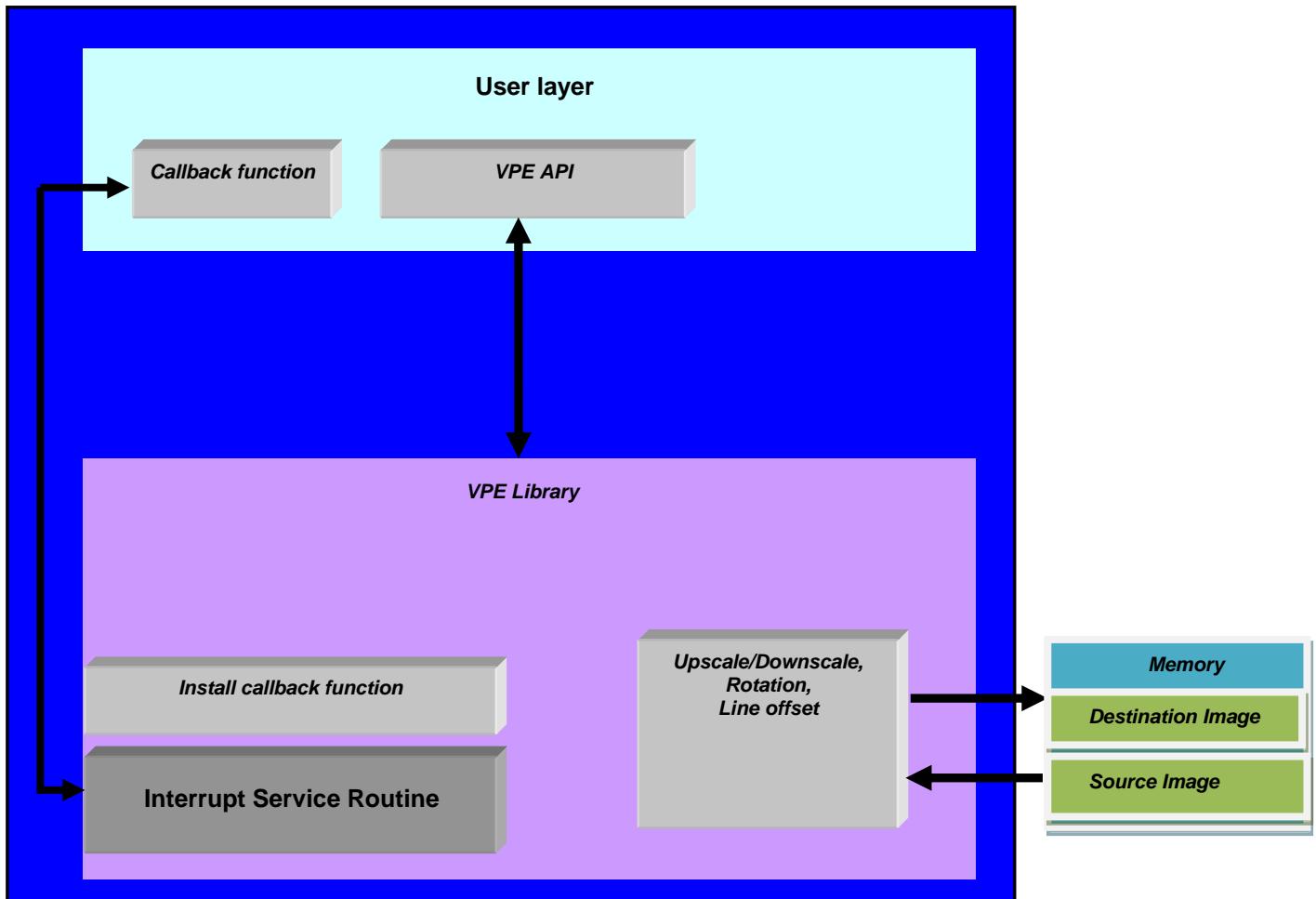
# 30. VPE Library Overview

## 30.1. Features

The VPE Library has the following features:

- Support format conversion
  - 1. Input format
    - Y only
    - Planar YUV420
    - Planar YUV411
    - Planar YUV422
    - Planar YUV422 Transpose
    - Planar YUV444
    - Packet YUV422
    - Packet RGB555
    - Packet RGB565
    - Packet RGB888
  - 2. Output format
    - Packet YUV422
    - Packet RGB555
    - Packet RGB565
    - Packet RGB888
- Support rotation
  - 1. Normal
  - 2. Right
  - 3. Left
  - 4. Upside down
  - 5. Horizontal mirror
  - 6. 180 degree
- Support MMU or Non-MMU mode.
  - 1. MMU.
    - Block base mode with better memory usage.
  - 2. Non-MMU mode
    - Line base mode with better performance.
- Support source and destination line offset
- Support limited on the fly mode with codec.

## 30.2. VPE Library Description



## 30.3. VPE Library API

### **vpeOpen**

#### Synopsis

ERRCODE

vpeOpen(void)

**Description**

Open VPE library.

**Parameter**

None

**Return Value**

Successful or error code.

**Example**

```
vpeOpen(); /* Enable VPE clock and interrupt */
```

**vpeClose****Synopsis**

```
ERRCODE vpeClose (void)
```

**Description**

Close VPE library.

**Parameter**

None

**Return Value**

None

**Example**

```
vpeOpen();  
...  
vpeClose();
```

**vpeInstallCallback****Synopsis**

```
ERRCODE vpeInstallCallback(E_VPE_INT_TYPE eIntType,  
                           PFN_VPE_CALLBACK pfnCallback,  
                           PFN_VPE_CALLBACK* pfnOldCallback)
```

**Description**

Install call back function for user layer. The function let the VPE library call back to upper lay to inform user for registered interrupt event. However, page fault and page missing will be handled in library. Upper layer can ignore both

#### Parameter

eIntType      Interrupt type.

*Table 30-1: Interrupt type*

eIntType	Value	Description
VPE_INT_COMP	0x0	Conversion complete
VPE_INT_PAGE_FAULT	0x1	Page fault if MMU on.
VPE_INT_PAGE_MISS	0x2	Page miss if MMU on.
VPE_INT_MB_COMP	0x3	Macro block complete if on the fly with codec.
VPE_INT_MB_ERR	0x4	Macro block error if on the fly with codec.
VPE_INT_DMA_ERR	0x5	DMA target abort if MMU on. The interrupt occurrence is only the page table is retrieved by OS and VPE is working. If the interrupt occurrence may memory is destroyed by VPE engine. So the application must write a signal handler for stopping the vpe engine.

pfnCallback      Function pointer for callback function.

pfnOldCallback    Old callback function.

#### Return Value

Successful or error code.

#### Example

```
/* Install call back function for conversion done */

vpeOpen();

vpeInstallCallback(VPE_INT_COMP,
                    vpeCompleteCallback,
                    &OldVpeCallback);

vpeEnableInt(VPE_INT_COMP);

vpeEnableInt(VPE_INT_PAGE_FAULT);

vpeEnableInt(VPE_INT_PAGE_MISS);
```

## vpeEnableInt

### Synopsis

```
ERRCODE vpeEnableInt(E_VPE_INT_TYPE eIntType)
```

### Description

Enable specified interrupt type.

### Parameter

eIntType      Reference    Please refer Table 29-3: [Interrupt type](#)

*Table 29-3: Interrupt type*

### Return Value

Successful or error code

### Example

```
/* Enable frame end interrupt */

vpeEnableInt(VPE_INT_COMP);

vpeEnableInt(VPE_INT_PAGE_FAULT);

vpeEnableInt(VPE_INT_PAGE_MISS);
```

## vpeDisableInt

### Synopsis

```
ERRCODE vpeDisableInt(E_VPE_INT_TYPE eIntType)
```

### Description

Disable specified interrupt type.

### Parameter

eIntType      Reference    Please refer Table 29-3: [Interrupt type](#)

*Table 29-3: Interrupt type*

### Return Value

Successful or error code

### Example

```
/* Disable frame end interrupt */

vpeDisableInt(VPE_INT_COMP);
```

**vpeIoctl****Synopsis**

```
ERRCODE vpeIoctl (UINT32 u32Cmd,  
                    UINT32 u32Element,  
                    UINT32 u32Arg0,  
                    UINT32 u32Arg1)
```

**Description**

VPE IO control function. The function is used to set some parameters for VPE hardware IP.

**Parameter**

<i>u32Cmd</i>	<i>Reference</i>
---------------	------------------

Table 30-2: IO Control table

	<i>u32Element</i>		<i>Reference</i>
VPE_IOCTL_TRIGGER	(Useless)	(Useless)	(Useless)
VPE_IOCTL_CHECK_TRIGGER	(Useless)	(Useless)	(Useless)
VPE_IOCTL_SET_MMU_ENTRY	TRUE: Enable MMU operation	TLB entry.	(Useless)
VPE_IOCTL_SET_TLB_ENTRY	Component entry: 0~7 0 and 4: For packet or Y component 1 and 5: For U component 2 and 6: For V component 3 and 7: For destination image	(Useless)	(Useless)
			Specified the component entry if page fault. <b>It has been handled by library</b>

Table 30-2: IO Control table

	<i>u32Arg0</i>		<i>Reference</i>	
VPE_IOCTL_TRIGGER	(Useless)	(Useless)	(Useless)	Trigger VPE
VPE_IOCTL_CHECK_TRIGGER	(Useless)	(Useless)	(Useless)	Check VPE complete Return 0, meaning VPE complete
VPE_IOCTL_SET_MMU_ENTRY	TRUE: Enable MMU operation	TLB entry.	(Useless)	Enable VPE MMU operation.
VPE_IOCTL_SET_TLB_ENTRY	Component entry: 0~7 0 and 4: For packet or Y component 1 and 5: For U component 2 and 6: For V component 3 and 7: For destination image	(Useless)	(Useless)	Specified the component entry if page fault. <b>It has been handled by library</b>

Table 30-2: IO Control table

	<i>u32Arg1</i>		<i>Reference</i>	
VPE_IOCTL_TRIGGER	(Useless)	(Useless)	(Useless)	Trigger VPE
VPE_IOCTL_CHECK_TRIGGER	(Useless)	(Useless)	(Useless)	Check VPE complete Return 0, meaning VPE complete
VPE_IOCTL_SET_MMU_ENTRY	TRUE: Enable MMU operation	TLB entry.	(Useless)	Enable VPE MMU operation.
VPE_IOCTL_SET_TLB_ENTRY	Component entry: 0~7 0 and 4: For packet or Y component 1 and 5: For U component 2 and 6: For V component 3 and 7: For destination image	(Useless)	(Useless)	Specified the component entry if page fault. <b>It has been handled by library</b>

Table 30-2: IO Control table

VPE_IOCTL_TRIGGER	(Useless)	(Useless)	(Useless)	Trigger VPE
VPE_IOCTL_CHECK_TRIGGER	(Useless)	(Useless)	(Useless)	Check VPE complete Return 0, meaning VPE complete
VPE_IOCTL_SET_MMU_ENTRY	TRUE: Enable MMU operation	TLB entry.	(Useless)	Enable VPE MMU operation.
VPE_IOCTL_SET_TLB_ENTRY	Component entry: 0~7 0 and 4: For packet or Y component 1 and 5: For U component 2 and 6: For V component 3 and 7: For destination image	(Useless)	(Useless)	Specified the component entry if page fault. <b>It has been handled by library</b>

u32Cmd	u32Arg0	u32Arg1	u32Arg2	Description
VPE_IOCTL_SET_SRCBUF_ADDR	Source packet buffer start address or planar Y buffer address	Source planar U buffer start address	Source planar V buffer start address	Specified the source buffer base address
VPE_IOCTL_SET_DSTBUF_ADDR	Destination packet buffer start address.	(Useless)	(Useless)	Specified the input order, input format and output format
VPE_IOCTL_SET_SRC_OFFSET	Left offset for source image	Right offset for source image	(Useless)	Specified the left and right offset for source image.
VPE_IOCTL_SET_SRC_DIMENSION	Width of source image	Height of source image	(Useless)	Specified the dimension of source image.
VPE_IOCTL_SET_DST_DIMENSION	Width of destination image	Height of destination image	(Useless)	Specified the dimension of destination image.
VPE_IOCTL_SET_COLOR_RANGE	Source format color range. TRUE: Y 16~235, U and V 16~240. FALSE: source format is full range.	Destination format color range. TRUE: Conversion to Y 16~235. U and V 16~240. FALSE: Full range for destination format	(Useless)	Specified color range
VPE_IOCTL_SET_FILTER	VPE_SCALE_DDA: Directly drop algorithm VPE_SCALE_BILINEAR: Bilinear algorithm	(Useless)	(Useless)	Specified upscale or downscale algorithm
VPE_IOCTL_SET_FORMAT	Source format	Destination format	(Useless)	Specified the source format and destination format
VPE_IOCTL_SET_MACRO_BLOCK	Y macro block if on the fly	X macro block if on the fly	(Useless)	It is useful if on the fly with codec.
VPE_IOCTL_HOST_OP	VPE_HOST_FRAME: Block bas mode VPE_HOST_VDEC_LINE: Line base mode	Rotation direction.	(Useless)	Specified the operation mode and rotation direction.

Table 30-2: IO Control table

VPE_IOCTL_TRIGGER	(Useless)	(Useless)	(Useless)	Trigger VPE
VPE_IOCTL_CHECK_TRIGGER	(Useless)	(Useless)	(Useless)	Check VPE complete Return 0, meaning VPE complete
VPE_IOCTL_SET_MMU_ENTRY	TRUE: Enable MMU operation	TLB entry.	(Useless)	Enable VPE MMU operation.
VPE_IOCTL_SET_TLB_ENTRY	Component entry: 0~7 0 and 4: For packet or Y component 1 and 5: For U component 2 and 6: For V component 3 and 7: For destination image	(Useless)	(Useless)	Specified the component entry if page fault. <b>It has been handled by library</b>

**Return Value**

None.

**Example**

```
/* Setup hardware IP through IO control */

vpeIoctl(VPE_IOCTL_SET_MMU_ENTRY,
          TRUE, // MMU Enable
          (UINT32)_mmuSectionTable, // TLB Entry
          0x0);

vpeIoctl(VPE_IOCTL_HOST_OP,
          VPE_HOST_FRAME,
          VPE_OP_NORMAL,
          NULL);

vpeIoctl(VPE_IOCTL_SET_SRCBUF_ADDR,
          (UINT32)pi8Y, /* They are virtual address if MMU on */
          (UINT32)pi8U,
          (UINT32)pi8V);

vpeIoctl(VPE_IOCTL_SET_FMT,
          VPE_SRC_PLANAR_YUV420, /* Src Format */
```

```
VPE_DST_PACKET_RGB565,           /* Dst Format */

0);

vpeIoctl(VPE_IOCTL_SET_SRC_OFFSET,
0,      /* Src Left offset */
0,      /* Src right offset */
NULL);

vpeIoctl(VPE_IOCTL_SET_DST_OFFSET,
0,      /* Dst Left offset */
0,      /* Dst right offset */
NULL);

vpeIoctl(VPE_IOCTL_SET_SRC_DIMENSION,
2048,
1536,
NULL);

vpeIoctl(VPE_IOCTL_SET_DST_DIMENSION,
640,
480,
NULL);

vpeIoctl(VPE_IOCTL_SET_COLOR_RANGE,
FALSE,           /* Soure image is full range */
FALSE,           /* Destination image is full range */
NULL);

vpeIoctl(VPE_IOCTL_SET_FILTER,
VPE_SCALE_BILINEAR,
NULL,
NULL);

vpeIoctl(VPE_IOCTL_SET_DST_OFFSET,
```

```
        0,           //left offset
        0,           //Right offset
        NULL);
vpeIoctl(VPE_IOCTL_SET_DSTBUF_ADDR,
          piDstAddr,
          NULL,
          NULL);
vpeIoctl(VPE_IOCTL_TRIGGER,
          NULL,
          NULL,
          NULL);
do{
    ERRCODE errcode;
    errcode = vpeIoctl(VPE_IOCTL_CHECK_TRIGGER,
                        NULL,
                        NULL,
                        NULL);
    if(errcode==0)
        break;
}while(1);
```

---

## 30.4. Error Code Table

Code Name	Value	Description
ERR_VPE_OPEN	0xFFFF1B01	VPE has been opened
ERR_VPE_CLOSE	0xFFFF1002	VPE has been closed.
ERR_VPE_SRC_FMT	0xFFFF1003	Invalid source format
ERR_VPE_DST_FMT	0xFFFF1004	Invalid destination format
ERR_VPE_OP	0xFFFF1005	Invalid operation mode
ERR_VPE_IOCTL	0xFFFF1006	Invalid ioctl
E_VPE_INVALID_INT	0xFFFF1007	Invalid interrupt

# 31. VPOST Overview

## 31.1. VPOST Overview

**Display Interface Controller VPOST** (include LCD Controller & TV encoder Controller) is used to display the video/image data to LCD device or to generate the composite signal to the TV system. The LCD timing can be synchronize with TV (NTSC/PAL non-interlace timing) or set by the LCD timing control register. The video/image data source may be came from the frame buffer, color bar and register settings. The frame buffer is stored in system memory (SDRAM). The TV picture and LCD picture can display individual image source simultaneously when the timing is synchronized with TV timing.

### ***How to build the VPOST library***

Due to lot of panels supported in the VPOST library and some sample code links VPOST library with same name, it does not generate library file for each panel. User can open w55fa93\_vpost.h file to define corresponding panel to generate wanted panel library for usage and rename it as required. Below code shows how to generate VPOST library for HannStar HSD043I9W1.

```
#define HAVE_HANNSTAR_HSD043I9W1

//#define HAVE_HANNSTAR_HSD070IDW1          // 800x480

//#define HAVE_GOWORLD_GW8973

//#define HAVE_GOWORLD_GWMTF9406A

//#define HAVE_GOWORLD_GWMTF9360A

//#define __HAVE_GOWORLD_GWMTF9360A_MODIFY // wait be tested in detail


//#define HAVE_SHARP_LQ035Q1DH02

//#define HAVE_WINTEK_WMF3324


//#define HAVE_AMPIRE_800x600
```

```

//#define HAVE_AMPIRE_800x480

//#define HAVE_HIMAX_HX8346           // MPU 320x240

//#define HAVE_TVOUT_720x480
//#define HAVE_TVOUT_640x480
//#define HAVE_TVOUT_320x240

```

If User's panel is not listed in the header file, it will need to add related code by User or Nuvoton.

## 31.2. API Enumeration

Name	Value	Description
<b>E_DRVPOST_TIMING_TYPE</b>		
eDRVPOST_SYNC_TV	0x0	LCD timing sync with TV
eDRVPOST_ASYNC_TV	0x1	LCD timing not sync with TV
<b>E_DRVPOST_IMAGE_SOURCE</b>		
eDRVPOST_RESERVED	0x0	Reserved for LC source
eDRVPOST_FRAME_BUFFER	0x1	LCD source from Frame buffer
eDRVPOST_REGISTER_SETTING	0x2	LCD source from Register setting color
eDRVPOST_COLOR_BAR	0x3	LCD source from internal color bar
<b>E_DRVPOST_IMAGE_SCALING</b>		
eDRVPOST_DUPLICATED	0x0	Duplicate for TV Line buffer scaling
eDRVPOST_INTERPOLATION	0x1	Interpolation for TV line buffer scaling
<b>E_DRVPOST_LCM_TYPE</b>		
eDRVPOST_HIGH_RESOLUTION_SYNC	0x0	High resolution LCD device type
eDRVPOST_SYNC	0x1	Sync-type TFT LCD
eDRVPOST_MPU	0x3	MPU-type LCD
<b>E_DRVPOST_MPU_TYPE</b>		

eDRVPOST_I80	0x0	80-series MPU interface
eDRVPOST_M68	0x1	68-series MPU interface
<b>E_DRVPOST_8BIT_SYNCLCM_INTERFACE</b>		
eDRVPOST_SRGB_YUV422	0x0	YUV422(CCIR601) for 8bit LCD data interface
eDRVPOST_SRGB_RGBDUMMY	0x1	RGB dummy serial for 8 bit LCD data interface
eDRVPOST_SRGB_CCIR656	0x2	CCIR656 for 8 bit LCD data interface
eDRVPOST_SRGB_RGBTHROUGH	0x3	Serial RGB for 8 bit LCD data interface
<b>E_DRVPOST_CCIR656_MODE</b>		
eDRVPOST_CCIR656_360	0x0	720Y 360CbCr mode for CCIR656 horizontal active width
eDRVPOST_CCIR656_320	0x1	640Y 320CbCr mode for CCIR656 horizontal active width
<b>E_DRVPOST_ENDIAN</b>		
eDRVPOST_YUV_BIG_ENDIAN	0x0	Big Endian for YCbCr
eDRVPOST_YUV_LITTLE_ENDIAN	0x1	Little Endian for YCbCr
<b>E_DRVPOST_SERAIL_SYNCLCM_COLOR_ORDER</b>		
eDRVPOST_SRGB_RGB	0x0	Data in RGB order
eDRVPOST_SRGB_BGR	0x1	Data in BGR order
eDRVPOST_SRGB_GBR	0x2	Data in GBR order
eDRVPOST_SRGB_RBGR	0x3	Data in RBG order
<b>E_DRVPOST_PARALLEL_SYNCLCM_INTERFACE</b>		
eDRVPOST_PRGB_16BITS	0x0	16 pin parallel RGB data bus
eDRVPOST_PRGB_18BITS	0x1	18 pin parallel RGB data bus
eDRVPOST_PRGB_24BITS	0x2	24 pin parallel RGB data bus
<b>E_DRVPOST_SYNCLCM_DATABUS</b>		
eDRVPOST_SYNC_8BITS	0x0	8 bit sync-type LCD
eDRVPOST_SYNC_9BITS	0x1	9 bit sync-type LCD
eDRVPOST_SYNC_16BITS	0x2	16 bit sync-type LCD
eDRVPOST_SYNC_18BITS	0x3	18 bit sync-type LCD
eDRVPOST_SYNC_24BITS	0x4	24 bit sync-type LCD
<b>E_DRVPOST_MPULCM_DATABUS</b>		
eDRVPOST_MPUP8_8	0x0	Transfer in 8-8 format for 16 bit color in 8 bit bus

		width
eDRVPOST_MPU_2_8_8	0x1	Transfer in 2-8-8 format for 18 bit color in 8 bit bus width
eDRVPOST_MPU_6_6_6	0x2	Transfer in 6-6-6 format for 18 bit color in 8 bit bus width
eDRVPOST_MPU_8_8_8	0x3	Transfer in 8-8-8 format for 24 bit color in 8 bit bus width
eDRVPOST_MPU_9_9	0x4	Transfer in 9-9 format for 18 bit color in 9 bit bus width
eDRVPOST_MPU_16	0x5	Transfer in 16 format for 16 bit color in 16 bit bus width
eDRVPOST_MPU_16_2	0x6	Transfer in 16-2 format for 18 bit color in 16 bit bus width
eDRVPOST_MPU_2_16	0x7	Transfer in 2-16 format for 18 bit color in 16 bit bus width
eDRVPOST_MPU_16_8	0x8	Transfer in 16-8 format for 24 bit color in 16 bit bus width
eDRVPOST_MPU_18	0x9	Transfer in 18 format for 18 bit color in 18 bit bus width
eDRVPOST_MPU_18_6	0xA	Transfer in 18-6 format for 124 bit color in 18 bit bus width
eDRVPOST_MPU_24	0xB	Transfer in 24 format for 24 bit color in 24 bit bus width
<b>E_DRVPOST_FRAME_DATA_TYPE</b>		
eDRVPOST_FRAME_RGB555	0x0	RGB555 Frame buffer data format
eDRVPOST_FRAME_RGB565	0x1	RGB565 Frame buffer data format
eDRVPOST_FRAME_RGBX888	0x2	RGB_Dummy888 Frame buffer data format
eDRVPOST_FRAME_RGB888X	0x3	RGB888_Dummy Frame buffer data format
eDRVPOST_FRAME_CBYCRY	0x4	Cb0Y0Cr0Y1 Frame buffer data format
eDRVPOST_FRAME_YCBYCR	0x5	Y0Cb0Y1Cr0 Frame buffer data format
eDRVPOST_FRAME_CRYCBY	0x6	Cr0Y0Cb0Y1 Frame buffer data format
eDRVPOST_FRAME_YCRYCB	0x7	Y0Cr0Y1Cb0 Frame buffer data format

E_DRVPOST_DATABUS			
eDRVPOST_DATA_8BITS	0x0	8 bits data bus	
eDRVPOST_DATA_9BITS	0x1	9 bits data bus	
eDRVPOST_DATA_16BITS	0x2	16 bits data bus	
eDRVPOST_DATA_18BITS	0x3	18 bits data bus	
eDRVPOST_DATA_24BITS	0x4	24 bits data bus	

### 31.3. API Structure

*Table 31-1: LCDFORMATEX structure*

Field	Type	Description
ucVASrcFormat	UINT32	User input Display source format
nScreenWidth	UINT32	Driver output LCD width
nScreenHeight	UINT32	Driver output LCD height
nFrameBufferSize	UINT32	Driver output Frame buffer size
ucROT90	UINT8	Rotate 90 degree or not

*Table 31-2: S\_DRVPOST\_SYNCLCM\_HTIMING structure*

Field	Type	Description
u8PulseWidth	UINT8	Horizontal sync pulse width
u8BackPorch	UINT8	Horizontal back porch
u8FrontPorch	UINT8	Horizontal front porch

*Table 31-3: S\_DRVPOST\_SYNCLCM\_VTIMING structure*

Field	Type	Description
u8PulseWidth	UINT8	Vertical sync pulse width
u8BackPorch	UINT8	Vertical back porch
u8FrontPorch	UINT8	Vertical front porch

*Table 31-4: S\_DRVPOST\_SYNCLCM\_WINDOW structure*

Field	Type	Description
u16ClockPerLine	UINT16	Specify the number of pixel clock in each line or row of screen

u16LinePerPanel	UINT16	Specify the number of active lines per screen
u16PixelPerLine	UINT16	Specify the number of pixel in each line or row of screen

*Table 3I-5: S\_DRVPOST\_SYNCLCM\_POLARITY structure*

Field	Type	Description
bIsVsyncActiveLow	BOOL	Vsync polarity
bIsHsyncActiveLow	BOOL	Hsync polarity
bIsVDenActiveLow	BOOL	VDEN polarity
bIsDClockRisingEdge	BOOL	Clock polarity

*Table 3I-6: S\_DRVPOST\_MPULCM\_WINDOW structure*

Field	Type	Description
u16LinePerPanel	BOOL	Specify the number of active lines per screen
u16PixelPerLine	BOOL	Specify the number of pixel in each line or row of screen

*Table 3I-7: S\_DRVPOST\_MPULCM\_WINDOW structure*

Field	Type	Description
u8CSnF2DCt	UINT8	CSn fall edge to Data change clock counter
u8WRnR2CSnRt	UINT8	WRn rising edge to CSn rising clock counter
u8WRnLWt	UINT8	WR Low pulse clock counter
u8CSnF2WRnFt	UINT8	Csn fall edge To WR falling edge clock counter

*Table 3I-8: S\_DRVPOST\_MPULCM\_TIMING structure*

Field	Type	Description
bIsSyncWithTV	BOOL	MPU timing sync with TV
bIsVsyncSignalOut	BOOL	Specify MPU FrameMark pin as input or output pin
bIsFrameMarkSignallIn	BOOL	Frame Mark detection disable or enable
eSource	E_DRVPOST_IMAGE_SOURCE	Specify the image source
eType	E_DRVPOST_LCM_TYPE	Specify the LCM type
eMPUType	E_DRVPOST_MPUM_TYPE	Specify the MPU type
eBus	E_DRVPOST_MPULCM_DATABUS	Specify the MPU data bus
psWindow	S_DRVPOST_MPULCM_WINDOW*	Specify MPU window

psTiming	S_DRVPOST_MPULCM_TIMING*	Specify MPU timing
----------	--------------------------	--------------------

---

## 31.4. API Functions

### ***vpostGetFrameBuffer***

#### **Synopsis**

```
void *vpostGetFrameBuffer (void);
```

#### **Description**

Get the display frame buffer address

#### **Parameter**

None

#### **Return Value**

Display frame buffer address.

#### **Example**

None.

### ***vpostSetFrameBuffer***

#### **Synopsis**

```
void vpostSetFrameBuffer (
    UINT32 pFramebuf
);
```

#### **Description**

Set the display frame buffer address

#### **Parameter**

UINT32 pFramebuf

Given frame buffer address

#### **Return Value**

None.

**Example**

None.

***vpostLCMInit*****Synopsis**

```
INT32  
vpostLCMInit (  
    PLCDFORMATEX plcdformatex,  
    UINT32 *pFramebuf  
) ;
```

**Description**

Initialize the VPOST display device

**Parameter****plcdformatex [in]**

Input the lcd format information to initialize.

**pFramebuf [in]**

Input the frame buffer address

**Return Value**

Successful: Success

ERRCODE: Error

**Example**

```
__align(32) UINT8 Vpost_Frame[480*272*2];  
  
lcdFormat.ucVASrcFormat = DRVVPOST_FRAME_RGB565;  
  
lcdFormat.nScreenWidth = 480;  
  
lcdFormat.nScreenHeight = 272;  
  
vpostLCMInit(&lcdFormat, (UINT32*)Vpost_Frame);
```

***vpostLCMDeinit*****Synopsis**

INT32

vpostLCMDeinit (void);

#### Description

The function will stop VPOST operation and turn off VPOST clock.

#### Parameter

None

#### Return Value

Successful: Success

ERRCODE: Error

#### Example

None.

---

## 31.5. Error Code Table

Code Name	Value	Description
ERR_NULL_BUF	0xFFFF06004	memory location error
ERR_NO_DEVICE	0xFFFF06005	No device error
ERR_BAD_PARAMETER	0xFFFF06006	Bad parameter error
ERR_POWER_STATE	0xFFFF06007	Power state control error

## 32. Revision History

Version	Date	Description
V1.00.001	May 8, 2013	<ul style="list-style-type: none"><li>• Created</li></ul>

**Important Notice**

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.