

## N9H20 Non-OS Library Reference Guide

### Document Information

<b>Abstract</b>	Introduce Non-OS Library for the N9H20 series microprocessor (MPU).
<b>Apply to</b>	N9H20 series

*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 microcontroller and microprocessor 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</b>	<b>ADC LIBRARY .....</b>	<b>6</b>
1.1	Overview .....	6
<b>2</b>	<b>AVI LIBRARY .....</b>	<b>16</b>
2.1	Video render .....	16
2.2	How to use AVI player library .....	16
2.3	AVI player user callback.....	16
2.4	AVI playback information.....	16
2.5	AVI Library APIs Specification .....	16
2.6	Functions .....	18
2.7	Font Library Overview .....	23
2.8	Font Library APIs Specification .....	23
<b>3</b>	<b>BLT LIBRARY .....</b>	<b>30</b>
3.1	Overview .....	30
3.2	Feature .....	30
3.3	Transformation Matrix .....	30
3.4	Amendment to User Transformation Matrix .....	31
3.5	Color Transformation .....	35
3.6	Pixel Mapping.....	35
3.7	Palette.....	36
3.8	API Data Structure.....	37
3.9	API Function.....	40
<b>4</b>	<b>GNAND LIBRARY .....</b>	<b>55</b>
4.1	overview .....	55
4.2	GNAND Library Introduction .....	55
4.3	Programming Guide.....	55
<b>5</b>	<b>GPIO LIBRARY .....</b>	<b>65</b>
5.1	Overview .....	65
5.2	GPIO Library APIs Specification.....	65
<b>6</b>	<b>I2C LIBRARY .....</b>	<b>74</b>

6.1	Overview .....	74
6.2	I2C Library APIs Specification .....	74
<b>7</b>	<b>JPEG LIBRARY .....</b>	<b>79</b>
7.1	Overview .....	79
7.2	JPEG Library Overview .....	79
7.3	Programming Guide.....	79
7.4	Example code.....	98
<b>8</b>	<b>KPI LIBRARY .....</b>	<b>99</b>
8.1	Overview .....	99
8.2	KPI Library APIs Specification Functions.....	99
<b>9</b>	<b>NVTFAT LIBRARY .....</b>	<b>101</b>
9.1	Features .....	101
9.2	General Description .....	101
9.3	Initialize File System .....	102
9.4	Error Code .....	102
9.5	File Handle.....	102
9.6	Format Flash Memory Card .....	102
9.7	File Operations.....	103
9.8	File System Library APIs Specification .....	105
9.9	File/Directory Operations.....	115
9.10	Language Support.....	132
<b>10</b>	<b>PWM LIBRARY .....</b>	<b>139</b>
10.1	Overview .....	139
10.2	Programming Guide.....	139
10.3	PWM Library APIs Specification .....	145
<b>11</b>	<b>RTC LIBRARY .....</b>	<b>155</b>
11.1	Overview .....	155
11.2	Programming Guide.....	155
11.3	RTC Library APIs Specification .....	159
<b>12</b>	<b>SIC LIBRARY .....</b>	<b>165</b>
12.1	Overview .....	165

12.2 Storage Interface Controller Library .....	165
12.3 Programming Guide.....	165
12.4 SIC APIs Specification.....	166
12.5 SIC / SD APIs Specification .....	168
12.6 SIC / NAND APIs Specification.....	171
<b>13 SPI LIBRARY .....</b>	<b>178</b>
13.1 Overview .....	178
13.2 SPI Library APIs Specification .....	178
<b>14 SPI_SECUREIC LIBRARY .....</b>	<b>186</b>
14.1 Overview .....	186
14.2 API Functions.....	186
<b>15 SPU LIBRARY .....</b>	<b>192</b>
15.1 Overview .....	192
15.2 SPU Library APIs Specification .....	192
<b>16 I2S LIBRARY.....</b>	<b>197</b>
16.1 overview .....	197
16.2 APIs Specification .....	197
16.3 Functions .....	197
<b>17 SYSTEM LIBRARY .....</b>	<b>202</b>
17.1 Overview .....	202
17.2 System Library APIs Specification.....	202
17.3 Timer Functions.....	202
17.4 UART Function.....	211
17.5 AIC Functions .....	218
17.6 Cache Function.....	224
17.7 Clock Control Function .....	227
17.8 Power management Function .....	233
17.9 Error Code Table .....	235
<b>18 UDC LIBRARY.....</b>	<b>236</b>
18.1 overview .....	236
18.2 Programming Guide.....	236

18.3 USB Device (UDC) APIs Specification .....	240
18.4 Mass Storage Class (MSCD) APIs Specification.....	243
<b>19 USB CORE LIBRARY .....</b>	<b>248</b>
19.1 Overview .....	248
19.2 Data Structures .....	248
19.3 Descriptor Structures .....	250
19.4 Data Transfer .....	258
19.5 USB Core Library APIs Specification .....	267
<b>20 VPOST LIBRARY .....</b>	<b>279</b>
20.1 Overview .....	279
20.2 VPOST APIs Specification .....	279
20.3 Enumeration .....	280
20.4 Functions .....	285
20.5 Error Code Table .....	291
<b>21 EDMA LIBRARY .....</b>	<b>292</b>
21.1 Overview .....	292
21.2 Programming Guide.....	292
21.3 APIs Specification Functions .....	295
21.4 Error Code Table .....	303

# 1 ADC Library

## 1.1 Overview

The N9H20 ADC library provides a set of APIs to report the X and Y-axis coordinate, battery voltage and audio recording. With these APIs, user can read the position that was touched in touch panel, or get the current battery voltage, or record mono signed 16 bits PCM data. Please note, position detection on touch panel and voltage detection will be invalid if enable audio recording.

These libraries are created by using Keil uVision IDE. Therefore, they only can be used in Keil environment.

### 1.1.1 ADC Library APIs Specification

#### adc\_init

##### Synopsis

```
void adc_init(void);
```

##### Description

This function is used to initialize the ADC library.

##### Parameter

None

##### Return Value

None

##### Example

```
/* Initialize ADC library */  
adc_init();  
adc_open(ADC_TS_4WIRE, 320, 240);
```

#### adc\_open

##### Synopsis

```
int adc_open (unsigned char type, unsigned short hr, unsigned short vr)
```

##### Description

This function opens the ADC library and set the panel type.

##### Parameter

type      ADC\_TS\_4WIRE. N9H20 only support 4-wire touch panel.  
hr Touch screen horizontal resolution. It is for future using.  
vr Touch screen vertical resolution. . It is for future using.

##### Return Value

Return 0 on success, -1 for ADC not being initialize, or wrong parameter.

### Example

```
/* Initialize ADC library */
adc_init();
adc_open(ADC_TS_4WIRE, 320, 240);
```

## adc\_close

### Synopsis

void adc\_close(void)

### Description

Close the ADC library.

### Parameter

None

### Return Value

None

### Example

```
/* Close ADC library*/
adc_close();
```

## adc\_read

### Synopsis

int adc\_read(unsigned char mode, unsigned short \*x, unsigned short \*y)

### Description

This function is used to read the position that was touched in touch panel.

### Parameter

mode      ADC\_NONBLOCK or ADC\_BLOCK

x   x-axis coordinate.

y   y-axis coordinate.

Table 1-1: ADC read mode

Field name	Value	Description
ADC_NONBLOCK	0	Non-block the program
ADC_BLOCK	1	Block the program until touch panel pressed

### Return Value

None

### Example

```
/* Read the x and y-axis coordinate */
adc_init();
adc_open(ADC_TS_4WIRE, 320, 240);
while (1)
{
    if(adc_read(ADC_NONBLOCK, &x, &y))
        sysprintf("x = %d, y = %d\n", x, y);
}
```

## adc\_normalread

### Synopsis

UINT32 adc\_normalread(UINT32 u32Channel, PUINT16 pu16Data)

### Description

This function is used to read the battery voltage

### Parameter

u32Channel        2, 3 or 4. It depends on the hardware connects to which channel.

pu16Data    10 bits unsigned integer. It means the battery voltage.

### Return Value

Error code or Successful

### Example

```
/* Read the battery voltage from channel 2 */
adc_init();
adc_open(ADC_TS_4WIRE, 320, 240);
while(1)
{
    if(adc_normalread(2, &u16Vol)==Successful)
        sysprintf("Battery voltage = %x\n", u16Vol);
}
```

## adc\_setTouchScreen

### Synopsis

VOID\_setTouchScreen(E\_ADC\_TSC\_MODE eTscMode, INT32  
u32DleayCycle,  
BOOL blsPullup, BOOL bMAVFilter)



### Description

This function is used to set some properties about touch screen

### Parameter

eTscMode Setup ADC to operation in auto conversion mode or wait for trigger mode.

Field name	Value	Description
eADC_TSCREEN_NORMAL	0	The mode is not supported for touch screen
eADC_TSCREEN_SEMI	1	The mode is not supported now.
eADC_TSCREEN_AUTO	2	ADC works in auto conversion mode
eADC_TSCREEN_TRIG	3	ADC works in waiting for trigger mode

u32DleayCycle The clock count between ADC to converse one sample of X or Y coordinate.

bIsPullup The parameter is used to pull up internal resister or not

bMAVFilter Enable MAV filter or not.

### Return Value

None

### Example

```
/* Power down wake up by touch panel event */
while(1)
{
    adc_setTouchScreen(eADC_TSCREEN_TRIG, /*E_DRVADC_TSC_MODE*/
                      180, /* Delay cycle each conversion
*/
                      TRUE, /* BOOL bIsPullup */
                      FALSE); /* Disable bMAVFilter */

    sysDelay(50);
    sysprintf("\n");
    if(Entry_PowerDown()!=Successful)/* system enter standby
mode*/
    {
        sysprintf("Memory copy error \n");
        break;
    }
    outp32(0xb0000030, inp32(0xb0000030)| 0x40000000);

    /* Clear ADC wakeup status */
    sysprintf("Wake up!!! %d \n", u32Count);
    u32Count = u32Count +1;
}
```

## audio\_Open

### Synopsis

INT32 audio\_Open(E\_SYS\_SRC\_CLK eSrcClock, UINT32 u32ConvClock)

### Description

Open the Audio record function

### Parameter

eSrcClock The source of clock,

u32ConvClock The specified sample rate. It is only support 8000, 11025, 12000 and 16000 sample rate. The frequency of source clock needs to meet the specified sampling rate.

### Return Value

Error code or Successful

### Example

```
/* Set the sample rate */
U32SampleRate = 16000;
audio_Open(eSYS_APLL, u32SampleRate);
```

## adc\_enableInt

### Synopsis

UINT32 adc\_enableInt(E\_ADC\_INT eIntType)

### Description

This function enables ADC interrupt

### Parameter

eIntType The ADC interrupt type

Table 1-2: ADC interrupt type

Field name	Value	Description
eADC_ADC_INT	0	ADC interrupt
eADC_AUD_INT	1	Audio interrupt.
eADC_LVD_INT	2	Low voltage detector (LVD) interrupt
eADC_WT_INT	3	Waiting for trigger interrupt

### Return Value

Error code or Successful

### Example

```
/* Set the sample rate and enable audio interrupt */
```

```
audio_Open(eSYS_APLL, u32SampleRate);
adc_enableInt(eADC_AUD_INT);
```

## adc\_disableInt

### Synopsis

UINT32 adc\_disableInt(E\_ADC\_INT eIntType)

### Description

This function disables ADC interrupt

### Parameter

eIntType The ADC interrupt type

Table 1-3: ADC interrupt type

Field name	Value	Description
eADC_ADC_INT	0	ADC interrupt
eADC_AUD_INT	1	Audio interrupt.
eADC_LVD_INT	2	Low voltage detector (LVD) interrupt
eADC_WT_INT	3	Waiting for trigger interrupt

### Return Value

Error code or Successful

### Example

```
/* Set the sample rate and disable audio interrupt */
audio_Open(eSYS_APLL, u32SampleRate);
adc_disableInt(eADC_AUD_INT);
```

## adc\_installCallback

### Synopsis

UINT32 adc\_installCallback(E\_ADC\_INT eIntType, PFN\_ADC\_CALLBACK pfnCallback, PFN\_ADC\_CALLBACK\* pfnOldCallback)

### Description

Set the callback function and store the old callback function.

### Parameter

eIntType The ADC interrupt type

pfnCallback Function pointer to the callback function.

pfnOldCallback Function pointer to the old callback function

### Return Value

Error code or Successful

### Example

```
/* Set the sample rate and disable audio interrupt, set the callback
function for audio interrupt */
audio_Open(eSYS_APLL, u32SampleRate);
adc_disableInt(eADC_AUD_INT);
adc_installCallback(eADC_AUD_INT, pfnRecordCallback, &pfnOldCallback);
```

## audio\_StartRecord

### Synopsis

void audio\_StartRecord(void)

### Description

Start to record Audio data. This function only can be used in ADC\_RECORD mode.

### Parameter

None

### Return Value

None

### Example

```
/* Start record audio data */
adc_StartRecord()
```

## audio\_StopRecord

### Synopsis

void audio\_StopRecord(void)

### Description

Stop to record Audio data. This function only can be used in ADC\_RECORD mode.

### Parameter

None

### Return Value

None

### Example

```
/* Start and stop record audio data */
```

```
audio_StartRecord()
.
.
.
audio_StopRecord()
```

## audio\_SetAutoGainTiming

### Synopsis

```
void audio_SetAutoGainTiming(UINT32 u32Period, UINT32 u32Attack,
UINT32 u32Recovery, UINT32 u32Hold);
```

### Description

Set timings for auto gain control.

### Parameter

u32Period The unit for attack, recovery and hold time.

u32Attack The attack time to decrease the gain for output target level

u32Recovery The recovery time to increase the gain for output target level

u32Hold The hold time to keep current gain if current level has meet output target level

### Return Value

None

### Example

```
UINT32 u32Period, u32Attack, u32Recovery, u32Hold;
audio_GetAutoGainTiming(&u32Period, &u32Attack, &u32Recovery,
&u32Hold);
if(u32Period<128)
{
    u32Period = u32Period+16;
    audio_SetAutoGainTiming(u32Period, u32Attack, u32Recovery,
u32Hold);
}
```

## audio\_GetAutoGainTiming

### Synopsis

```
void audio_GetAutoGainTiming(PUINT32 pu32Period,PUINT32
pu32Attack,PUINT32 pu32Recovery,PUINT32 pu32Hold);
```

### Description

Get current timings for auto gain control.

#### Parameter

pu32Period	The unit for attack, recovery and hold time.
pu32Attack	The attack time to decrease the gain for output target level
pu32Recovery level	The recovery time to increase the gain for output target level
pu32Hold	The hold time to keep current gain if current level has meet output target level

#### Return Value

None

#### Example

```
UINT32 u32Period, u32Attack, u32Recovery, u32Hold;
    audio_GetAutoGainTiming(&u32Period, &u32Attack, &u32Recovery,
&u32Hold);
    if(u32Period<128)
    {
        u32Period = u32Period+16;
        audio_SetAutoGainTiming(u32Period, u32Attack, u32Recovery,
u32Hold);
    }
```

### audio\_SetAutoGainControl

#### Synopsis

```
void audio_SetAutoGainControl(BOOL blsEnable, UINT32 u32OutputLevel,
E_ADC_UPBAND eAdcUpBand,E_ADC_DOWNBAND eAdcDownBand);
```

#### Description

Set audio record output target level, ramp up and ramp down step if enables AGC function.

#### Parameter

blsEnable	Enable AGC or not.
u32OutputLevel	Output target level. The value from 0 ~ 15. Value 0 means - 28.5 db with each level 1.5 db. Value 15 means -6 db.
eAdcUpBand	Ramp up each step
eAdcDownBand	Ramp down each step

Table 1-4 eAdcUpBand - Ramp up

Field name	Value	Description
eADC_BAND_P0P5	0	Increase 0.5db each step
eADC_BAND_P0P75	1	Increase 0.75db each step

Table 1-5 eAdcDownBand - Ramp down

Field name	Value	Description
eADC_BAND_N0P5	0	Decrease 0.5db each step
eADC_BAND_N0P75	1	Decrease 0.75db each step

#### Return Value

None

#### Example

```
audio_SetAutoGainControl(TRUE, /* AGC enable */
                          13, /* Output target aszOTLArray[13] = -
9db */
                          eADC_BAND_P0P5,
                          eADC_BAND_N0P5);
```

#### Error Code Table

Code Name	Value	Description
E_DRVADC_INVALID_INT	0xFFFFB8E0	Invalid interrupt
E_DRVADC_INVALID_CHANNEL	0xFFFFB8E1	Invalid channel
E_DRVADC_INVALID_TIMING	0xFFFFB8E2	ADC is busy. It is only occurrence if ADC works in touch panel mode and battery detection in the same time,

## 2 AVI Library

### 2.1 Video render

N9H20 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 the 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.

### 2.2 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 the 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 **DRVVPOST\_FRAME\_RGB555**
- Direct RGB565 – VPOST should select **DRVVPOST\_FRAME\_RGB565**
- Direct RGB888 – VPOST should select **DRVVPOST\_FRAME\_RGBx888** or **DRVVPOST\_FRAME\_RGB888x**
- Direct YUV422 – VPOST should select **DRVVPOST\_FRAME\_CBYCRY** or **DRVVPOST\_FRAME\_YCBYCR** or **DRVVPOST\_FRAME\_CRYCBY** or **DRVVPOST\_FRAME\_YCRYCB**

The AVI playback function supports (x, y) coordinate that are the second and third argument of ***aviPlayFile()*** used to specify the render location on LCD now. Moreover, decoded image scales by 1/2 in horizontal and vertical direction if the decoded video width is larger than the panel width.

### 2.3 AVI player user callback

While playing an AVI move, 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 the insufficiency of CPU time.

### 2.4 AVI playback information

While playing an AVI move, 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.

### 2.5 AVI Library APIs Specification



## Enumeration

Name	Value	Description
JV_MODE_E		
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
AU_TYPE_E		
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
PLAY_CTRL_E		
PLAY_CTRL_FF	0x0	(Not supported)
PLAY_CTRL_FB	0x1	(Not supported)
PLAY_CTRL_FS	0x2	(Not supported)
PLAY_CTRL_PAUSE	0x3	Pause Playback
PLAY_CTRL_RESUME	0x4	Resume Playback
PLAY_CTRL_STOP	0x5	Stop Playback
PLAY_CTRL_SPEED	0x6	(Not supported)

## 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

Table 2-1: AVI\_INFO\_T structure

## 2.6 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 on the specified coordinate.

#### Parameter

##### suFileName [in]

The full path file name of input AVI file.

##### x [in]

The x-coordinate from left-up corner of AVI video render area.

##### y [in]

The y-coordinate from left-up corner of AVI video render area.

There are 3 cases for the x/y coordinate specified.

Case 1: x=0 and y=0

The AVI display data is aligned to center of panel.

Case 2: x<0 or y<0

The AVI display data is aligned to coordinate (0,0) on the left-up corner

Case 3: others

The AVI display data is aligned to coordinate (x,y).

**mode [in]**

Video render mode.

**cb [in]**

User application callback function.

#### Return Value

Successful: Success

ERRCODE: Error

#### Example

```
/*-----*/
/* Direct RGB565 AVI playback !!          */
/*-----*/
lcdformatex.ucVASrcFormat = DRVVPOST_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);  
aviGetFileInfo(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(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(0x0);
```

**aviSetLeftChannelVolume****Synopsis**

```
int  
aviSetLeftChannelVolume (  
int vol  
);
```

**Description**

Set the Left channel audio playback volume only.

**Parameter**

**vol [in]**

The audio volume

**Return Value**

Successful: Success

ERRCODE: Error

**Example**

```
// Set Left Channel as Mute  
aviSetPlayLeftChannelVolume(0x0);
```

**mfl\_get\_last\_buf****Synopsis**

```
char *
```

```
mfl_get_last_buf (void);
```

#### Description

Get the last JPEG decoded buffer address.

#### Parameter

None

#### Return Value

Last JPEG decoded buffer address

#### Example

```
// Get last Jpeg Decoded buffer address
JpgBuf = mfl_get_last_buf();
```

### 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
MFL_ERR_INFO_NA	0xFFFF81E0	media information is insuficient
MFL_ERR_MOVIE_PLAYING	0xFFFF81E4	movie is still in play
MFL_ERR_ULTRAM_TMPF	0xFFFF81E6	ultra merge file stream must use temp file

## 2.7 Font Library Overview

The N9H20 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 strings on N9H20 demo board or evaluation board. The library is a software solution. After updating the frame buffer, VPOST controller can show the content of panel or TV.

These libraries are created by using Keil uVision IDE. Therefore, they only can be used in Keil environment.

## 2.8 Font Library APIs Specification

### *InitFont*

#### Synopsis

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

#### Description

This function is used to initialize the font library, and get some information of font library.

#### Parameter

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

Field name	Data Type	Description
u32FontRectWidth	UINT32	Font width. Now it is fixed to 16
u32FontRectHeight	UINT32	Font height. Now it is fixed to 22
u32FontOffset	UINT32	Font Offset. Now it is fixed to 11
u32FontStep	UINT32	Font Step. Now it is fixed to 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 de-initialized.
u32FontFileSize	UINT32	Reserved
pu32FontFileTmp	UINT32	Reserved
pu32FontFile	UINT32	Pointer of font file
au16FontColor[3]	UINT16	RGB565 color au16FontColor[0]: Font background color au16FontColor[1]: Font color au16FontColor[2]: Border color

Table 2-2:Font Information

### 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. Refer to the Table 2-2:Font Information

u32x start x position.

u32y start y position.

pszString The specified string is written 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, "N9H20 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. Refer to the Table 2-2: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. Refer to the Table 2-2:Font Information

ptRect     Solid rectangle pointer

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
---------	--------	---------------------------------------

Table 2-3: Rectangle Information

### 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. Refer to the Table 2-2:Font Information

ptRect     Solid rectangle pointer. Refer to the Table 2-3: Rectangle Information

### 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 draws a hollow rectangle with the specified border width.

### Parameter

ptFont Font library information pointer. Refer to the Table 2-2:Font Information

ptRect Solid rectangle pointer. Refer to the Table 2-3: Rectangle Information.

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. Refer to the Table 2-2:Font Information
u16RGB565	RGB565 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 the current font color. The return value format is RGB565.

**Parameter**

ptFont     Font library information pointer. Refer to the Table 2-2:Font Information

**Return Value**

RGB565 format

**Example**

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

## 3 BLT Library

### 3.1 Overview

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

### 3.2 Feature

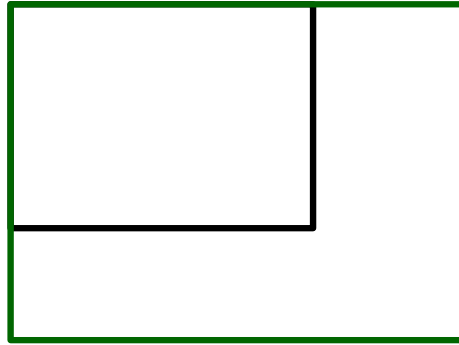
- Fill operation.
  - Fill color with alpha channel
- Blit operation
  - Transformation effects (Scaling, Rotation, Shearing, etc.) through 2x2 inverse transformation matrix.
  - Bitmap smoothing in bi-linear algorithm.
  - Tiling mode (for inversely mapped source pixels lying outside the boundaries 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

### 3.3 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.

#### 3.3.1 Scaling

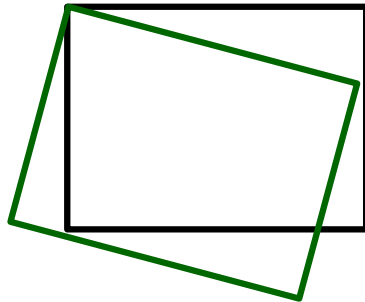
Resize the image by multiplying the location of each pixel by  $s_x$  on the x axis and  $s_y$  on the y axis.



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

### 3.3.2 Rotation

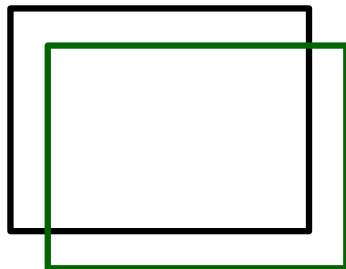
Rotate the image by an angle  $\theta$  clockwise.



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

### 3.3.3 Translation

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



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

## 3.4 Amendment to User Transformation Matrix

On mapping back from destination  $CS^1$  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  $180^\circ$ .

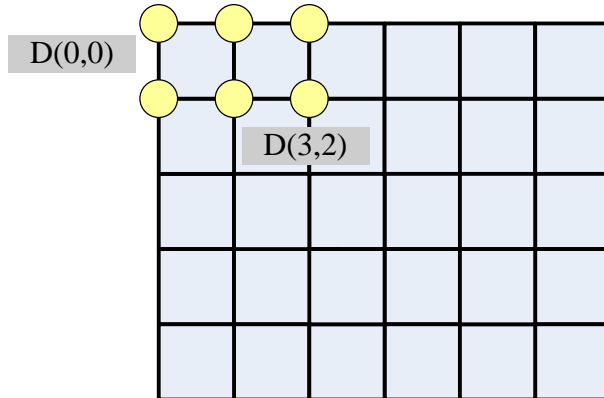


Figure 3-1: Top-left point as mapping point of destination pixel

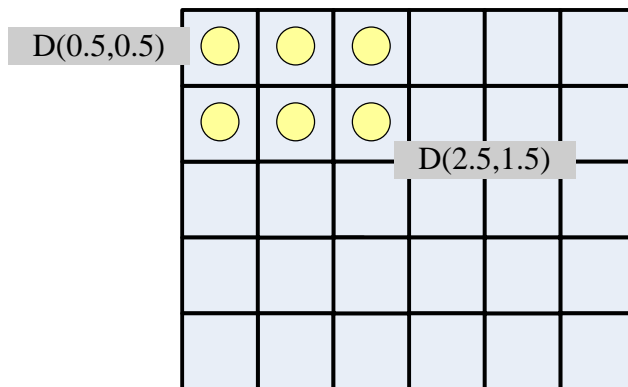


Figure 3-2: Center point as mapping point of destination pixel

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

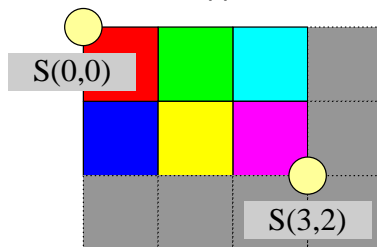


Figure 3-3: Source image

<sup>1</sup> Coordinate system.



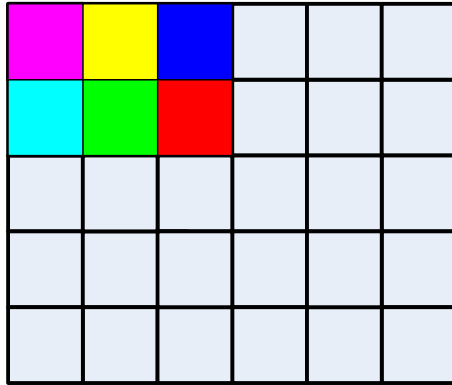


Figure 3-4: Final

2. First, just copy without any transformation effect and get [No transform](#) result.

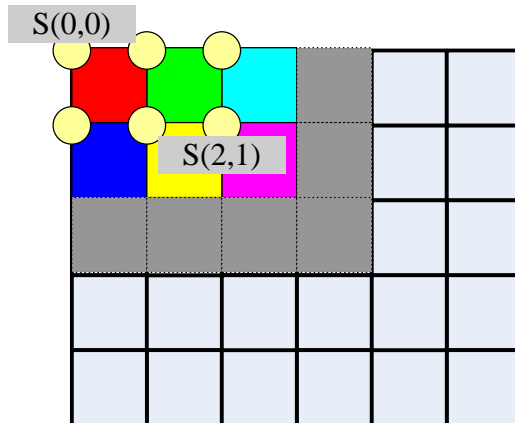


Figure 3-5: No transform

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

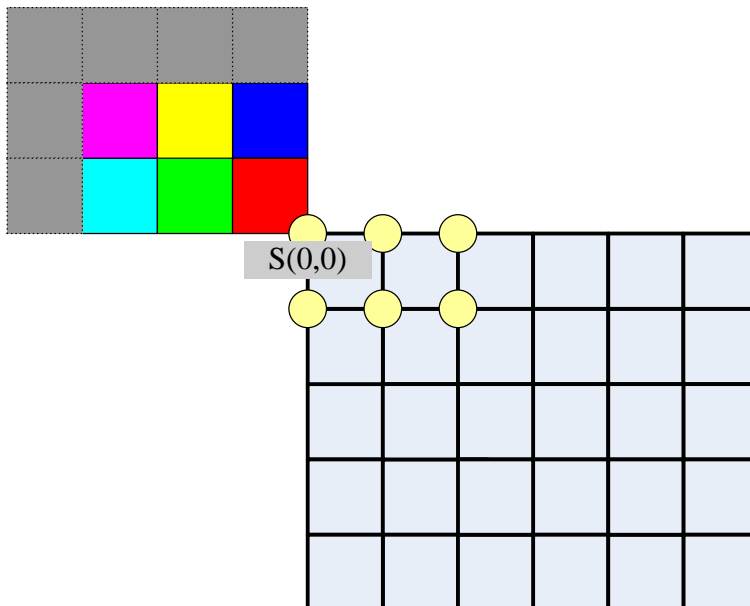


Figure 3-6: Rotate 180°

4. Translate 3 along x-axis and 2 along y-axis and get Rotate 180o + Translate (3, 2) result. But actually, we will get incorrect Rotate 180o + 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.

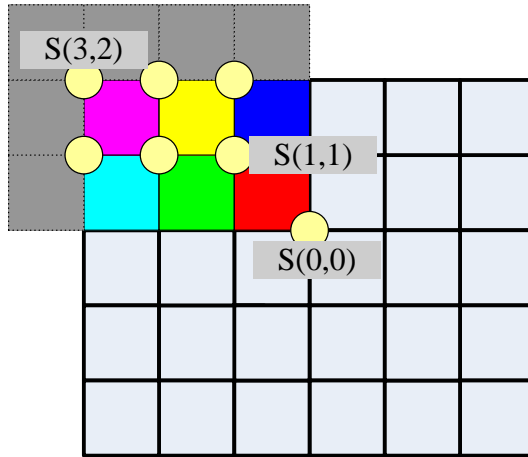


Figure 3-7: Rotate 180° + Translate (3, 2)

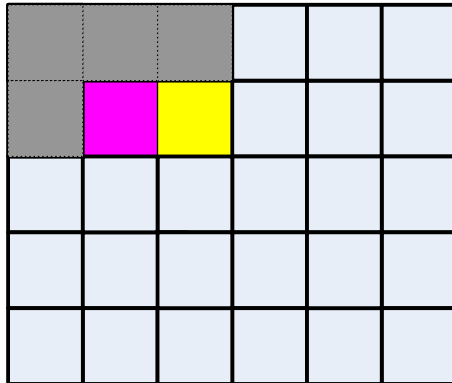


Figure 3-8: Rotate 180° + Translate (3, 2) (2)

5. To fix the issue, translate (-0.5, -0.5) to the end of all above transformations, and get Rotate 180o + 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.

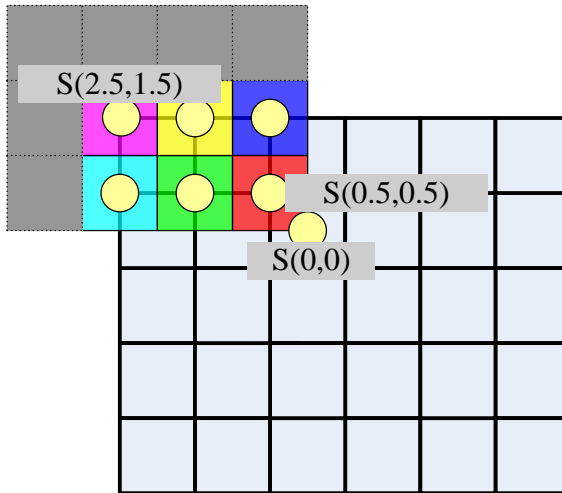


Figure 3-9: Rotate 180° + Translate (3, 2) + Translate (-0.5, -0.5)

### 3.5 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**

### 3.6 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.

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

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

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

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.

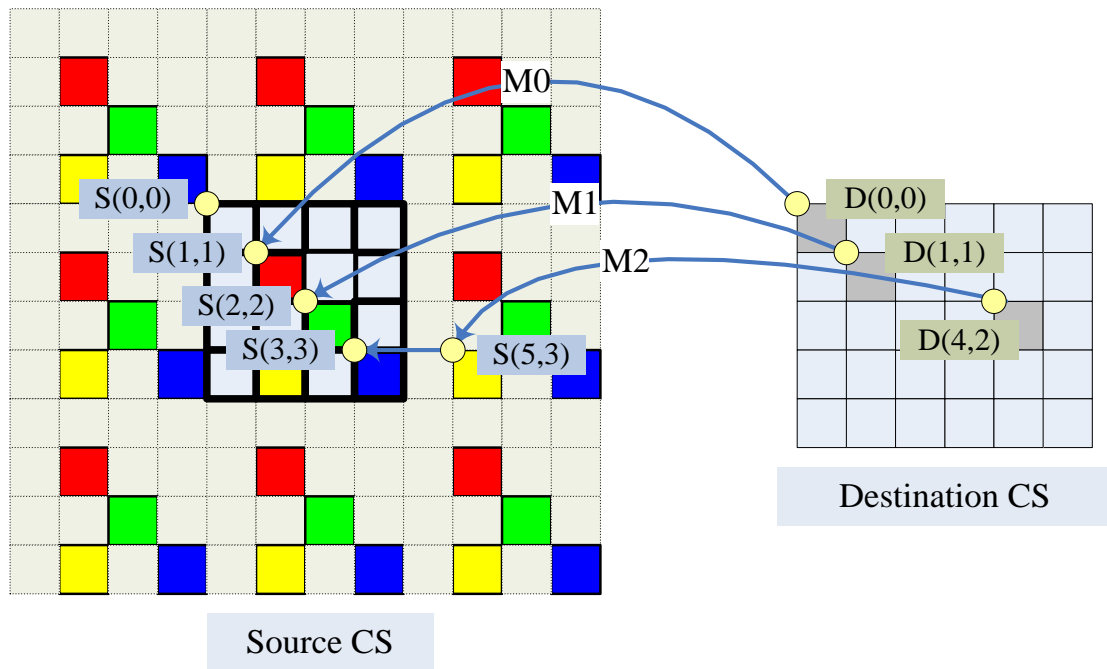


Figure 3-10: Mapping from destination CS to source CS

### 3.7 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.

### 3.7.1 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.

### 3.7.2 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

## 3.8 API Data Structure

### E\_BLT\_INT\_TYPE

Interrupt type.

Name	Value	Description
BLT_INT_CMPLT	1	Fill/Blit operation completed

### E\_DRVBLT\_FILLOP

Fill or Blit operation.

Name	Value	Description
eDRVBLT_DISABLE	0	Blit operation
eDRVBLT_ENABLE	1	Fill operation

### E\_DRVBLT\_REVEAL\_ALPHA

Premultiplied alpha or not for source format of ARGB8888

Name	Value	Description
eDRVBLT_EFFECTIVE	0	Premultiplied alpha
eDRVBLT_NO_EFFECTIVE	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
eDRVBLT_NONTRANSPARENCY	0	No per-pixel transparency in the source.
eDRVBLT_HASTRANSARENCY	1	Has per-pixel transparency in the source.
eDRVBLT_HASCOLORTRANSFORM	2	Apply color transformation formula.
eDRVBLT_HASALPHAONLY	4	If color transformation enabled, just apply the alpha-only formula.

## E\_DRVBLT\_BMPIXEL\_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
eDRVBLT_SRC_ARGB8888	1	RGB888/ARGB8888
eDRVBLT_SRC_RGB565	2	RGB565
eDRVBLT_SRC_1BPP	4	1-bit palette index
eDRVBLT_SRC_2BPP	8	2-bit palette index
eDRVBLT_SRC_4BPP	16	4-bit palette index
eDRVBLT_SRC_8BPP	32	8-bit palette index

## E\_DRVBLT\_DISPLAY\_FORMAT

Destination format for Fill/Blit operation.

Name	Value	Description
eDRVBLT_DEST_ARGB8888	1	ARGB8888
eDRVBLT_DEST_RGB565	2	RGB565
eDRVBLT_DEST_RGB555	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

## **3.9 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

[S\\_DRVBLT\\_MATRIX](#).

Transformation matrix as defined in

#### 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 [S\\_DRVBLT\\_MATRIX](#).

#### Return Value

None

## bltSetSrcFormat

### Synopsis

```
ERRCODE bltSetSrcFormat (E_DRVBLT_BMPIXEL_FORMAT eSrcFmt);
```

### Description

Set up source format.

### Parameter

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

### Return Value

E_SUCCESS	Success
ERR_BLT_INVALID_SRCFMT	Invalid source format

## bltGetSrcFormat

### Synopsis

```
E_DRVBLT_BMPIXEL_FORMAT bltGetSrcFormat(void);
```

### Description

Retrieve source format which has set up.

### Parameter

None

### Return Value

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

## bltSetDisplayFormat

### Synopsis

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

### Description

Set up destination format.

### Parameter

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

### 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 [E\\_BLT\\_INT\\_TYPE](#).

### Return Value

None

## bltDisableInt

### Synopsis

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

### Description

Disable specified interrupt type.

### Parameter

eIntType                      Interrupt type as defined in [E\\_BLT\\_INT\\_TYPE](#).

### 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 invoked 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                      Color multipliers of A, R, G, and B channels as defined in [S\\_DRVBLT\\_ARGB16](#).

#### **Return Value**

None

### **bltGetColorMultiplier**

#### **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 [E\\_DRVBLT\\_TRANSFORM\\_FLAG](#).

### Return Value

None





```
void bltSetColorPalette(UINT32 u32PaletteIdx, UINT32 u32Num,
S_DRVBLT_ARGB8 *psARGB);
```

#### Description

Set up palette's colors.

#### Parameter

u32PaletteIdx	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



### 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

### bltSetRGB565TransparentCtl

#### Synopsis

```
void bltSetRGB565TransparentCtl(BOOL bEnable);
```

#### Description

Enable color key or not.

#### Parameter

bEnable

TRUE

Enable color key

FALSE

Disable color key

#### Return Value

None

### bltGetRGB565TransparentCtl

#### Synopsis

```
BOOL bltGetRGB565TransparentCtl(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

**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   0x03	Invalid destination format

## 4 GNAND Library

### 4.1 overview

N9H20 Non-OS library consists of the 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), 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 N9H20 micro processor.

These libraries are created by using Keil uVision IDE. Therefore, they only can be used in Keil environment.

### 4.2 GNAND Library Introduction

In GNAND library, a NAND is though 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.

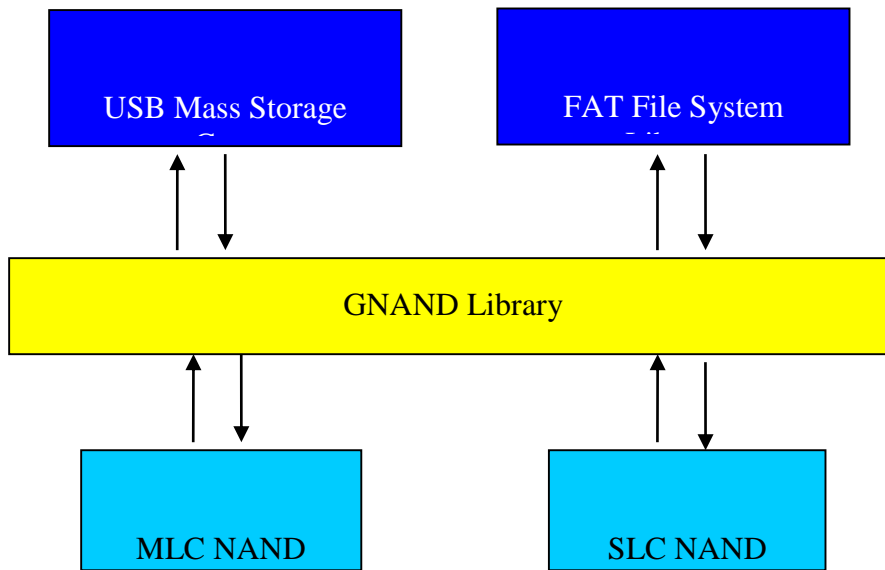
The Generic NAND (GNAND) library has the following features:

- 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)

### 4.3 Programming Guide

#### 4.3.1 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 the upper layer FAT file system library or USB mass storage device driver. The relationship between these component libraries was shown in the following picture:



### 4.3.2 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()**.

### 4.3.3 GNAND work with Nuvoton FAT Library

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

### 4.3.4 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 ndrvt
struct ndrvt
{
    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 (*ioctl1)(INT param1, INT param2, INT param3, INT param4);
    INT (*block_erase)(INT nPBlockAddr);
    INT (*chip_erase)(VOID);
}
    
```



```
VOID *next;
};
```

In **init(NDISK\_T \*NDInfo)** 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-

NDISK\_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	
write_page_in_seq	Must	Programming pages out of sequence within a block is prohibited or not. NAND_TYPE_PAGE_OUT_SEQ or NAND_TYPE_PAGE_IN_SEQ
reserved[59]	Ignore	
need2P2LN	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.

## 4.3.5 GNAND Library APIs Specification

### GNAND\_InitNAND

#### Synopsis

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

#### Description

Initialize a NAND disk.

#### Parameter

ndriver                      NAND driver function sets 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 a 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, 192000, 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 refers 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 refers to the NAND disk information that initiated by GNAND\_InitNAND().

nSectorNoRead 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

```
/* Read data from NAND disk sector 100 to sector 104 and store data to
buff on RAM */
__align (32) UINT8 buff [512*5];
GNAND_read(ptNDISK, 100, 5, 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 refers to the NAND disk information that initiated by GNAND\_InitNAND().

nSectorNoWrite the start sector number.

nSectorCnt Number of sectors to be written.

buff Memory buffer for writing data, which is 32 bytes aligned non-cacheable buffer

#### Return Value

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

#### Example

```
/* Write data from buff to NAND disk sector 100 to sector 104 */
__align (32) UINT8 buff [512*5];
...
GNAND_write(ptNDISK, 100, 5, buff);
```

### GNAND\_block\_erase

#### Synopsis

INT GNAND\_block\_erase (NDISK\_T \*ptNDisk, INT pba)

#### Description

Erase a physical block.

#### Parameter

ptNDisk The pointer refers 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

```
int status;

/* erase physical block 10 */
status = GNAND_block_erase(ptNDisk, 10);
if (status != 0)
{
/* handle error status */
}
```

### GNAND\_chip\_erase

#### Synopsis

INT GNAND\_chip\_erase (NDISK\_T \*ptNDisk)

#### Description

This function erases all blocks in NAND chip. All data in chip will be lost, including information for GNAND library.

**Parameter**

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

**Return Value**

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

**Example**

```
int status;  
  
/* erase whole NAND chip */  
status = GNAND_chip_erase(ptNDisk);  
if (status != 0)  
{  
    /* handle error status */  
}
```

**GNAND\_UnMountNandDisk****Synopsis**

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

**Description**

Unmount NAND disk from NVT FAT file system.

**Parameter**

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

**Return Value**

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

**Example**

```
int status;  
  
status = GNAND_UnMountNandDisk(ptNDisk);  
if (status != 0)  
{  
    /* handle error status */  
}
```

```
} 
```

### Example code

The example code tests the GNAND library, please refer to the SIC example code of BSP Non-OS.

### 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 read only 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 ID



## 5 GPIO Library

### 5.1 Overview

The GPIO library provides a set of APIs to control on-chip GPIO pins. This library depends on N9H20 System Library.

### 5.2 GPIO Library APIs Specification

#### gpio\_open

##### Synopsis

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

##### Description

This function enables GPIO port A, D, and E, which GPIO is not the default pad function. There is no need to call this function for GPIO port B and C.

##### Parameter

port      GPIO\_PORTA, GPIO\_PORTD, GPIO\_PORTE

##### Return Value

Return 0 on success. -1 for unknown port number

##### Example

```
/* Open port D*/  
gpio_open (GPIO_PORTD);
```

#### 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, and  
GPIO\_PORTE

num      pin number

##### Return Value

Return 0 on success. -1 for unknown port number

##### Example

```
/* Configure the pin 0 of port D 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, and  
GPIO\_PORTE

\*val      Return port value

### Return Value

Return 0 on success, -1 for unknown port number

### Example

```
/* Read PORT C 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 the second parameter.

### Parameter

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

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 on success, -1 for unknown port number

### Example

```
/* Set PORT C pin 1 to output mode, and pin 0 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 the second parameter.

### Parameter

port      GPIO\_PORTA, GPIO\_PORTB, GPIO\_PORTC, GPIO\_PORTD, and GPIO\_PORTE  
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 on success, -1 for unknown port number

### Example

```
/* Set PORT C pin 1 to output high, and pin 0 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 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, and GPIO\_PORTE  
mask      pin mask, each bit stands for one pin  
pull      Pull up resistor 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 PORT C pin 1 pull up resistor, and disable pin 0 pull up resistor */
```

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

## gpio\_setdebounce

### Synopsis

```
int gpio_setdebounce(unsigned int 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 int *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 int 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, and  
GPIO\_PORTE

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 pin 0 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, and  
GPIO\_PORTE

\*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_getsrcgrp (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, and  
GPIO\_PORTE

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 0 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, and  
GPIO\_PORTE

\*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, and  
GPIO\_PORTE

\*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, and  
GPIO\_PORTE

\*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, and  
GPIO\_PORTE



**Return Value**

Return 0 on success, -1 for parameter error

**Example**

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

## 6 I2C Library

### 6.1 Overview

This library provides APIs for programmers to access I2C slaves connecting with N9H20 I2C interfaces. The default clock frequency is configured at 100 kHz after i2cOpen() is called, programmers could use i2cClock() 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.

### 6.2 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 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 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 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 1 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 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
```

## i2cloctl

### Synopsis

INT32 i2cloctl(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	This command sets the clock frequency
I2C_IOC_SET_SUB_ADDRESS	Unsigned integer stores the sub address	Sub-address length	This command sets the sub-address and its length

### 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();
```

#### Error Code Table

Code Name	Value	Description
I2C_ERR_LOSTARBITRATION	0xFFFF1101	Arbitration lost during transmission
I2C_ERR_BUSBUSY	0xFFFF1102	I2C bus is busy
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

## 7 JPEG Library

### 7.1 Overview

N9H20 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 (NVTFAT), 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 N9H20 micro-processor.

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

### 7.2 JPEG Library Overview

This library is designed to make user application to use N9H20 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

### 7.3 Programming Guide

#### 7.3.1 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.

#### 7.3.2 JPEG Features

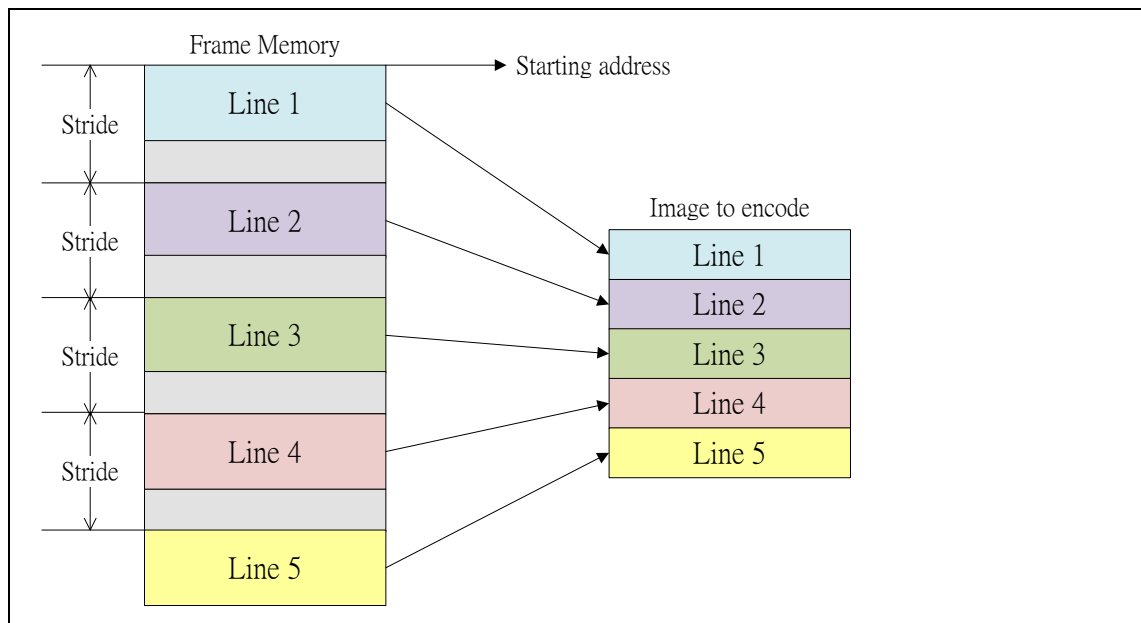
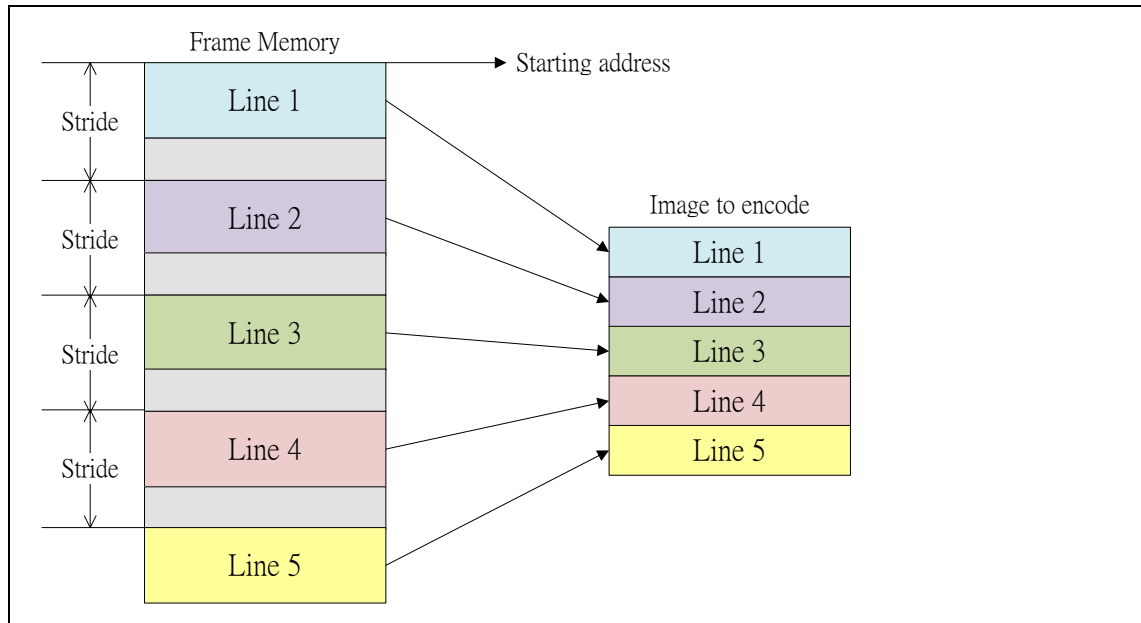
- Support to encode interleaved YCbCr 4:2:2/4:2:0 and gray-level (Y only) format image
- Support to decode interleaved YCbCr 4:4:4/4:2:2/4:2:0/4:1:1 and gray-level (Y only) format image
- Support to decode YCbCr 4:2:2 transpose format
- The encoded JPEG bit-stream format is fully compatible with JFIF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

### 7.3.3 JPEG Operation Control

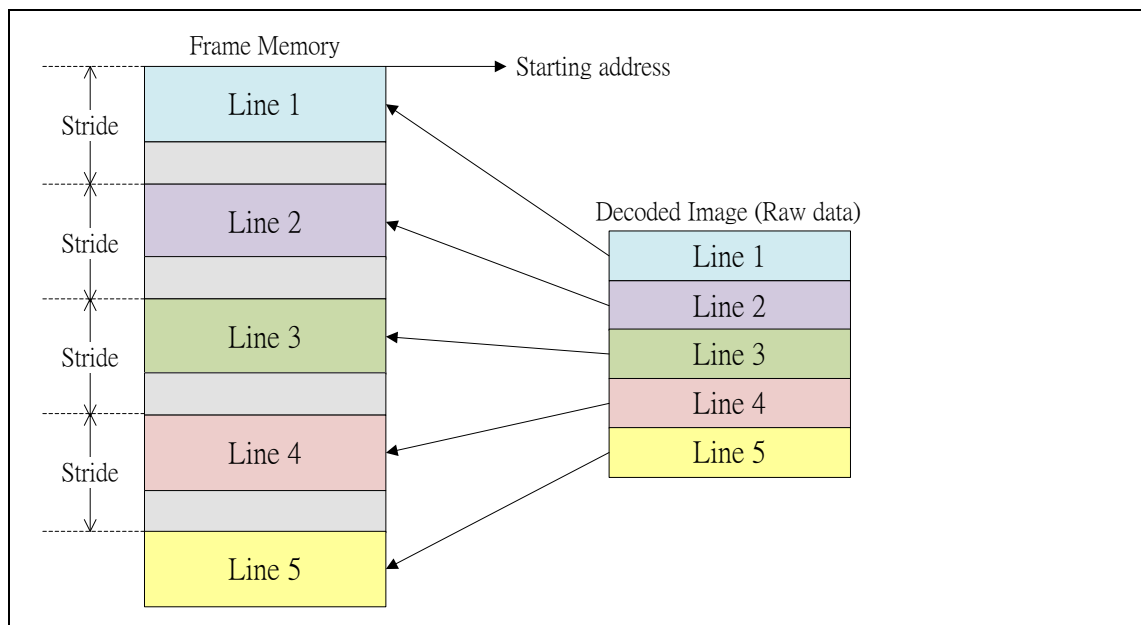
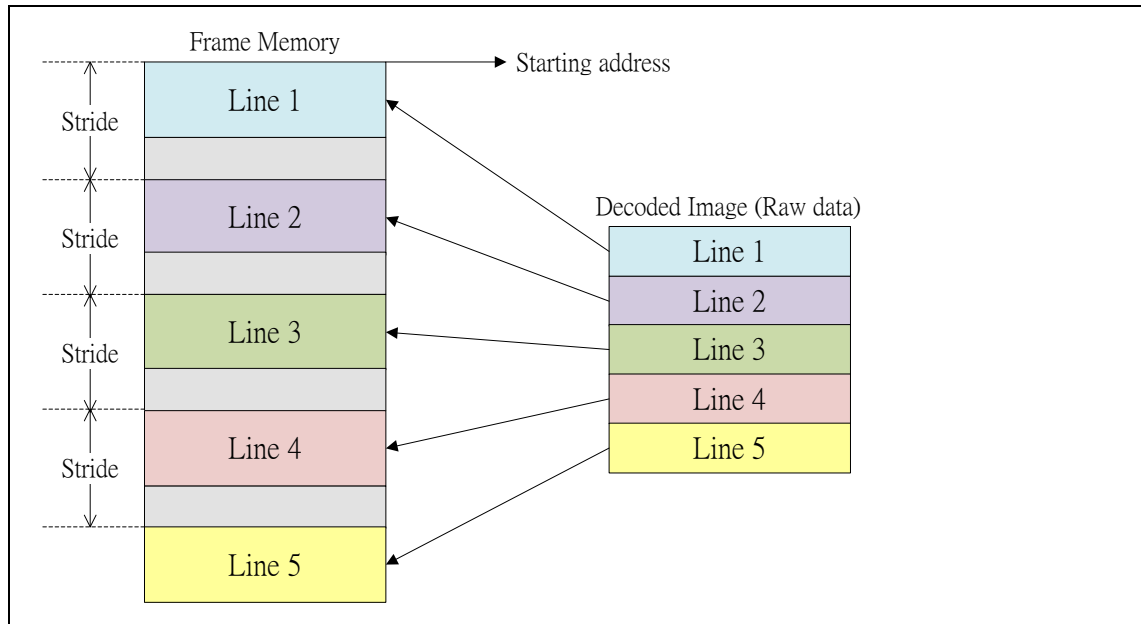
#### 7.3.3.1 Memory access

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

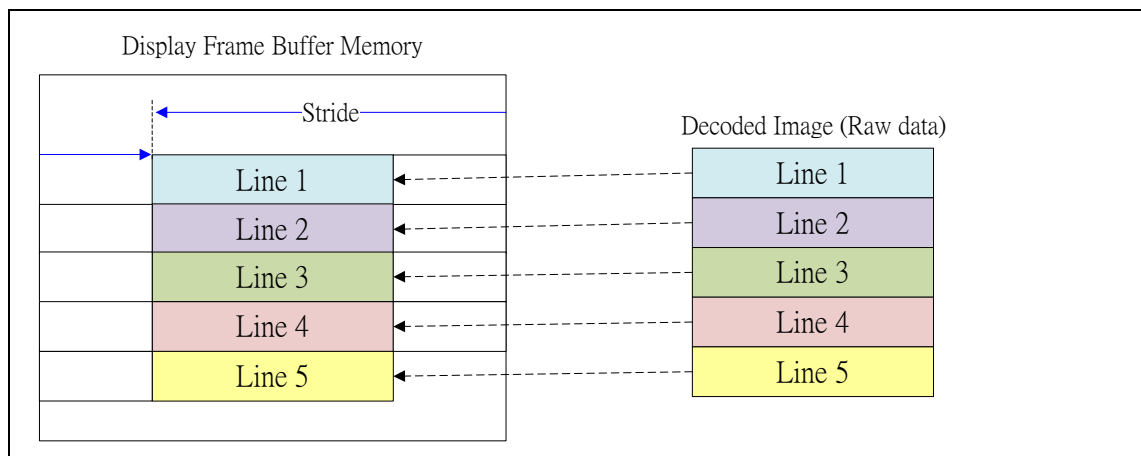
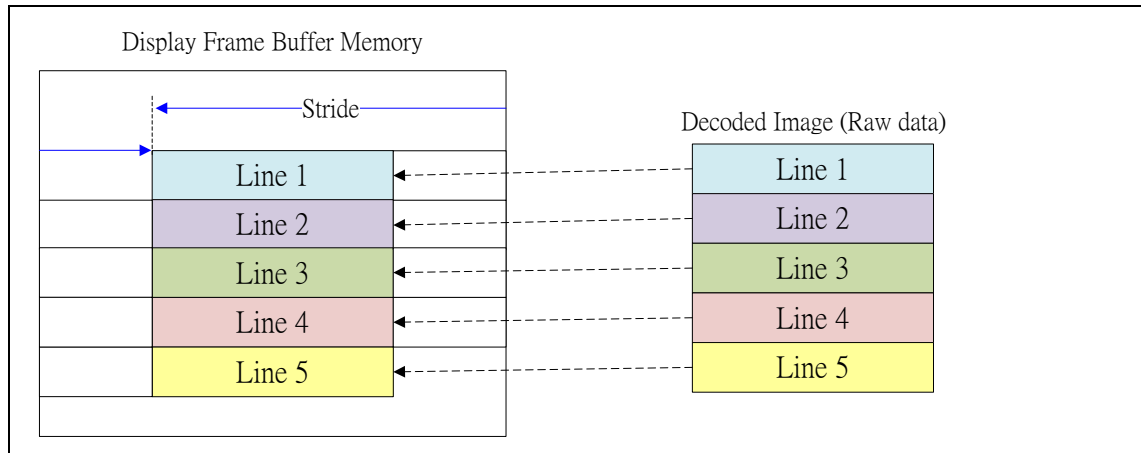


The following figure shows the decode mode to output the decoded raw data on the SDRAM.

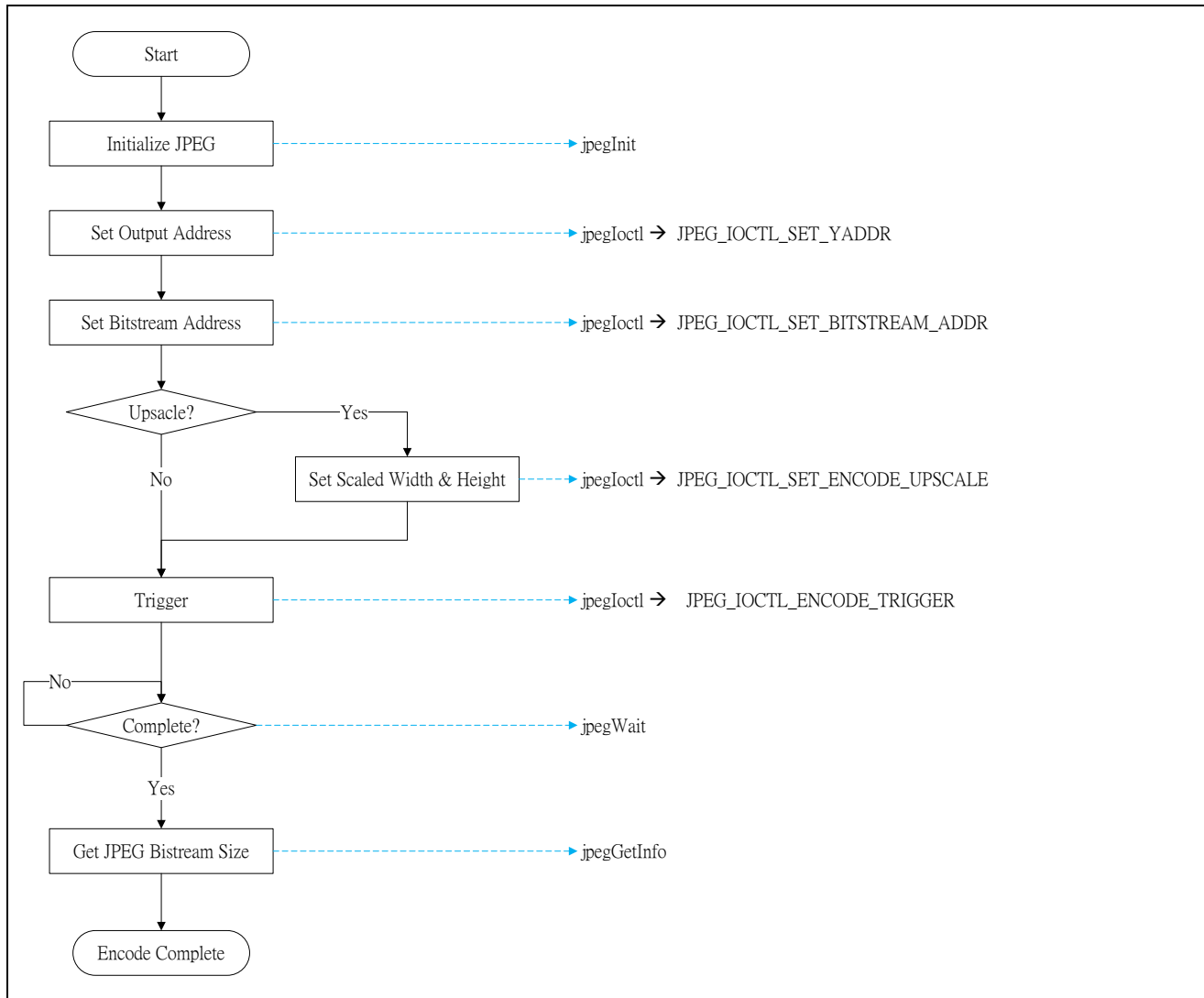




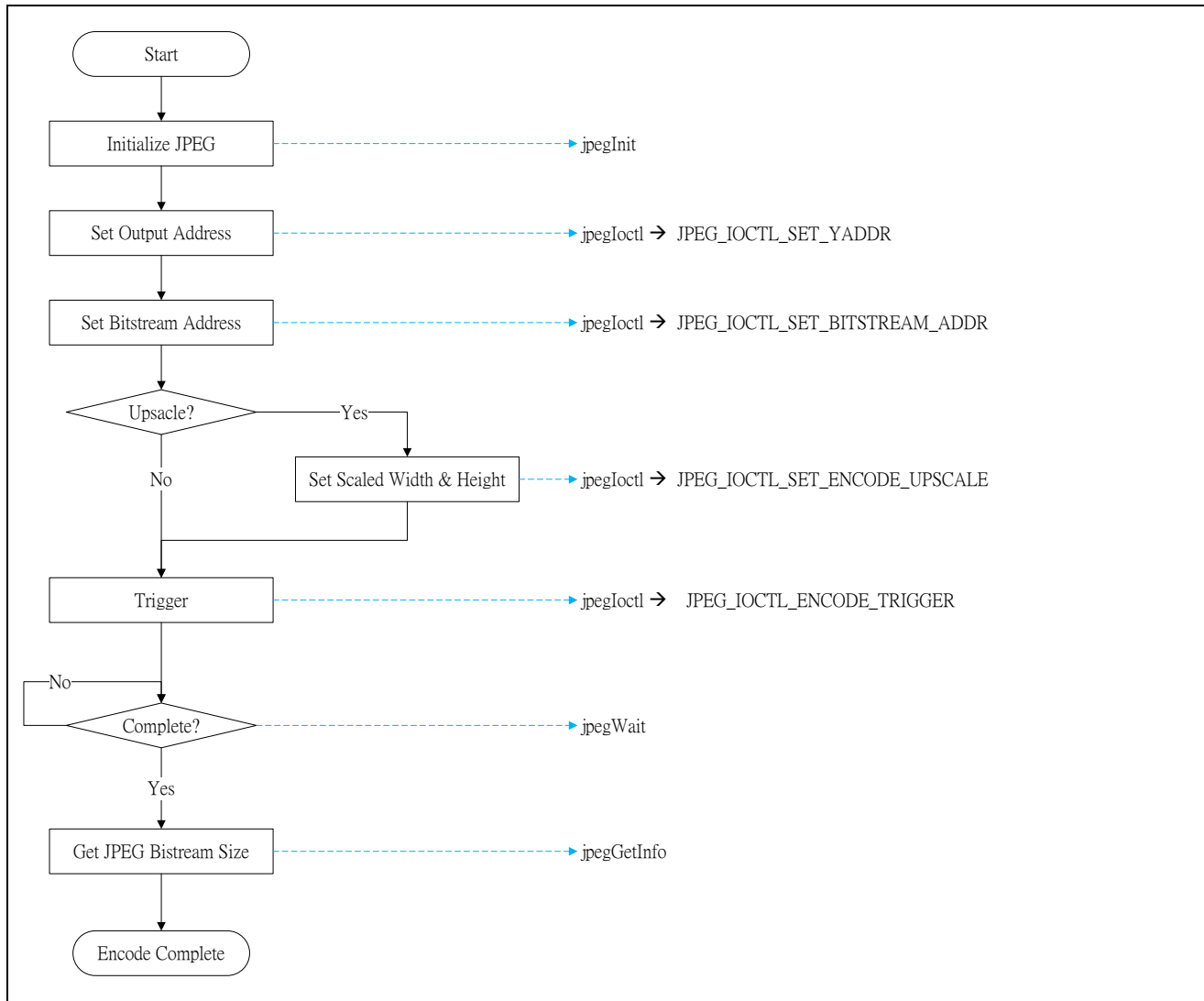
User can use stride function to output decoded image to any position on the Display Frame Buffer for Display. Following figure shows the decode mode with stride to output the decoded raw data on the Display Frame Buffer.

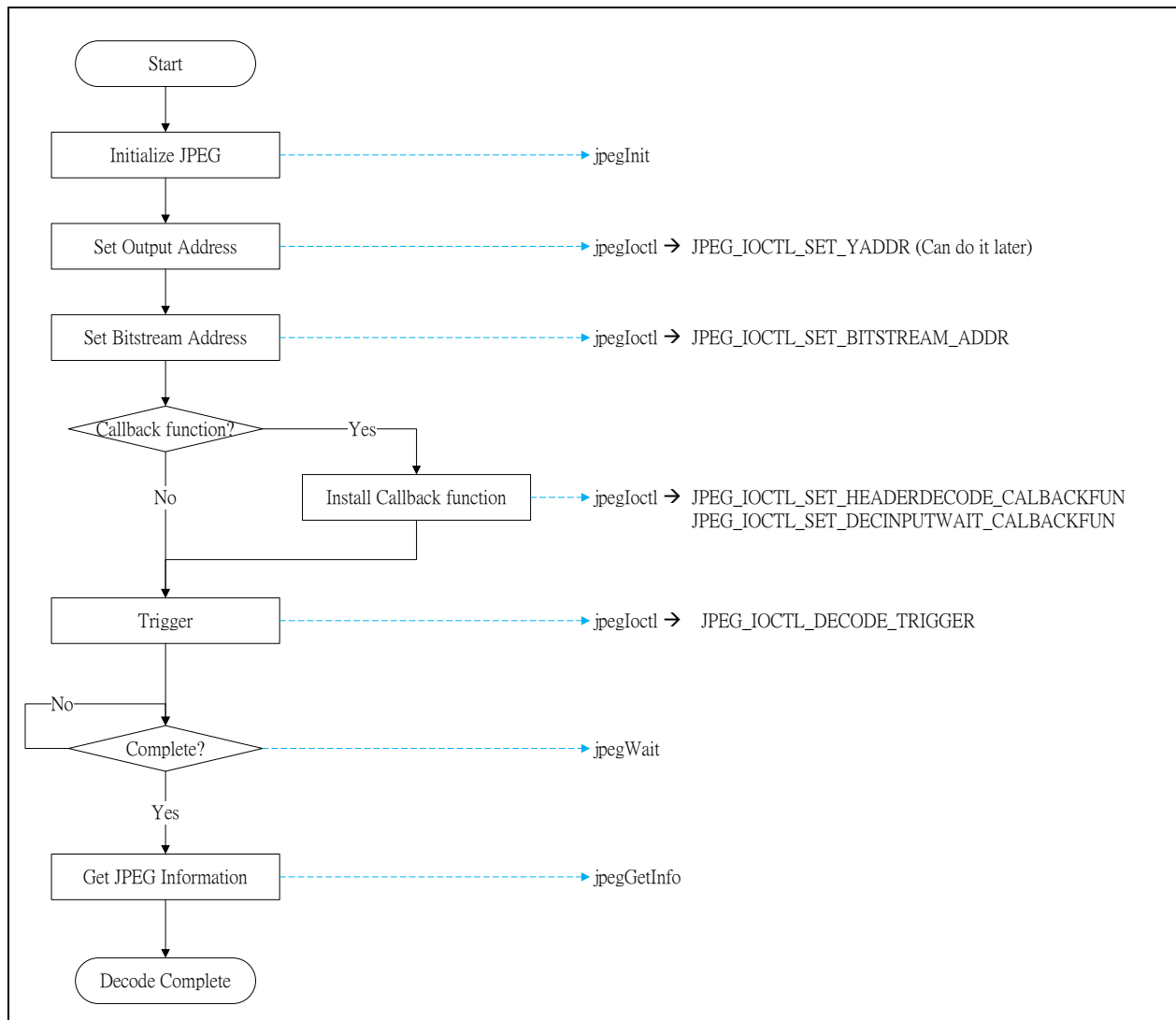


### 7.3.3.2 Encode operation flow



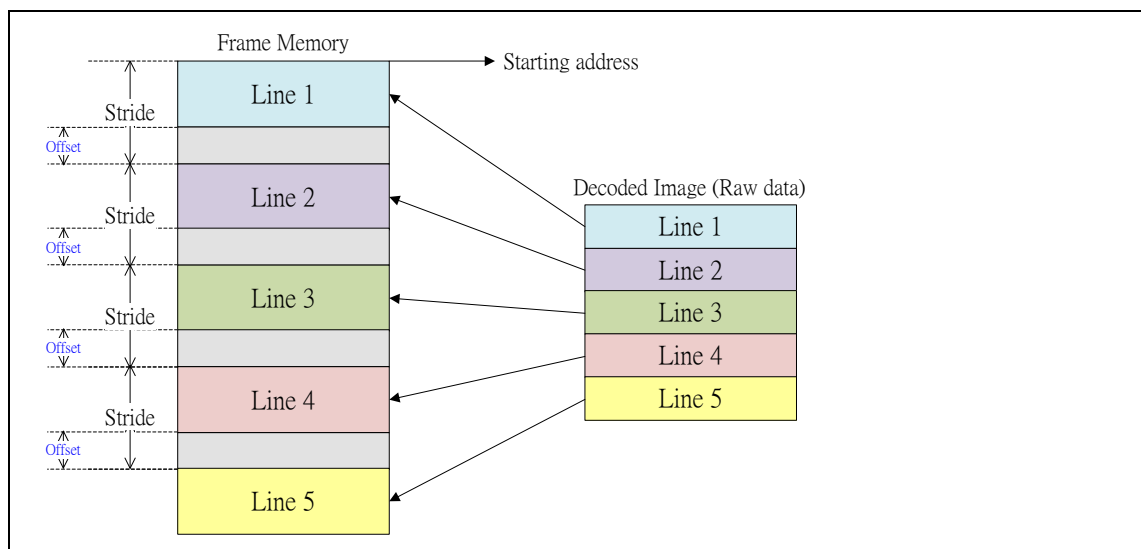
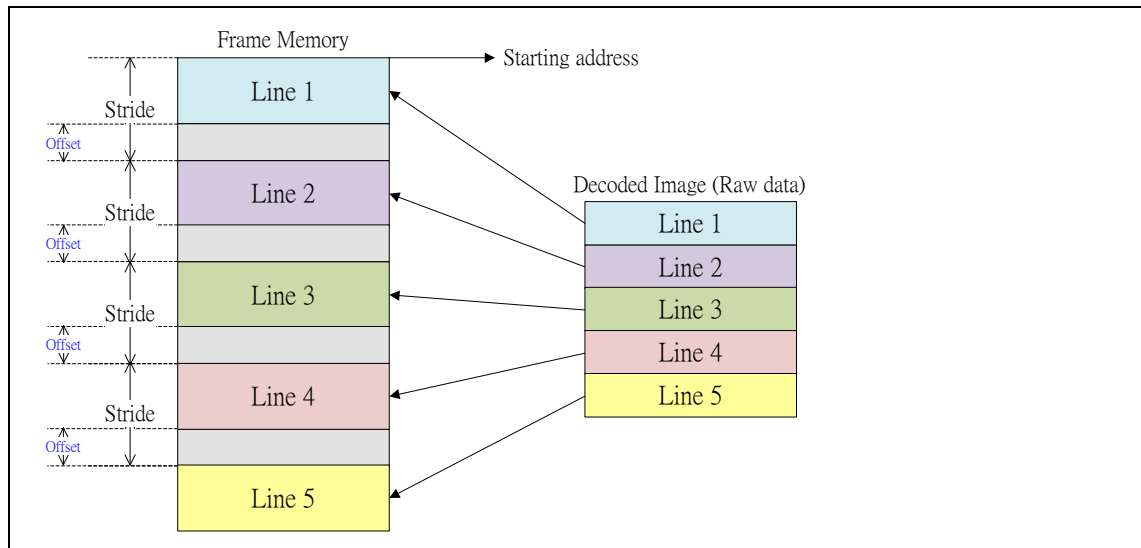
### 7.3.3.3 Decode operation flow





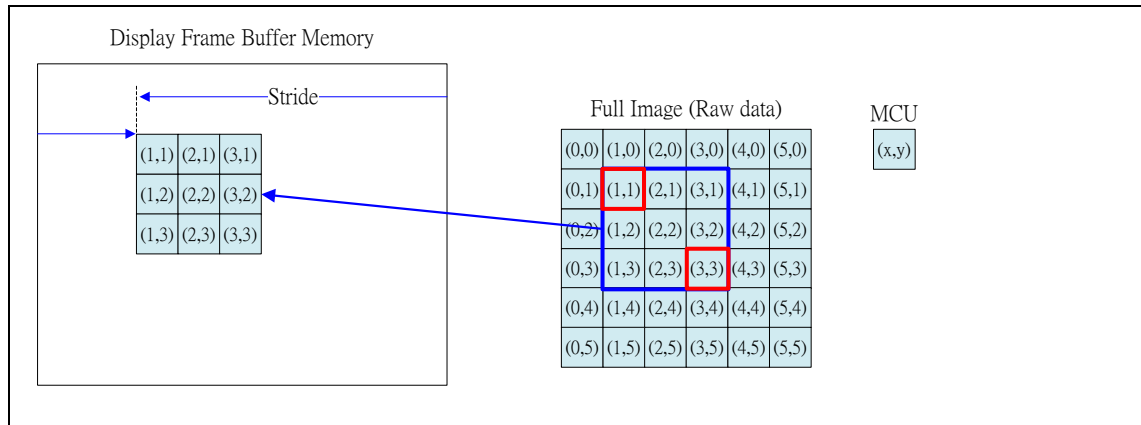
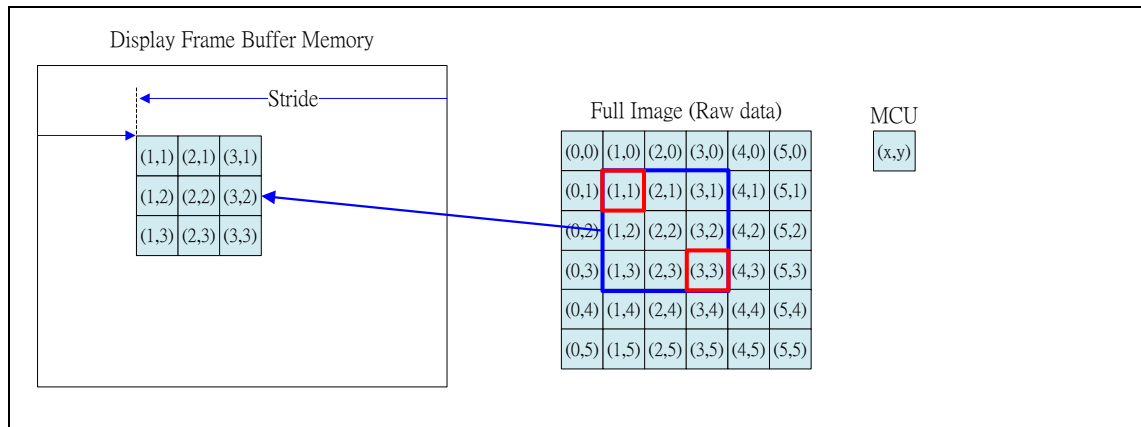
### 7.3.3.4 Decode stride

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.



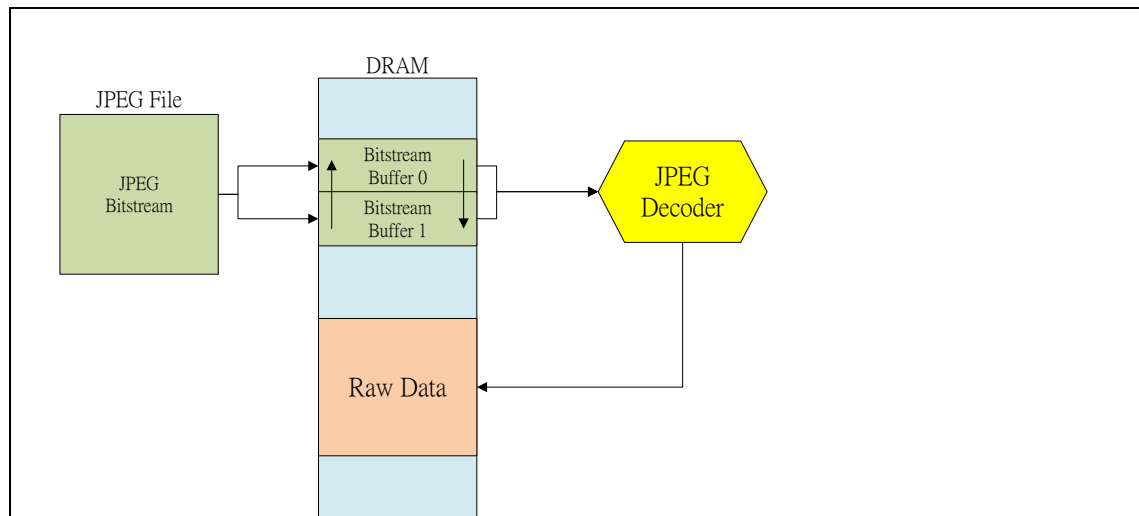
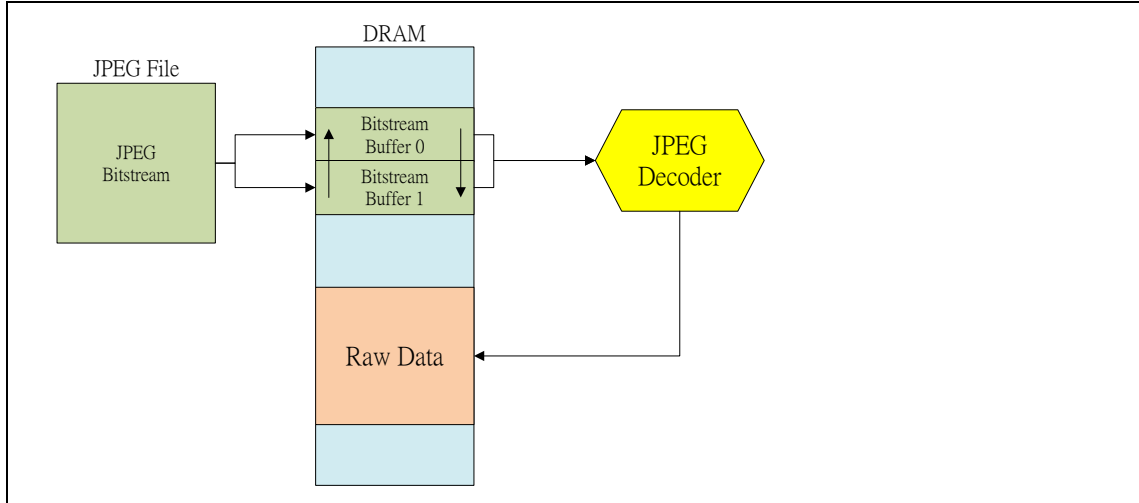
### 7.3.3.5 Window Decode

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.



### 7.3.3.6 Decode Input Wait

When the JPEG is in decoding mode, the input source is the JPEG bit-stream written by host. The bit-stream buffer size is in 2K unit dual-buffer manner. If the buffer-size is 2KB, host need to fill 1KB bit-stream into one of the half buffer region before resuming JPEG operation when an input-wait interrupt is generated.



### 7.3.3.7 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
- ◆ Change output buffer address → JPEG\_IOCTL\_SET\_YADDR
- ◆ Set Downscale → JPEG\_IOCTL\_SET\_DECODE\_DOWNSCALE
- ◆ Set Decode output Stride → JPEG\_IOCTL\_SET\_DECODE\_STRIDE
- ◆ Set windows decode → JPEG\_IOCTL\_SET\_WINDOW\_DECODE



### 7.3.4 JPEG Library Constant Definition

#### Error Code

Name	Value	Description
E_FAIL	0	Fail
E_SUCCESS	1	Success
E_JPEG_INVALID_PARAM	2	Invalid parameter
E_JPEG_TIMEOUT	3	Time out

#### Encode operation

Name	Value	Description
Encode format		
JPEG_ENC_PRIMARY	0	Encode operation : Primary JPEG
JPEG_ENC_THUMBNAI	1	Encode operation : Thumbnail JPEG
JPEG_ENC_SOURCE_PLANAR	0	Encode source : planar format
JPEG_ENC_SOURCE_PACKET	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_THUMBNAI_YUV420	0x90	Thumbnail Encode image format : YUV 4:2:0
JPEG_ENC_THUMBNAI_YUV422	0x98	Thumbnail Encode image format : YUV 4:2:2
JPEG_ENC_THUMBNAI_GRAY	0x91	Thumbnail Encode image format : GRAY
Encode Header control		
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_THUMBNAI_DRI	0x1	Restart Interval in Thumbnail JPEG Header
JPEG_ENC_THUMBNAI_QTAB	0x2	Quantization-Table in Thumbnail JPEG Header
JPEG_ENC_THUMBNAI_HTAB	0x4	Huffman-Table in Thumbnail JPEG Header
JPEG_ENC_THUMBNAI_JFIF	0x8	JFIF Header in Thumbnail JPEG Header

## Decode operation

Name	Value	Description
Decode output format		
JPEG_DEC_PRIMARY_PLANAR_YUV	0x08021	Primary Decode output format : planar format
JPEG_DEC_PRIMARY_PACKET_YUV422	0x00021	Primary Decode output format : planar YUV422
JPEG_DEC_PRIMARY_PACKET_RGB555	0x04021	Primary Decode output format : packet RGB555
JPEG_DEC_PRIMARY_PACKET_RGB565	0x06021	Primary Decode output format : packet RGB565
JPEG_DEC_PRIMARY_PACKET_RGB888	0x14021	Primary Decode output format : packet RGB888
JPEG_DEC_THUMBNAIL_PLANAR_YUV	0x08011	Thumbnail Decode output format : planar YUV
JPEG_DEC_THUMBNAIL_PACKET_YUV422	0x00031	Thumbnail Decode output format : packet RGB555
JPEG_DEC_THUMBNAIL_PACKET_RGB555	0x40031	Thumbnail Decode output format : packet RGB565
JPEG format		
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

### 7.3.5 JPEG Library Property Definition

The JPEG library provides the property structure to set JPEG property.

#### JPEG\_INFO\_T;

Name	Value	Description
yuvformat	JPEG_DEC_YUV420 JPEG_DEC_YUV422 JPEG_DEC_YUV444 JPEG_DEC_YUV411 JPEG_DEC_GRAY JPEG_DEC_YUV422T	JPEG format (Decode only)
width	< 8192	Decode Output width (Decode only)
height	< 8192	Decode Output height (Decode only)
jpeg_width	< 65535	JPEG width (Decode only)
jpeg_height	< 65535	JPEG height (Decode only)
stride	< 8192	Decode output Stride (Decode only)
bufferend	Reserved	Reserved

image_size[2]	< $2^{24} - 1$	Encode Bitstream Size (Encode Only)
---------------	----------------	-------------------------------------

The JPEG library provides 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

### 7.3.6 JPEG Library APIs Specification

#### 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 completes and get JPEG bit stream size after encode completes.

### 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 needs to wait the completion flag while JPEG engine completes its job.

#### Parameter

None

#### Return Value

E_FAIL	Error happens
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_YA DDR	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_USSTRIDE	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_BIT STREAM_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_YUV422 JPEG_DEC_PRIMARY_PACKET_RGB555 JPEG_DEC_PRIMARY_PACKET_RGB565		Specify the decoded image output format

	JPEG_DEC_PRIMARY_PACKET_RGB888 JPEG_DEC_THUMBNAI_PLANAR_YUV JPEG_DEC_THUMBNAI_PACKET_YUV422 JPEG_DEC_THUMBNAI_PACKET_RGB555		
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_HEADERDECODE_CALLBACKFUN	Header Decode Complete Call Back function pointer		Set Header Decode Complete Call Back function pointer
JPEG_IOCTL_SET_DECODEINPUTWAIT_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_THUMBNAI	Quantization-Table Adjustment and control values[0]	Set Quantization-Table Adjustment and control
JPEG_IOCTL_ENCODE_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_THUMBNAI_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_THUMBNAI_RESTART_INTERVAL	The pointer to store Thumbnail Restart interval size		Get Thumbnail Restart interval size
JPEG_IOCTL_SET_THUMBNAI_DIMENSION	Thumbnail Height	Thumbnail Width	Set Thumbnail Dimension
JPEG_IOCTL_SET_ENCODE_SW_OFFSET	Offset		Set Software Encode Offset

JPEG_IOCTL_GET_THUMB_NAIL_DIMENSION	The pointer to store Thumbnail Height	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 Height	Primary Downscaled Width	Set Primary Encode downscale Size (Planar format only)
JPEG_IOCTL_SET_ENCODE_THUMB_NAIL_DOWNSCALE	Thumbnail Downscaled Height	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)

#### Parameter

cmd	Command
arg0	The first argument of the command
arg1	The 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 pixels) */
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
P_QADJUST				P_QVS			

Bits	Descriptions	
[7:4]	P_QADJUST	Primary Quantization-Table Adjustment If the sum of the position (x, y) of quantization-table is greater than P_QADJUST, the quantization value will be set to 127. Otherwise the value will keep as the original. 8x8 DCT block: x = 0~7, y = 0~7 if ((x+y) > P_QADJUST) => Q' = 127

		else	=> Q' = Q
[3:0]	P_QVS	Primary Quantization-Table Scaling Control $Q' = (P\_QVS[3]*2*Q)+(P\_QVS[2]*Q)+(P\_QVS[1]*Q/2)+(P\_QVS[0]*Q/4)$	

## 7.4 Example code

This demo code has sample code for “Normal Encode”, “Encode Upscale”, “Normal Decode”, “Decode Downscale”, “Decode Input”, and “Stride” (write/read from SD Card).

Please refer to the JPEG sample code of BSP Non-OS.

## 8 KPI Library

### 8.1 Overview

The GPIO library provides a set of APIs to control keypad interface. This library depends on both N9H20 System Library and N9H20 GPIO Library.

### 8.2 KPI Library APIs Specification Functions

#### kpi\_init

**Synopsis**

```
void kpi_init (void)
```

**Description**

This function initialized the keypad interface.

**Parameter**

None

**Return Value**

None

**Example**

```
kpi_init();
```

#### kpi\_open

**Synopsis**

```
int kpi_open (unsigned int src)
```

**Description**

This function is used to open keypad interface. kpi\_init() should be called before this function.

**Parameter**

src          External interrupt source for KPI to use

**Return Value**

Return 0 on success, -1 for parameter error, or duplicate open call

**Example**

```
/* Assign nIRQ3 for KPI */  
kpi_open (3);
```

#### kpi\_close

### Synopsis

void kpi\_close (void)

### Description

This function is used to close keypad interface.

### Parameter

None

### Return Value

None

### Example

```
kpi_close ();
```

## kpi\_read

### Synopsis

int kpi\_read (unsigned char mode)

### Description

This function is used to read keypad input. It supports both blocking and non-blocking mode.

### Parameter

mode      Read mode, KPI\_NONBLOCK or KPI\_BLOCK

### Return Value

Return -1 for unknown read mode or un-opened interface. Return 0 for no key in non-blocking mode. Return key value in other situation.

### Example

```
/* Read in blocking mode*/
kpi_read (KPI_BLOCK);
```

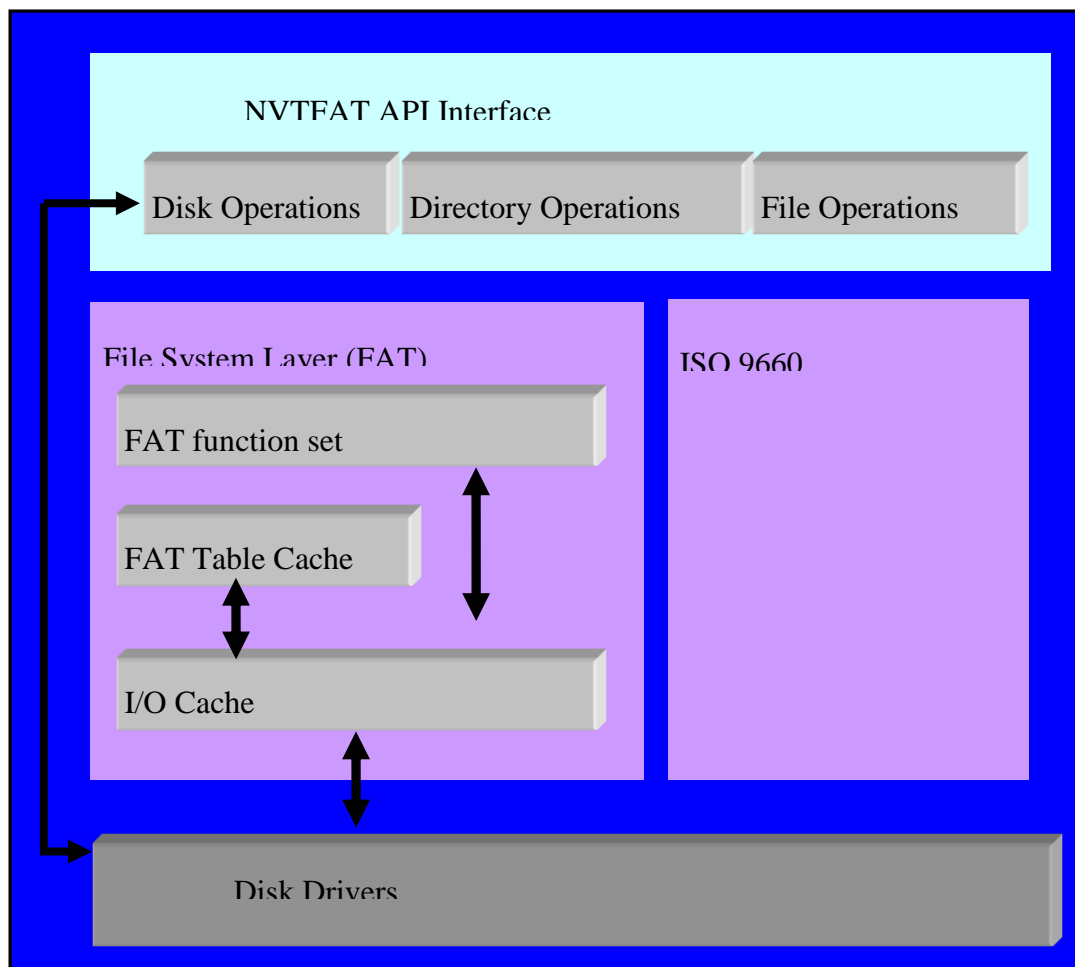
## 9 NVTFAT Library

### 9.1 Features

The NVTFAT File System Library has the following features:

- Support FAT12/FAT16/FAT32
- Support multiple disks and multiple partitions
- Dynamically mount and un-mount disk
- Support sub-directory
- Support long file name. The length of file name can be up to 514 characters. The length of file path, including the file name, can be up to 520 characters.
- Can format flash memory cards
- Get disk physical size and free space
- Can open at most 12 files at the same time
- Open files with create, truncate, append
- Create, delete, rename, move, copy, seek, read, and write files
- Enumerate files under a directory
- Get file position and get file status
- Set file size and set file attributes
- Create, rename, remove, and move directories

### 9.2 General Description



### 9.3 Initialize File System

To initialize this file system, just invoke *fsInitFileSystem()*. The underlying disk driver should be initialized followed the file system initialization.

### 9.4 Error Code

Because the file operation may fail due to the various reasons, it's strongly recommended that application should check the return value of each file system API call. The File System Library provides the very detailed error code to indicate the error reasons.

### 9.5 File Handle

File handle is a handle obtained by opening a file. Application should check the return value of *fsOpenFile()*. If the return value > 0, it's a valid handle. Otherwise, some errors happened for the file open operation. A file handle is valid until the file was closed by *fsCloseFile()*.

### 9.6 Format Flash Memory Card

The File System Library provides *fsFormatFlashMemoryCard()* to format flash memory card, such as SD, MMC, CF, or Smart Media. This function requires caller to pass a physical disk pointer as parameter, which can be obtained by *fsGetFullDiskInformation()*.

The format of File System Library was fully compliant to Smart Media disk format standard. The rules of disk formatting are defined in table 2-1.

Disk Size	FAT Type	Cluster Size	Capacity
1 MB	FAT12	4 KB	984 KB
2 MB	FAT12	4 KB	1984 KB
4 MB	FAT12	8 KB	3976 KB
8 MB	FAT12	8 KB	7976 KB
16 MB	FAT12	16 KB	15968 KB
32 MB	FAT12	16 KB	31968 KB
64 MB	FAT12	16 KB	63952 KB
128 MB	FAT16	16 KB	127936 KB
256 MB	FAT16	32 KB	255744 KB
512 MB	FAT16	32 KB	511744 KB
1024 MB	FAT16	32 KB	1023616 KB
2048 MB	FAT16	32 KB	2047288 KB

Table 9-1 Disk Format

## 9.7 File Operations

Many of the file operations can be done only if the file has been opened. These file operations determine the target by file handle. In this section, all file operations based on file handle will be introduced.

### 9.7.1 Open File

To read or write a file, applications must first open the file and obtain a file handle, which is an integer. Function *fsOpenFile()* is used to open a file. If the opening file operation succeed, the caller will obtain a file handle, whose value is  $\geq 3000$ . Otherwise, the call will receive a negative value, which represented an error code (refer to *Error Code Table*).

Function *fsOpenFile()* receives two parameters. The first parameter is the full path file name of the file to be opened. Both long file name or short file name are acceptable and are non-case-sensitive. The full path file name must also include disk number. For example, the full path file name is "C:\\OpenATestFile.txt" or "C:\\OpenAT~1.txt". The second parameter is combination of control flags. It uses bit-OR to represent various control flags. The control flags and their effectiveness are listed in Table 2-2.

Flag	Description
O_RDONLY	Open with read capability. In addition, O_DIR and O_APPEND have implicit read capability.
O_WRONLY	Open with write capability. In addition, O_APPEND, O_CREATE, and O_TRUNC have implicit write capability.
O_RDWR	Open with read and write capabilities
O_APPEND	Open an exist file and set the file access position to end of file. O_APPEND has implicit read and write capabilities.

O_CREATE	Open or create a file. If the file did not exist, File System Library would create it. Otherwise, if the file existed, File System Library would just open it and set file access position to start of file. O_CREATE has implicit write capability.
O_TRUNC	Open an existed file and truncate it. If the file did not exist, return an error code. If the file existed, open it. O_TRUNC has implicit write capability.
O_FSEEK	File system will create cluster chain for this file to speed up file seeking operation. It will allocate 1KB extra memory.

Table 9-2 File open control flags

## 9.7.2 File Access Position

Each opened file has one and only one access position. Subsequent *fsReadFile()* and *fsWriteFile()* operations are started from the file access position. File access position can be obtained by *fsGetFilePosition()* and can be changed by *fsFileSeek()*.

When a file was opened, the file access position was initially set as 0, that is, start of file. The only exception is a file opened with O\_APPEND flag. In this case, the file access position will be set as end of file.

When file access position is at the end of file, *fsReadFile()* will result in EOF error, while *fsWriteFile()* will extend the file size.

## 9.7.3 Read File

A file can be read after it was opened. *fsReadFile()* was used to read data from a file. It receives a file handle as the first parameter, which was previously obtained by *fsOpenFile()*. The general scenario of reading files is:

*fsOpenFile()* → *fsReadFile()* → *fsCloseFile()*

## 9.7.4 Write File

A file can be written after it was opened with write capability. *fsWriteFile()* was used to write data to a file. It receives a file handle as the first parameter, which was previously obtained by *fsOpenFile()*. The general scenario of writing files is:

*fsOpenFile()* → *fsWriteFile()* → *fsCloseFile()*

## 9.7.5 Directory Operations

File System Library supports sub-directory and provides supporting routines to manage directories. It supports directory creation, remove, rename, and move.

## 9.7.6 Create/Remove Directories

*fsMakeDirectory()* can be used to create a new directory. Directory name can be long file name, and the name must not be conflicted with any existed files or sub-directories under the same directory.

*fsRemoveDirectory()* can be used to remove an empty directory. If there are some files or sub-directories under the directory to be removed, an error will be received. Root directory cannot be removed.

## 9.7.7 Move/Rename Directories

A directory can be completely moved from a directory to another directory. *fsMoveFile()* can be used to move directory. All files and sub-directories under that directory will be completely moved at the same time. If the target directory contained a file or directory whose name was conflicted with the directory to be moved, the operation will be canceled.

A directory can be renamed with *fsRenameFile()*. If the new name will be conflicted with any existed files or directories under the same directory, the operation will be canceled.

## 9.7.8 Delete/Rename/Move Files

A file can be deleted with *fsDeleteFile()*. All disk space occupied by this file will be released immediately and



can be used by other files.

A file can be moved from a directory to another directory with *fsMoveFile()*. If the target directory contained a file or directory whose name was conflicted with the file to be moved, the operation will be canceled.

A file can be renamed with *fsRenameFile()*. If the name will be conflicted with any existed files or directories under the same directory, the operation will be canceled.

### 9.7.9 Enumerate Files In a Directory

File System Library provides a set of functions to support the enumerating files under a specific directory. These functions are ***FSFINDFIRST()***, ***FSFINDNEXT()***, and ***FSFINDCLOSE()***.

Firstly user uses ***FSFINDFIRST()*** to specify the directory to be searched, and specify search conditions. If there is any file or sub-directory to match the search conditions, ***FSFINDFIRST()*** will return 0 and user can obtain a file-find object (FILE\_FIND\_T) . The file-find object contains the information of the first found file, including the file name and attributes. User can use the same file-find object to do the subsequent searches by calling ***FSFINDNEXT()***. Each call to ***FSFINDNEXT()*** will obtain a newly found file or sub-directory, if it returns 0. ***FSFINDNEXT()*** returns non-zero value means that there is no any other file or sub-directory to match the search conditions and the file enumeration should be terminated. User should call ***FSFINDCLOSE()*** to terminate a search series.

## 9.8 File System Library APIs Specification

### 9.8.1 Disk Operations

#### fsPhysicalDiskConnected

##### Synopsis

```
INT fsPhysicalDiskConnected(PDISK_T *ptPDisk)
```

##### Description

Register and parsing a newly detected disk.

##### Parameter

ptPDisk                      The pointer refers to the physical disk descriptor

##### Return Value

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

##### Example

```
STORAGE_DRIVER_T SD0DiskDriver =
{ /* SD driver low level operation API */
    sd_disk_init0,
    sd_disk_read0,
    sd_disk_write0,
    sd_disk_ioctl0
};
/* Reference SIC driver */
```

```
pDisk->szManufacture[0] = '\\0';
strcpy(pDisk->szProduct, (char *)pSDDisk->product);
strcpy(pDisk->szSerialNo, (char *)pSDDisk->serial);
pDisk->nDiskType = DISK_TYPE_SD_MMC;
pDisk->nPartitionN = 0;
pDisk->ptPartList = NULL;
pDisk->nSectorSize = 512;
pDisk->uTotalSectorN = pSDDisk->totalSectorN;
pDisk->uDiskSize = pSDDisk->diskSize;

pDisk->ptDriver = &_SD0DiskDriver; /* register low level operation API
*/
fsPhysicalDiskConnected(pDisk);
...
```

## fsPhysicalDiskDisconnected

### Synopsis

```
INT fsPhysicalDiskDisconnected(PDISK_T *ptPDisk)
```

### Description

Flush I/O cache and unlink logical disk as remove physical disk

### Parameter

ptPDisk                      The pointer refers to the physical disk descriptor

### Return Value

0 – Success

Otherwise – error code defined in Error Code Table

### Example

```
static INT ram_disk_init(PDISK_T *ptPDisk)
{
    return 0;
}

static INT ram_disk_ioctl(PDISK_T *ptPDisk, INT control, VOID *param)
{
    return 0;
}

static INT ram_disk_read(PDISK_T *ptPDisk, UINT32 uSecNo,
                        INT nSecCnt, UINT8
                        *pucBuff)
```

```
{
    memcpy(pucBuff, (UINT8 *)(_RAMDiskBase + uSecNo * 512), nSecCnt *
512);
    return FS_OK;
}
static INT ram_disk_write(PDISK_T *ptPDisk, UINT32 uSecNo,
                           INT nSecCnt, UINT8
*pucBuff, BOOL bWait)
{
    memcpy((UINT8 *)(_RAMDiskBase + uSecNo * 512), pucBuff, nSecCnt *
512);
    return FS_OK;
}
STORAGE_DRIVER_T _RAMDiskDriver =
{
    ram_disk_init,
    ram_disk_read,
    ram_disk_write,
    ram_disk_ioctl,
};
static PDISK_T *ptRAMDisk;
INT32 RemoveRAMDisk(void)
{
    fsPhysicalDiskDisconnected(ptRAMDisk);
    return 0;
}
```

## fsUnmountPhysicalDisk

### Synopsis

INT fsUnmountPhysicalDisk(PDISK\_T \*ptPDisk)

### Description

Flush I/O cache and unlink logical disk as remove physical disk. The function is almost same as function-fsPhysicalDiskDisconnected

### Parameter

ptPDisk                      The pointer refers to the physical disk descriptor

### Return Value

0 – Success

Otherwise – error code defined in Error Code Table

### Example

```
PDISK_T *pDisk_SD0 = NULL;
/* Reference SIC driver */
/* Detect Card insert */
...
pDisk_SD0 = pDisk;
fsPhysicalDiskConnected(pDisk);
...
...
/* Detect Card remove */
fsUnmountPhysicalDisk(pDisk_SD0);
free(pDisk_SD0);
pDisk_SD0 = NULL;
```

## fsDiskFreeSpace

### Synopsis

```
INT fsDiskFreeSpace(INT nDriveNo, UINT32 *puBlockSize,
                    UINT32 *puFreeSize, INT32 *puDiskSize);
```

### Description

Format a flash memory card by FAT12/FAT16/FAT32 format. NVT FAT will first create a MBR for this disk and configure it to be the single partition. Then NVT FAT will format it to be FAT12/FAT16 format.

### Parameter

ptPDisk    Get free space of disk <driveNo>

### Return Value

0 – Success

Otherwise – error code defined in Error Code Table

### Example

```
UINT32 uBlockSize, uFreeSize, uDiskSize;
...
if (fsDiskFreeSpace ('C', &blockSize, &freeSize,
&diskSize) == FS_OK)
    sysprintf("Disk C block size=%d, free space=%d MB,
disk size=%d MB\n", blockSize, (INT)freeSize/1024,
(INT)diskSize/1024;
```

## fsFormatFlashMemoryCard

### Synopsis

```
INT fsFormatFlashMemoryCard(PDISK_T *ptPDisk);
```

### Description

Format a flash memory card by FAT12/FAT16/FAT32 format. NVTFAT will first create a MBR for this disk and configure it to be the single partition. Then NVTFAT will format it to be FAT12/FAT16 format.

### Parameter

ptPDisk The pointer refers to the physical disk descriptor.

### Return Value

0 – Success

Otherwise – error code defined in Error Code Table

### Example

```
PDISK_T      *ptPDiskList, *ptPDisk;
PARTITION_T  *ptPartition;

/* Get complete disk information */
ptPDiskList = fsGetFullDiskInfomation();

/* Format the first physical disk */
ptPartition = ptPDiskList;
ptPDisk = ptPDiskList;      /* format the first physical disk */
fsFormatDiskPartition(ptPDisk);
/* Release allocated memory */
fsReleaseDiskInformation(pDiskList);
```

## fsTwoPartAndFormatAll

### Synopsis

```
INT fsTwoPartAndFormatAll(PDISK_T *ptPDisk,
                          INT firstPartSize,
                          INT secondPartSize);
```

### Description

Configure the disk to be two partitions and format these two partitions as FAT32 format. If the total sizes of these two partitions are larger than disk size, NVTFAT will automatically shrink the size of the second partition to fit disk size.

### Parameter

ptPDisk The pointer refers to the physical disk descriptor.

firstPartSize      The size (in KBs) of the first partition  
secondPartSize    The size (in KBs) of the second partition.

#### Return Value

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

#### Example

```
PDISK_T      *ptPDiskList, *ptPDisk;
PARTITION_T *ptPartition;

/* Get complete disk information */
ptPDiskList = fsGetFullDiskInformation();

/* Format the first physical disk */
ptPartition = ptPDiskList;
ptPDisk = ptPDiskList;      /* format the first physical disk */
fsTwoPartAndFormatAll(ptPDisk, 2048, 10240);

/* Release allocated memory */
fsReleaseDiskInformation(pDiskList);
```

### fsAssignDriveNumber

#### Synopsis

```
INT fsAssignDriveNumber (INT nDriveNo,
                        INT disk_type,
                        INT instance,
                        INT partition)
```

#### Description

Claim the drive number assignment. This API must be called prior to fsInitFileSystem().

#### Parameter

nDriveNo    The drive number. Valid number is 'A' ~ 'Z'.  
disk\_type    Disk type defines in nvtfat.h. Prefixed with "DISK\_TYPE\_".  
              For example, NAND disk type is DISK\_TYPE\_SMART\_MEDIA  
instance    The disk instance of specified <disk\_type>, start from 0.  
              For example, the first NAND disk is instance 0, the second NAND is instance 1  
Partition    Which partition of the specified <disk\_type><instance>.  
              The first partition is 1, the second partition is 2, and so on.

### Return Value

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

### Example

```
// SD0 first partition => C
fsAssignDriveNumber('C', DISK_TYPE_SD_MMC, 0, 1);

// NAND0 first partition => E
fsAssignDriveNumber('E', DISK_TYPE_SMART_MEDIA, 0, 1);

// NAND1 first partition => H
fsAssignDriveNumber('H', DISK_TYPE_SMART_MEDIA, 1, 1);

// NAND1 second partition => I
fsAssignDriveNumber('I', DISK_TYPE_SMART_MEDIA, 1, 2);
```

## fsFormatFixedDrive

### Synopsis

INT fsFormatFixedDrive (INT nDriveNo)

### Description

Format the specified drive. The drive number must be have been successfully assigned by fsAssignDriveNumber().

### Parameter

nDriveNo The drive number. Valid number is 'A' ~ 'Z'.

### Return Value

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

### Example

```
#define DISK_TYPE_SMART_MEDIA 0x00000008 // defined in nvtfat.h
#define DISK_TYPE_SD_MMC      0x00000020 // defined in nvtfat.h

fsAssignDriveNumber('C', DISK_TYPE_SD_MMC, 0, 1);
fsAssignDriveNumber('E', DISK_TYPE_SMART_MEDIA, 0, 1);

fsFormatFixedDrive('C');
fsFormatFixedDrive('E');
```

## fsGetFullDiskInfomation

### Synopsis

```
PDISK_T *fsGetFullDiskInfomation(VOID)
```

### Description

Get the complete information list of physical disk, disk partitions, and logical disk information. The returned PDISK\_T pointer was referred to a dynamically allocated memory, which contains the complete disk information list. Note that caller is responsible to deallocate it by calling fsReleaseDiskInformation().

### Parameter

None

### Return Value

0 – Success

Otherwise – error code defined in Error Code Table

### Example

```
PDISK_T      *pDiskList, *ptPDiskPtr;
PARTITION_T  *ptPartition;
INT          nDiskIdx = 0;
INT          nPartIdx;
ptPDiskPtr = pDiskList = fsGetFullDiskInfomation();
while (ptPDiskPtr != NULL)
{
    sysprintf("\n\n=== Disk %d (%s) =====\n",
        nDiskIdx++, (ptPDiskPtr->nDiskType &
            DISK_TYPE_USB_DEVICE) ? "USB" : "IDE");
    sysprintf("    name:    [%s%s]\n", ptPDiskPtr->szManufacture,
        ptPDiskPtr->szProduct);
    sysprintf("    head:    [%d]\n", ptPDiskPtr->nHeadNum);
    sysprintf("    sector:  [%d]\n", ptPDiskPtr->nSectorNum);
    sysprintf("    cylinder: [%d]\n", ptPDiskPtr->nCylinderNum);
    sysprintf("    size:    [%d MB]\n", ptPDiskPtr->uDiskSize / 1024);

    ptPartition = ptPDiskPtr->ptPartList;
    nPartIdx = 1;
    while (ptPartition != NULL)
    {
        sysprintf("\n    --- Partition %d ----- \n",
            nPartIdx++);
        sysprintf("        active: [%s]\n",
```



```
(ptPartition->ucState & 0x80) ? "Yes" : "No");
sysprintf("      size:  [%d MB]\n",
          (ptPartition->uTotalSecN / 1024) / 2);
sysprintf("      start: [%d]\n", ptPartition->uStartSecN);
sysprintf("      type:  ");
ptPartition = ptPartition->ptNextPart;
}
ptDiskPtr = ptDiskPtr->ptDiskAllLink;
}
fsReleaseDiskInformation(pDiskList);
```

## fsReleaseDiskInformation

### Synopsis

```
VOID fsReleaseDiskInformation(PDISK_T *ptPDiskList)
```

### Description

Release the memory allocated by fsGetFullDiskInformation().

### Parameter

ptPDiskList	The PDISK_T pointer returned by the previous call to fsGetFullDiskInformation()
-------------	---

### Return Value

0 – Success  
 Otherwise – error code defined in Error Code Table  
 Example  
 See example code of fsGetFullDiskInformation()

## fsReleaseDiskInformation

### Synopsis

```
VOID fsReleaseDiskInformation(PDISK_T *ptPDiskList)
```

### Description

Release the memory allocated by fsGetFullDiskInformation().

### Parameter

ptPDiskList	The PDISK_T pointer returned by the previous call to fsGetFullDiskInformation()
-------------	---

### Return Value

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

### Example

See example code of fsGetFullDiskInformation()

## fsInitFileSystem

### Synopsis

VOID fsInitFileSystem(VOID)

### Description

Initialize file system.

### Parameter

None

### Return Value

None

### Example

```
sysEnableCache(CACHE_WRITE_THROUGH);
fsInitFileSystem();
fmiInitDevice();
fmiInitSDDDevice();
```

## fsFixDriveNumber

### Synopsis

INT fsFixDriveNumber(CHAR sd\_drive,  
CHAR sm\_drive,  
CHAR cf\_drive)

### Description

Specify the fixed driver number of SD card, SM/NAND, and CF. If the specified drive number was used, NTFAT will find other driver number for it. This API must be called prior to fsInitFileSystem().

### Parameter

sd_drive	'A' ~ 'Z'
sm_drive	'A' ~ 'Z'
cf_drive	'A' ~ 'Z'

### Return Value

0	Success
---	---------

ERR\_DRIVE\_INVALID\_NUMBER

invalid drive

number one

Example

```
fsFixDriveNumber('D', 'C', 'F');
fsInitFileSystem();
```

## fsSetReservedArea

### Synopsis

INT fsSetReservedArea(UINT32 u32StartSector)

### Description

Specify the start sector in file system.

### Parameter

u32StartSector      Start sector of file system. To set the start sector is only for special application. The reserved space may store some binary image or data for booting. The function should be called before format disk.

### Return Value

Success

### Example

```
#define RESERVED_SIZE (1024*1024)

fsSetReservedArea(RESERVED_SIZE/512);/* Start sector from 2048 sector
*/
pDiskList = fsGetFullDiskInfomation();
fsFormatFlashMemoryCard(pDiskList);
```

## 9.9 File/Directory Operations

### fsCloseFile

#### Synopsis

INT fsCloseFile(INT hFile)

#### Description

Close a file, that was previously opened by fsOpenFile().

#### Parameter

hFile                      The file handle of the file to be closed.

#### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

### Example

Refer to the example of fsOpenFile().

## fsDeleteFile

### Synopsis

INT fsDeleteFile(CHAR \*suFileName, CHAR \*szAsciiName)

### Description

Delete a file.

### Parameter

suFileName The Unicode full path of file name for the file to be opened.

The file name must include its absolute full path with drive number specified. The full path file name must be ended with two 0x00 character.

szAsciiName The ASCII version name of <suFileName> excluding the file path. This parameter is optional. Caller must set this parameter as NULL if it was not used. If caller did not give the ASCII name, NVT FAT will generate the ASCII version name from the <suFileName>. Note that if two-bytes code language was used in <suFileName>, NVT FAT generated ASCII version name will be incorrect. It was suggested to set this parameter if two-bytes code language contained in <suFileName>

### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

### Example

```
CHAR suFileName[] = { 'C', 0, ':', 0, '\\', 0, 'l', 0, 'o', 0,
                      'g', 0, '.', 0, 't', 0, 'x', 0, 't', 0, 0, 0 };
INT nStatus;
/* Delete file C:\log.txt */
nStatus = fsDeleteFile(suFileName, NULL);
if (nStatus < 0)
    sysprintf("Cannot delete file log.txt!\n");
```

## fsFileSeek

### Synopsis

INT64 fsFileSeek(INT64 hFile, INT n64Offset, INT16 usWhence)

### Description

Set the current read/write position of an opened file.

### Parameter

hFile                      The file handle of the file to be closed.  
n64Offset    Byte offset from the position indicated by <usWhence>  
usWhence    Seek position base

Table 9-3: Seek Position Base

usWhence	Description
SEEK_SET	"file offset 0" + <n64Offset>
SEEK_CUR	file current position" + <n64Offset>
SEEK_END	"end of file position"+ <n64Offset>

### Return Value

0                      Success  
Otherwise              error code defined in Error Code Table

### Example

```
INT    hFile, nReadLen;
CHAR  suFileName[] = { 'C', 0, ':', 0, '\\', 0, '1', 0, 'o', 0,
                        'g', 0, '.', 0, 't', 0, 'x', 0, 't', 0, 0, 0};
UINT8  pucBuff[64];

if ((hFile = fsOpenFile(suFileName, NULL, O_RDONLY) < 0)
    return hFile;
/* read 10 bytes from file offset 1000 */
fsFileSeek(hFile, 1000, SEEK_SET);
fsReadFile(hFile, pucBuff, 10, &nReadLen)
fsCloseFile(hFile);
```

## fsIsEOF

### Synopsis

BOOL fsIsEOF(INT hFile)

### Description

Check whether the file pointer has reached the end of file or not.

**Parameter**

hFile      The file handle of the file.

**Return Value**

TRUE      It's end of file.

FALSE     It's not end of file.

**Example**

Refer to the example of fsFindFirst().

**fsFindClose****Synopsis**

```
INT fsFindClose(FILE_FIND_T *ptFindObj)
```

**Description**

Close a search series.

**Parameter**

ptFindObj The file-search object obtained by previous fsFindFirst() call.

**Return Value**

FS\_OK     Success.

Otherwise   Error code defined in Error Code Table

**Example**

Refer to the example of fsFindFirst().

**fsFindFirst****Synopsis**

```
INT fsFindFirst (CHAR *suDirName,  
                CHAR *szAsciiName,  
                FILE_FIND_T *ptFindObj)
```

**Description**

Start a file search and get the first file/directory entry found.

**Parameter**

suDirName      The Unicode full path name of the directory to be searched.  
The name must include its absolute full path with drive number  
specified. The full path name must be ended with two  
0x00 characters.

szAsciiName     The ASCII version name of <suDirName> excluding the

path part. This parameter is optional. Caller must set this parameter as NULL if it was not used. If caller did not give the ASCII name, NVT FAT will generate the ASCII version name from the <suDirName>. Note that if two-bytes code language was used in <suDirName>, NVT FAT generated ASCII version name will be incorrect. It was suggested to set this parameter if two-bytes code language contained in <suDirName>.

ptFindObj caller prepares file/directory entry container.

### Return Value

0	Success
Otherwise	error code defined in Error Code Table

### Example

```

INT ListDir(CHAR *szPath)
{
    INT          nIdx, nStatus;
    CHAR          szMainName[12], szExtName[8], *pcPtr;
    FILE_FIND_T   tFileInfo;
    nStatus = fsFindFirst(szPath, NULL, &tFileInfo);
    if (nStatus < 0)
        return nStatus;

    do
    {
        pcPtr = tFileInfo.szShortName;
        if ((tFileInfo.ucAttrib & A_DIR) &&
            (!strcmp(pcPtr, ".") || !strcmp(pcPtr, "..")))
            strcat(tFileInfo.szShortName, ".");
        memset(szMainName, 0x20, 9);
        szMainName[8] = 0;
        memset(szExtName, 0x20, 4);
        szExtName[3] = 0;
        i = 0;
        while (*pcPtr && (*pcPtr != '.'))
            szMainName[i++] = *pcPtr++;
        if (*pcPtr++)
        {
            nIdx = 0;
            while (*pcPtr)

```

```

        szExtName[nIdx++] = *pcPtr++;
    }

    if (tFileInfo.ucAttrib & A_DIR)
        sysprintf("%s %s <DIR> %02d-%02d-%04d %02d:%02d %s\n",
            szMainName, szExtName, tFileInfo.ucWDateMonth,
            tFileInfo.ucWDateDay, tFileInfo.ucWDateYear+80)%100 ,
            tFileInfo.ucWTimeHour, tFileInfo.ucWTimeMin,
            tFileInfo.szLongName);
    else
        sysprintf("%s %s %10d %02d-%02d-%04d %02d:%02d %s\n",
            szMainName, szExtName, (UINT32)tFileInfo.nFileSize,
            tFileInfo.ucWDateMonth, tFileInfo.ucWDateDay,
            (tFileInfo.ucWDateYear+80)%100, tFileInfo.ucWTimeHour,
            tFileInfo.ucWTimeMin, tFileInfo.szLongName);
    } while (!fsFindNext(&tFileInfo));
    fsFindClose(&tFileInfo);
}

```

## fsFindNext

### Synopsis

INT fsFindNext(FILE\_FIND\_T \*ptFindObj)

### Description

Continue the previous fsFindFirst() file search and get the next matched file. If there's no more match found, the search series will be closed automatically.

### Parameter

ptFindObj The file-search object used in the previous fsFindFirst() call.

### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

### Example

Refer to the example of fsFindFirst().

## fsGetFilePosition

### Synopsis

INT fsGetFilePosition(INT hFile, UINT32 \*puPos)

### Description



Get the current read/write position of an opened file.

**Parameter**

hFile	The file handle of the opened file to get file read/write position.
puPos	The current read/write position.

**Return Value**

FS\_OK – Success

Otherwise – error code defined in Error Code Table

**Example**

```
INT    hFile, nStatus;
UINT32 uFilePos;
/* Open a read-only file */
hFile = fsOpenFile(file, O_RDONLY);
fsFileSeek(hFile, 1000, SEEK_SET);
fsGetFilePosition(hFile, &uFilePos);
sysprintf("Current file position is: %d\n", uFilePos);
fsCloseFile(hFile);
```

**fsGetFileSize****Synopsis**

```
INT fsGetFileSize(INT hFile)
```

**Description**

Get the current size of an opened file.

**Parameter**

hFile	The file handle of the opened file to get size.
-------	---

**Return Value**

FS\_OK – Success

Otherwise – error code defined in Error Code Table

**Example**

```
INT    hFile, nStatus;
UINT32 uFilePos;
/* Open a read-only file */
hFile = fsOpenFile(file, O_RDONLY);
sysprintf("The size of %s is %d\n", file, fsGetFileSize(hFile));
fsCloseFile(hFile);
```

## fsGetFileStatus

### Synopsis

```
INT fsGetFileStatus(INT hFile,
                    CHAR *suFileName,
                    CHAR *szAsciiName,
                    FILE_STAT_T *ptFileStat)
```

### Description

Get the file status of a specific file or directory.

### Parameter

hFile                    The file handle of the opened file.

suFileName              The Unicode full path of file name for the file to be opened.  
It was used only if <hFile> is < 0.

szAsciiName             The ASCII version name of <suFileName> excluding the  
file path.

ptFileStat              Caller prepares the container to receive status of this file.

### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

### Example

```
CHAR suFileName[] = { 'C', 0, ':', 0, '\\', 0, '1', 0, 'o',
0,
'g', 0, '.', 0, 't', 0, 'x', 0, 't', 0, 0, 0};
FILE_STAT_T tFileStat;
INT nStatus
if ((nStatus = fsGetFileStatus(-1, suFileName, NULL, &stat)) < 0)
{
    sysprintf("fsGetFileStatus failed\n");
    fsGetErrorDescription(nStatus, NULL, 1);
    return nStatus;
}
```

## fsMakeDirectory

### Synopsis

```
INT fsMakeDirectory(CHAR *suDirName, CHAR *szAsciiName)
```

### Description

Create a new directory if not exists.

#### Parameter

suDirName        The Unicode full path name of the directory to be created.

szAsciiName      The ASCII version name of <suDirName> excluding the path part.

#### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

#### Example

```
CHAR suDirName[] = { 'C', 0, ':', 0, '\\', 0, 't', 0, 'e',
0,
'm', 0, 'p', 0, 0, 0 };
/* Create a new directory "temp" under "C:\" */
fsMakeDirectory(suDirName, NULL);
```

### fsMoveFile

#### Synopsis

```
INT fsMoveFile(CHAR *suOldName,
               CHAR *szOldAsciiName,
               CHAR *suNewName,
               CHAR *szNewAsciiName,
               INT blsDirectory)
```

#### Description

Move a file or a whole directory.

#### Parameter

suOldName        The Unicode full path name of the file/directory to be moved.

szOldAsciiName   The ASCII version name of < suOldName > excluding the path part.

suNewName        The Unicode full path name of the old file/directory to be moved to.

szNewAsciiName   The ASCII version name of < suNewName > excluding the path part.

blsDirectory     TRUE: is moving a directory; FALSE: is moving a file

#### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

### Example

```
CHAR  szOldFile[] = "C:\\log.txt"
CHAR  szNewFile[] = "C:\\temp\\log.txt"
CHAR  suOldFile[128], suNewFile[128];

fsAsciiToUnicode(szOldFile, suOldFile, TRUE);
fsAsciiToUnicode(szNewFile, suNewFile, TRUE);
fsMoveFile(suOldFile, NULL, suNewFile, "log.txt", FALSE);
```

## fsCopyFile

### Synopsis

```
INT  fsCopyFile(CHAR *suSrcName,
                CHAR *szSrcAsciiName,
                CHAR *suDstName,
                CHAR *szDstAsciiName)
```

### Description

Copy a file. (Copy directory was not allowed.)

### Parameter

suSrcName	The Unicode full path name of the file to be copied.
szSrcAsciiName	The ASCII version name of < suSrcName > excluding the path part.
suDstName	The Unicode full path name of the file/directory to be
	generated.
szDstAsciiName	The ASCII version name of < suDsrName > excluding the path part.

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

### Example

Refer to the example of fsOpenFile().

## fsCloseFile

### Synopsis

```
INT  fsCloseFile(INT hFile)
```

### Description

Close a file, that was previously opened by fsOpenFile().

### Parameter

hFile                      The file handle of the file to be closed.

### Return Value

FS\_OK – Success

Otherwise – error code defined in Error Code Table

### Example

None.

## fsOpenFile

### Synopsis

INT fsOpenFile(CHAR \*suFileName, CHAR \*szAsciiName,UINT32 uFlag)

### Description

Open/Create a file.

### Parameter

suFileName                      The Unicode full path file name of the file to be opened. The

file name must include its absolute full path with drive number

specified. The full path file name must be ended with two 0x00 character.

szAsciiName                      The ASCII version name of <suFileName> excluding the

file path. This parameter is optional. Caller must set this parameter as NULL if it was not used. If caller did not give the ASCII name, NVTFAT will generate the ASCII version name from the <suFileName>. Note that if two-bytes code language was used in <suFileName>, NVTFAT generated ASCII version name will be incorrect. It was suggested to set this parameter if two-bytes code language contained in <suFileName>.

uFlag

Table 9-4: Open File capability

uFlag	Description
O_ RDONLY	open file with read capability

O_WRONLY	open file with write capability
O_APPEND	open file with write-append operation, the file position was set to end of file on open
O_CREATE	If the file exists, open it. If the file is not exists, create it.
O_TRUNC	Open a file and truncate it, file size becomes 0
O_DIR	open a directory file

#### Return Value

< 0                      error code defined in Error Code Table  
 Otherwise                file handle

#### Example

```

        INT  hFile;
    CHAR  suFileName[] = { 'C', 0, ':', 0, '\\', 0, '1', 0, 'o', 0,
                           'g', 0, '.', 0, 't', 0, 'x', 0, 't', 0, 0, 0};
    CHAR  szAsciiName[] = "log.txt";
    /* Open a read-only file */
    hFile = fsOpenFile(suFileName, szAsciiName, O_RDONLY);
    if (hFile < 0)
        return hFile;
    fsCloseFile(hFile);
    
```

### fsReadFile

#### Synopsis

INT fsReadFile(INT hFile, UINT8 \*pucBuff, INT nBytes, INT \*pnReadCnt)

#### Description

Read <nBytes> of octets from an opened file

#### Parameter

hFile                      The file handle of an opened file.  
 pucBuff  
 file                      Refer to the buffer to receive data read from the specified  
 nBytes                    Number of bytes to read  
 pnReadCnt                Number of bytes actually read.

#### Return Value

FS\_OK      Success  
 Otherwise                error code defined in Error Code Table

### Example

```

UINT8  pucBuff[4096];
INT     hFileSrc, hFileOut;
INT     nReadLen, nWriteLen, nStatus;
if ((hFileSrc = fsOpenFile("C:\\log.txt", O_RDONLY) < 0)
return hFileSrc;
if ((hFileOut = fsOpenFile("C:\\logcopy.txt", O_CREATE) < 0)
return hFileOut;
while (1)
{
    if ((nStatus = fsReadFile(hFileSrc, pucBuff, 4096,
&nReadLen) < 0)
        break;
    if ((nStatus = fsWriteFile(hFileOut, pucBuff, nReadLen,
&nWriteLen);
        break;
    if ((nReadLen < 4096) || (nWriteLen != nReadLen)
        break;
}
fsCloseFile(hFileSrc);
fsCloseFile(hFileOut);

```

## fsRemoveDirectory

### Synopsis

INT fsRemoveDirectory(CHAR \*suDirName, CHAR \*szAsciiName)

### Description

Remove an empty directory. If the directory is not empty, an  
ERR\_DIR\_REMOVE\_NOT\_EMPTY error will be returned.

### Parameter

suDirName	The Unicode full path name of the directory to be removed.
szAsciiName	The ASCII version name of <suDirName> excluding the path part.

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

### Example

```

CHAR  szDirName[] = "C:\\temp"
CHAR  suDirName[128];

```

```
fsAsciiToUnicode(szDirName, suDirName, TRUE);  
/* Remove directory "C:\temp" */  
nStatus = fsRemoveDirectory(suDirName, "temp");
```

## fsRenameFile

### Synopsis

```
INT fsRenameFile(CHAR *suOldName,  
                 CHAR *szOldAsciiName,  
                 CHAR *suNewName,  
                 CHAR *szNewAsciiName,  
                 BOOL blsDirectory)
```

### Description

Rename a file or directory.

### Parameter

suOldName	The Unicode full path name of the file/directory to be renamed.
szOldAsciiName	The ASCII version name of < suOldName > excluding the path part.
suNewName	Rename into the Unicode full path name of the file/directory
szNewAsciiName	The ASCII version name of < suNewName > excluding the path part.
blsDirectory	TRUE: is renaming a directory; FALSE: is renaming a file.

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

### Example

```
CHAR szOldFile[] = "C:\log.txt"  
CHAR szNewFile[] = "C:\log2.txt"  
CHAR suOldFile[128], suNewFile[128];  
  
fsAsciiToUnicode(szOldFile, suOldFile, TRUE);  
fsAsciiToUnicode(szNewFile, suNewFile, TRUE);  
fsRenameFile(suOldFile, NULL, suNewFile, "log2.txt", FALSE);
```



## fsSetFileAttribut

### Synopsis

```
INT fsSetFileAttribute(INT hFile,
                      CHAR *suFileName,
                      CHAR *szAsciiName,
                      UINT8 ucAttrib,
                      FILE_STAT_T *ptFileStat);
```

### Description

Modify file attribute of a specific file or directory..

### Parameter

hFile	The file handle of the opened file to be set attribute,
suFileName	The Unicode full path of file name for the file to be set attribute.
szAsciiName	The ASCII version name of <suFileName> excluding the file path.
ptFileStat	The specified file attribute.

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

### Example

```
CHAR szFileName[] = "C:\\temp"
CHAR suFileName[128];
FILE_STAT_T tFileStat;
fsGetFileStatus(-1, suFileName, NULL, &tFileStat)
/* force changing file to be hidden */
tFileStat.ucAttrib |= FA_HIDDEN;
fsSetFileAttribute(-1, suFileName, NULL, &tFileStat);
```

## fsSetFileSize

### Synopsis

```
INT fsSetFileSize(INT hFile,
                 CHAR *suFileName,
                 CHAR *szAsciiName,
                 UINT32 nNewSize)
```

### Description

Resize the file size. If specified new size is larger than the current size, NVT FAT will allocate disk space and extend this file. On the other hand, if specified new size is smaller than the current size, this file will be truncated.

### Parameter

hFile	The file handle of the opened file.
suFileName	The Unicode full path of file name for the file to be set size. It was used only if <hFile> is < 0..
szAsciiName	The ASCII version name of <suFileName> excluding the file path.
newSize	Set the file size to be extended to or truncated.

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

### Example

```
int ChangeFileSize(INT hFile, INT32 uLen)
{
    if (fsSetFileSize(hFile, NULL, NULL, uLen) < 0)
        sysprintf("fsSetFileSize error!!\n");
}
```

## fsSetFileTime

### Synopsis

```
INT fsSetFileTime(INT hFile,
                  CHAR *suFileName,
                  CHAR *szAsciiName,
                  UINT8 ucYear,
                  UINT8 ucMonth,
                  UINT8 ucDay,
                  UINT8 ucHour,
                  UINT8 ucMin,
                  UINT8 ucSec);
```

### Description

Set the date/time attribute of a file/directory. Note that fsSetFileTime() will set the last access date and modify date/time, but the create date/time was left unchanged.

### Parameter

hFile	The file handle of the opened file.
suFileName	The Unicode full path of file name for the file to be set time. It was used only if <hFile> is < 0..
szAsciiName	The ASCII version name of <suFileName> excluding the file path.
ucYear	Years from 1980. For example, for 2003, <year> is equal to 23.
ucMonth	1 <= month <= 12
ucHour	0 <= hour <= 23
ucMin	0 <= min <= 59
unSec	0 <= sec <= 59

#### Return Value

0	Success
Otherwise	error code defined in Error Code Table

#### Example

None

### fsWriteFile

#### Synopsis

```
INT fsWriteFile(INT hFile, UINT8 *pucBuff, INT nBytes, INT *pnWriteCnt)
```

#### Description

Write <nBytes> bytes data to an opened file

#### Parameter

hFile	The file handle of an opened file.
pucBuff	The buffer contains the data to be written
nBytes	Number of bytes to written
pnWriteCnt	Number of bytes actually written

#### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

#### Example

```
int CopyFile(int hFileSrc, int hFileOut)
{
    UINT8 pucBuff[4096];
    INT nReadLen, nWriteLen, nStatus;
```

```
while (1)
{
    if (fsReadFile(hFileSrc, pucBuff, 4096, &nReadLen) < 0)
        break;
    fsWriteFile(hFileOut, pucBuff, nReadLen, &nWriteLen);
    if ((nReadLen < 4096) || (nWriteLen != nReadLen))
        break;
}
}
```

## 9.10 Language Support

### fsUnicodeToAscii

#### Synopsis

```
INT fsUnicodeToAscii(VOID *pvUniStr,
                     VOID *pvASCII,
                     BOOL blsNullTerm)
```

#### Description

Translate a Unicode string into an ASCII string. This function can only translate single byte language (for example, English). If the Unicode string contained two-bytes code language (for example, BIG5 or GB), the translation result will be wrong, because NVT FAT has no built-in Unicode-ASCII translation table.

#### Parameter

pvUniStr	The Unicode string to be translated. It must be ended with two 0x0 characters.
pvASCII translation result.	Caller prepares the container to accommodate the
blsNullTerm	Add a NULL character (0x0) to the end of pvASCII

#### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

#### Example

```
CHAR suRoot[] = { 'C', 0, ':', 0, '\\', 0, 0, 0 };
CHAR szLongName[MAX_FILE_NAME_LEN/2];
FILE_FIND_T tFileInfo;
fsFindFirst(suRoot, NULL, &tFileInfo); /* C:\ */
do
{
```

```
fsUnicodeToAscii(tFileInfo.suLongName, szLongName, TRUE);
sysprintf("%s\n szLongName);
} while (!fsFindNext(&tFileInfo));
fsFindClose(&tFileInfo);
```

## fsAsciiToUnicode

### Synopsis

```
INT fsAsciiToUnicode(VOID *pvASCII,
                     VOID *pvUniStr,
                     BOOL blsNullTerm)
```

### Description

Translate an ASCII string into a Unicode string. This function can only translate single byte language (for example, English). If the ASCII string contained two-bytes code language (for example, BIG5 or GB), the translation result will be wrong, because NVT FAT has no built-in ASCII-Unicode translation table.

### Parameter.

pvASCII	The ASCII string to be translated. It must be NULL-terminated.
pvUniStr	Caller prepares the container to accommodate the translation result.
blsNullTerm	Add two 0x0 characters to the end of pvUniStr

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table

### Example

```
CHAR szDirName[] = "C:\temp"
CHAR suDirName[128];
fsAsciiToUnicode(szDirName, suDirName, TRUE);
/* Remove directory "C:\temp" */
nStatus = fsRemoveDirectory(suDirName, "temp");
```

## fsUnicodeNonCaseCompare

### Synopsis

```
INT fsUnicodeNonCaseCompare(VOID *pvUnicode1,
                             VOID *pvUnicode2)
```

### Description

Compare two Unicode strings by case non-sensitive. The Unicode strings must be ended with two 0x0 characters.

**Parameter.**

pvUnicode1 compared.	The source (0x0,0x0)-ended Unicode string to
pvUnicode2 compared.	The target (0x0,0x0)-ended Unicode string to

**Return Value**

0	The two Unicode strings are treated to be equal
Otherwise	The two Unicode strings are treated to be unequal

**Example**

```
CHAR  szName1[] = "log.txt"
CHAR  szName2[] = "Log.TXT";
CHAR  suName1[32], suName2[32];
fsAsciiToUnicode(szName1, suName1, TRUE);
fsAsciiToUnicode(szName2, suName2, TRUE);
if (fsUnicodeNonCaseCompare(suName1, suName2) == 0)
    sysprintf("Equal!\n");
else
    sysprintf("Non-equal!");
```

## fsUnicodeCopyStr

**Synopsis**

```
INT  fsUnicodeCopyStr(VOID *pvStr1,
                      VOID *pvStr2)
```

**Description**

Copy a Unicode string

**Parameter.**

pvStr1	The Unicode string to be copied to.
pvStr2	The source Unicode string. It must be (0x0,0x0)-ended.

**Return Value**

0	The two Unicode strings are treated to be equal
Otherwise	The two Unicode strings are treated to be unequal

**Example**

```
CHAR  szName1[] = "log.txt"
CHAR  szName2[] = "Log.TXT";
```

```
CHAR suName1[32], suName2[32];
fsAsciiToUnicode(szName1, suName1, TRUE);
fsAsciiToUnicode(szName2, suName2, TRUE);
if (fsUnicodeNonCaseCompare(suName1, suName2) == 0)
    sysprintf("Equal!\n");
else
    sysprintf("Non-equal!");
```

## fsUnicodeNonCaseCompare

### Synopsis

```
INT fsUnicodeNonCaseCompare(VOID *pvUnicode1,
                             VOID *pvUnicode2)
```

### Description

Compare two Unicode strings by case non-sensitive. The Unicode strings must be ended with two 0x0 characters.

### Parameter.

pvUnicode1	The source (0x0,0x0)-ended Unicode string to compared.
pvUnicode2	The target (0x0,0x0)-ended Unicode string to compared.

### Return Value

0	The two Unicode strings are treated to be equal
Otherwise	The two Unicode strings are treated to be unequal

### Example

```
CHAR szName1[] = "log.txt"
CHAR szName2[] = "Log.TXT";
CHAR suName1[32], suName2[32];
fsAsciiToUnicode(szName1, suName1, TRUE);
fsAsciiToUnicode(szName2, suName2, TRUE);
if (fsUnicodeNonCaseCompare(suName1, suName2) == 0)
    sysprintf("Equal!\n");
else
    sysprintf("Non-equal!");
```

## fsUnicodeCopyStr

### Synopsis

```
INT fsUnicodeCopyStr(VOID *pvStr1,
```

VOID \*pvStr2)

### Description

Copy a Unicode string

### Parameter.

pvStr1	The Unicode string to be copied to.
pvStr2	The source Unicode string. It must be (0x0,0x0)-ended.

### Return Value

FS_OK	Success
Otherwise	error code defined in Error Code Table.

### Example

```
FILE_FIND_T tFileInfo;
CHAR      suSlash[] = { '\\', 0x00, 0x00, 0x00 };
CHAR      suFullName[MAX_PATH_LEN];
INT nLen, nStatus;
fsFindFirst(suDirName, NULL, &tFileInfo);
do
{
    fsUnicodeCopyStr(suFullName, suDirName);
    fsUnicodeStrCat(suFullName, suSlash);
    fsUnicodeStrCat(suFullName, tFileInfo.suLongName);
    fsDeleteFile(suFullName, NULL);
} while (!fsFindNext(&tFileInfo));
fsFindClose(&tFileInfo);
```

## fsUnicodeStrCat

### Synopsis

```
INT fsUnicodeStrCat(VOID *pvUniStr1,
                   VOID *pvUniStr2)
```

### Description

Concatenate two (0x0,0x0)-ended Unicode strings.

### Parameter.

pvUniStr1	The Unicode string to be concatenated to.
pvUniStr2	The Unicode to be concatenated to the end of < pvUniStr1>.

### Return Value

FS_OK	Success
-------	---------



Otherwise error code defined in Error Code Table.

### Example

Refer to the example of fsUnicodeCopyStr();

**Error Code Table**

Code Name	Value	Description
ERR_FILE_EOF	0xFFFF8200	End of file
ERR_GENERAL_FILE_ERROR	0xFFFF8202	General file error
ERR_NO_FREE_MEMORY	0xFFFF8204	No available memory
ERR_NO_FREE_BUFFER	0xFFFF8206	No available sector buffer
ERR_NOT_SUPPORTED	0xFFFF8208	Operation was not supported
ERR_UNKNOWN_OP_CODE	0xFFFF820A	Unrecognized operation code
ERR_INTERNAL_ERROR	0xFFFF820C	File system internal error
ERR_FILE_NOT_FOUND	0xFFFF8220	File not found
ERR_FILE_INVALID_NAME	0xFFFF8222	Invalid file name
ERR_FILE_INVALID_HANDLE	0xFFFF8224	Invalid file handle
ERR_FILE_IS_DIRECTORY	0xFFFF8226	The file to be opened is a directory
ERR_FILE_IS_NOT_DIRECTORY	0xFFFF8228	The directory to be opened is a file
ERR_FILE_CREATE_NEW	0xFFFF822A	Cannot create new directory entry
ERR_FILE_OPEN_MAX_LIMIT	0xFFFF822C	Number of opened files has reached limitation
ERR_FILE_RENAME_EXIST	0xFFFF822E	Rename file conflict with an existent file
ERR_FILE_INVALID_OP	0xFFFF8230	Invalid file operation
ERR_FILE_INVALID_ATTR	0xFFFF8232	Invalid file attribute
ERR_FILE_INVALID_TIME	0xFFFF8234	Invalid time specified
ERR_FILE_TRUNC_UNDER	0xFFFF8236	Truncate file underflow, size < pos
ERR_FILE_NO_MORE	0xFFFF8238	Actually not an error, used to identify end of file in the enumeration of a directory
ERR_FILE_IS_CORRUPT	0xFFFF823A	File is corrupt
ERR_PATH_INVALID	0xFFFF8260	Invalid path name
ERR_PATH_TOO_LONG	0xFFFF8262	Path too long
ERR_PATH_NOT_FOUND	0xFFFF8264	Path not found
ERR_DRIVE_NOT_FOUND	0xFFFF8270	Drive not found, the disk may have been unmounted
ERR_DRIVE_INVALID_NUMBER	0xFFFF8272	Invalid drive number
ERR_DRIVE_NO_FREE_SLOT	0xFFFF8274	Cannot mount more drive
ERR_DIR_BUILD_EXIST	0xFFFF8290	Try to build an existent directory
ERR_DIR_REMOVE_MISS	0xFFFF8292	Try to remove a nonexistent directory
ERR_DIR_REMOVE_ROOT	0xFFFF8294	Try to remove root directory
ERR_DIR_REMOVE_NOT_EMPTY	0xFFFF8296	Try to remove a non-empty directory
ERR_DIR_DIFFERENT_DRIVE	0xFFFF8298	Specified files on different drive
ERR_DIR_ROOT_FULL	0xFFFF829A	FAT12/FAT16 root directory full

ERR_DIR_SET_SIZE	0xFFFF829C	Try to set file size of a directory
ERR_READ_VIOLATE	0xFFFF82C0	User has no read privilege
ERR_WRITE_VIOLATE	0xFFFF82C2	User has no write privilege
ERR_ACCESS_VIOLATE	0xFFFF82C4	Cannot access
ERR_READ_ONLY	0xFFFF82C6	Try to write a read-only file
ERR_WRITE_CAP	0xFFFF82C8	Try to write file/directory which was opened with read-only
ERR_NO_DISK_MOUNT	0xFFFF8300	There's no any disk mounted
ERR_DISK_CHANGE_DIRTY	0xFFFF8302	Disk change, buffer is dirty
ERR_DISK_REMOVED	0xFFFF8304	Portable disk has been removed
ERR_DISK_WRITE_PROTECT	0xFFFF8306	Disk is write-protected
ERR_DISK_FULL	0xFFFF8308	Disk full
ERR_DISK_BAD_PARTITION	0xFFFF830A	Bad partition
ERR_DISK_UNKNOWN_PARTITION	0xFFFF830C	Unknown or not supported partition type
ERR_DISK_UNFORMAT	0xFFFF830E	Disk partition was not formatted
ERR_DISK_UNKNOWN_FORMAT	0xFFFF8310	Unknown disk format
ERR_DISK_BAD_BPB	0xFFFF8312	Bad BPB, disk may not be formatted
ERR_DISK_IO	0xFFFF8314	Disk I/O failure
ERR_DISK_IO_TIMEOUT	0xFFFF8316	Disk I/O time-out
ERR_DISK_FAT_BAD_CLUS	0xFFFF8318	Bad cluster number in FAT table
ERR_DISK_IO_BUSY	0xFFFF831A	I/O device is busy writing, must retry. direct-write mode only
ERR_DISK_INVALID_PARM	0xFFFF831C	Invalid parameter
ERR_DISK_CANNOT_LOCK	0xFFFF831E	Cannot lock disk, the disk was in-use or locked by other one
ERR_SEEK_SET_EXCEED	0xFFFF8350	File seek set exceed end-of-file
ERR_ACCESS_SEEK_WRITE	0xFFFF8352	Try to seek a file which was opened for written
ERR_FILE_SYSTEM_NOT_INIT	0xFFFF83A0	File system was not initialized
ERR_ILLEGAL_ATTR_CHANGE	0xFFFF83A2	Illegal file attribute change

## 10 PWM Library

### 10.1 Overview

This library is designed to make user application to set N9H20 PWM more easily.

The PWM library has the following features:

- PWM signal frequency and duty setting
- PWM Capture function

### 10.2 Programming Guide

#### 10.2.1 System Overview

The N9H20 have 4 channels pwm-timers. The 4 channels pwm-timers has 2 prescalers, 2 clock dividers, 4 clock selectors, 4 16-bit counters, 4 16-bit comparators, 2 Dead-Zone generators. 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 prescaler0 and channel2-3 share prescaler1). 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 receives 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 N9H20 has 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 have 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.

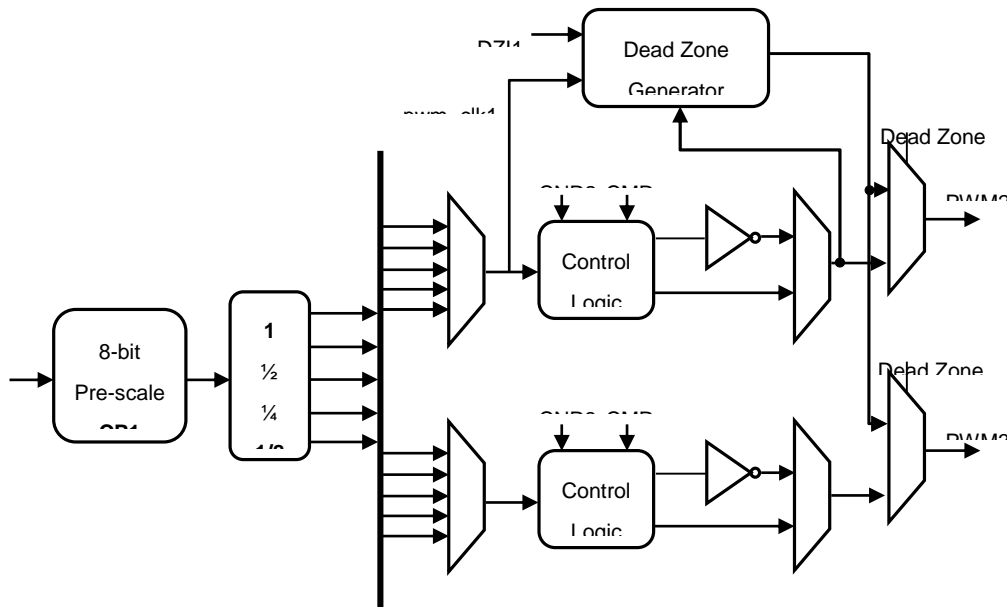
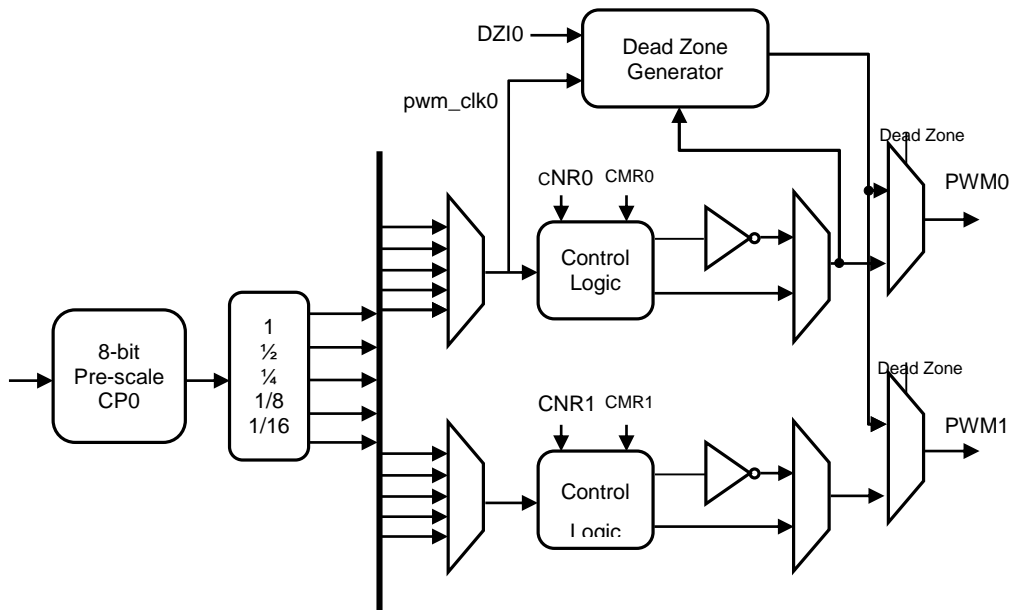
There are only four interrupts from PWM to advanced interrupt controller (AIC). PWM 0 and Capture 0 share the same interrupt channel, PWM1 and Capture 1 share the same interrupt and so on. Therefore, PWM function and Capture function in the same channel cannot be used at the same time.

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 generators
- Capture function

#### 10.2.2 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)



### 10.2.3 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).

$$\text{period} = \frac{1}{(PCLK) \div (PPR + 1) \div CSR}$$

When the PCLK = 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

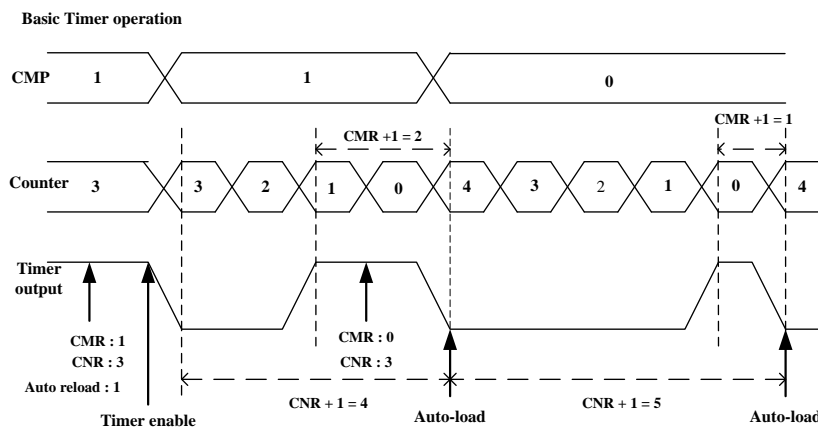
$$\text{period}_{\max} = \frac{1}{(60\text{MHz}) \div (255 + 1) \div 16} = 68.266\mu\text{s}$$

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  $(65535+1) \cdot (68.266\mu\text{s})$  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



## PWM Double Buffering and Automatic Reload

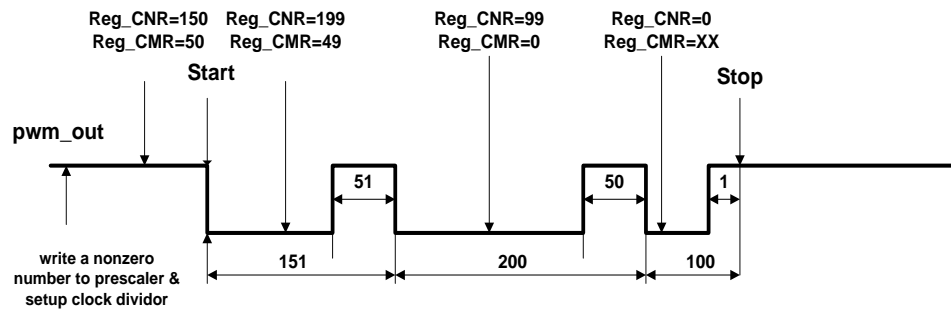
N9H20 PWM Timers have a double buffering function, enabling the reload value changed for next timer

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.

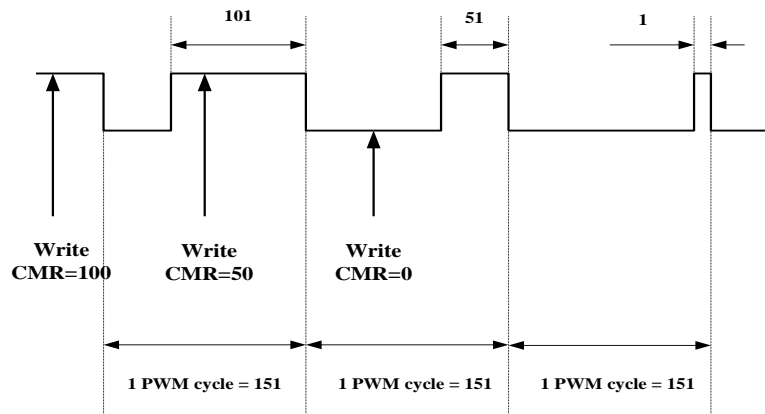
The auto-reload operation copies from CNR0~3 to down-counter when down-counter reaches zero. If CNR0~3 are set as zero, counter will be halt when counter count to zero. If auto-reload bit is set as zero, counter will be stopped immediately

#### PWM double buffering



The double buffering function allows CMR written at any point in current cycle. The loaded value will take effect from next cycle.

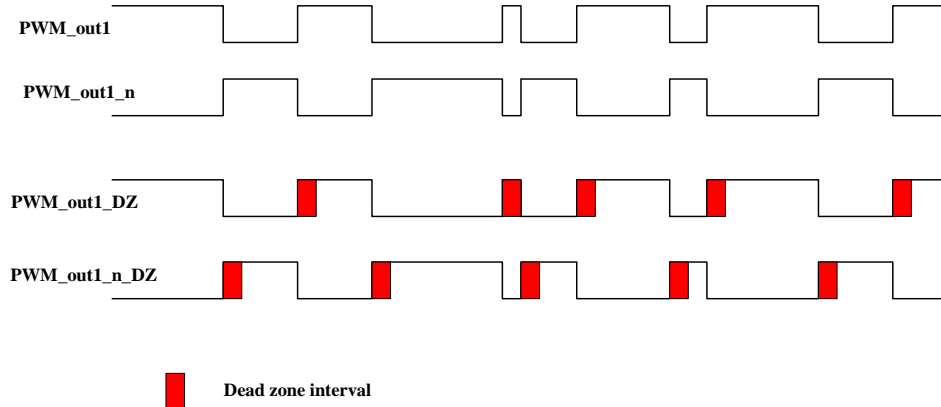
#### Modulate PWM controller output duty ratio(CNR = 150)



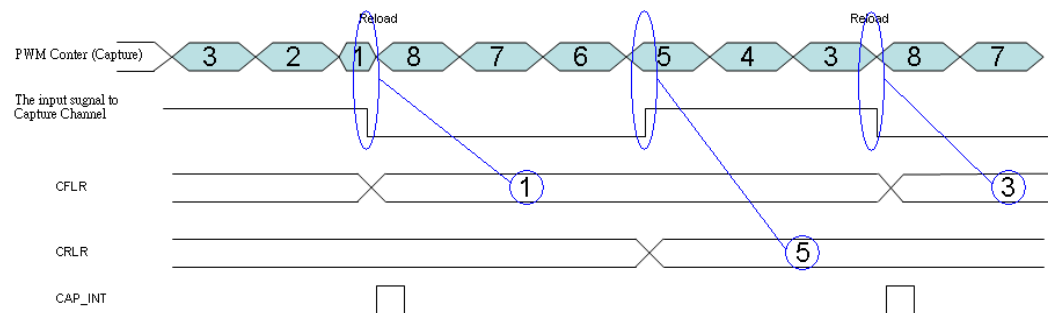
#### Dead-Zone Generator

N9H20 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:

When set falling interrupt enable, the pwm counter will be reload at time of interrupt occur.

The channel low pulse width is (CNR – CRLR).

The channel high pulse width is (CRLR - CFLR).

The channel cycle time is (CNR – CFLR).

## 10.2.4 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	0x10	PWM Capture 0
PWM_CAP1	0x11	PWM Capture 1
PWM_CAP2	0x12	PWM Capture 2
PWM_CAP3	0x13	PWM Capture 3

PWM_CAP_ALL_INT	0	PWM Capture All 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	1	Input clock divided by 1
PWM_CLOCK_DIV_2	2	Input clock divided by 2
PWM_CLOCK_DIV_4	4	Input clock divided by 4
PWM_CLOCK_DIV_8	8	Input clock divided by 8
PWM_CLOCK_DIV_16	16	Input clock divided by 16
PWM_TOGGLE_MODE	TRUE	PWM Timer Toggle mode
PWM_ONE_SHOT_MODE	FALSE	PWM Timer One-shot mode

### 10.2.5 PWM Library Property Definition

The PWM library provides property structure to set PWM timer property.

Name	Value	Description
fFrequency	>= 0	The timer/capture frequency[0]
u8HighPulseRatio	1~100	High pulse ratio
u8Mode	PWM_ONE_SHOT_MODE / PWM_TOGGLE_MODE	PWM Timer Trigger mode
blInverter	TRUE / FALSE	Inverter Enable / Inverter Disable
u8ClockSelector	PWM_CLOCK_DIV_1/ PWM_CLOCK_DIV_2/ PWM_CLOCK_DIV_4/ PWM_CLOCK_DIV_8/ PWM_CLOCK_DIV_16	Clock Selector [1]
u16PreScale	2 ~ 256	Clock Prescale [1]
u32Duty	0~65535	Pulse duty [2]

[0] PWM provides two timer setting mode: Frequency-setting and Property-setting modes.

- Frequency-setting mode (**fFrequency** > 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 (**fFrequency** = 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.

## 10.3 PWM Library APIs Specification

### 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

```
PWM_Close();
```

### 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 is set by PWM timer.

### Note

1. 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
2. The function can set the proper frequency property (Clock selector/Prescale) for capture function and user needs to set the proper pulse duty 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 PWM Timer 0 */
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

TURE                    - 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

### 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 PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3
u8Int capture	Capture interrupt type (The parameter is valid only when 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

### 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.

**FALSE** - The specified interrupt doesn't occur.

### Example

```
/* Get PWM Timer 0 Interrupt flag*/
PWM_GetIntFlag(PWM_TIMER0);
```

## PWM\_GetCaptureIntStatus

## Synopsis

BOOL 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	Capture interrupt type (The parameter is valid only when function)
------------------	---

PWM\_CAP\_RISING\_INT: The capture rising interrupt.

PWM\_CAP\_FALLING\_INT: The capture falling interrupt.

## Return Value

TRUE - The specified interrupt occurs.

**FALSE** - 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    Capture interrupt type (The parameter is valid only when 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

This function is used to get 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

The 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

This function is used to get 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**

The value which latches the counter when there's a falling transition

**Example**

```
/* Get the Falling Counter Data */  
u32Count[u32i++] = PWM_GetFallingCounter(PWM_CAP0);
```

## 11 RTC Library

### 11.1 Overview

This library is designed to make user application access N9H20 RTC more easily.

The RTC library has the following features:

- Support RTC Current/Alarm time access.
- Support System Power Off Control

### 11.2 Programming Guide

#### 11.2.1 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.
- Alarm register (second, minute, hour, day, month, year).
- 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

#### 11.2.2 System Power Control Flow

##### Normal system Power Control Flow

The control steps are as follows

User presses the power key, RPWR, to makes the power control signal, PWRCE pin, to high. If the PWR\_ON bit, PWRON[0], be set, the power key can be released and the PWRCE will keep on. If the PWR\_ON bit, PWRON [0], doesn't be set as 1, the PWRCE will back to low when the power key is released.

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 PWRCE will go to low to turn off the core power. If the PWRON bit is also kept high, the PWRCE pin will keep in

high level. If there is no any pulse on the power key and the PWR\_ON bit is clear by user, the PWRCE pin is also set to low at this time.

The following table is the system power control flow table.

Input		Output	Note
PWRKey	PWR_ON	PWCE	
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 when X2=1
PWCE is open drain output X1, internal pull-up X2, it is R/W able There is Interrupt from key be pressed			

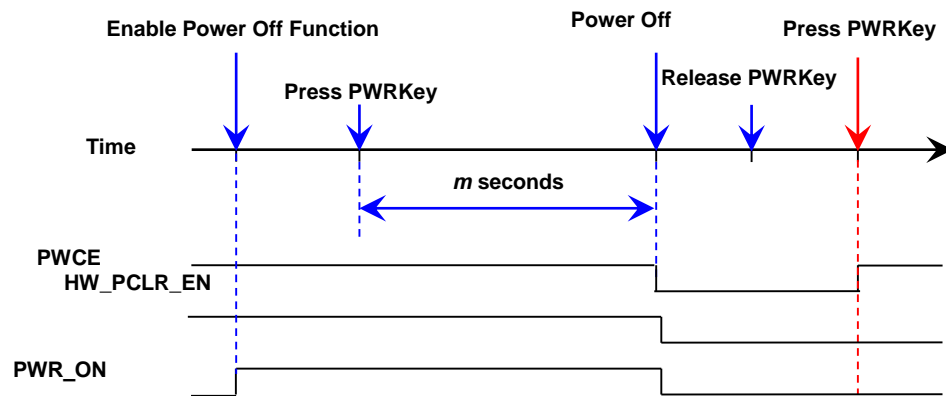
## Force system Power Off Control Flow

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 for pressing the power button to power off is configured in PCLR\_TIME.

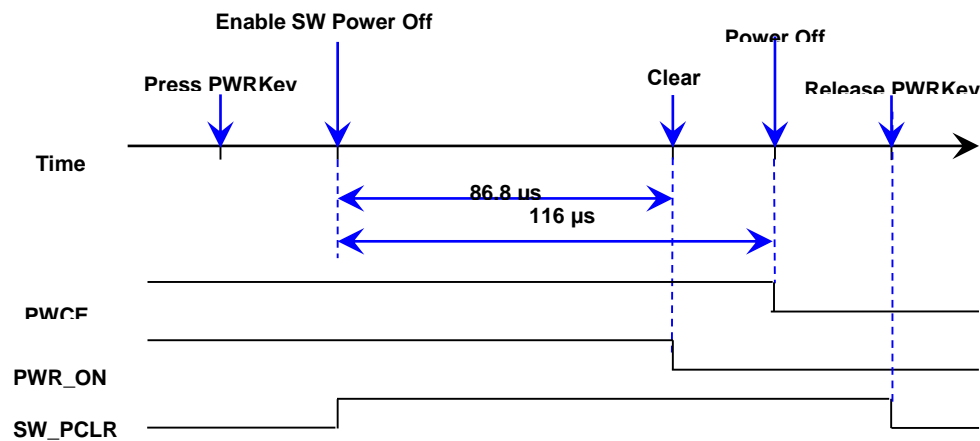
PCLR_TIME Setting	Pressed time to power off	PCLR_TIME Setting	Pressed time to power off
0	Power off right away	8	7~9 seconds
1	0~1 second	9	8~9 seconds
2	1~2 seconds	10	9~10 seconds
3	2~3 seconds	11	10~11 seconds
4	3~4 seconds	12	11~12 seconds
5	4~5 seconds	13	12~13 seconds
6	5~6 seconds	14	13~14 seconds
7	6~7 seconds	15	14~15 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 PWRCE outputs low after 116 $\mu$ s 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	Alarm Interrupt
RTC_TICK_INT	0x02	Tick Interrupt
RTC_PSWI_INT	0x04	Power Switch Interrupt
RTC_ALL_INT	0x07	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_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_FCR_REFERENCE	32761	RTC FRC Reference Value

### 11.2.3 RTC Library Time and Date Definition

The RTC library provides time structure to access RTC time property.

Name	Value	Description
u8cClockDisplay	RTC_CLOCK_12 / RTC_CLOCK_24	12 Hour Clock / 24 Hour Clock
u8cAmPm	RTC_AM / RTC_PM	the AM hours / the PM hours
u32cSecond	0~59	Second value
u32cMinute	0~59	Minute value
u32cHour	1~11 / 0~23	Hour value
u32cDayOfWeek	RTC_SUNDAY~ RTC_SATURDAY	Day of week
u32cDay	1~31	Day value
u32cMonth	1~12	Month value
u32Year	0~99	Year value

## 11.3 RTC Library APIs Specification

### 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, 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\_Write

### Synopsis

UINT32 RTC\_Write (E\_RTC\_TIME\_SELECT eTime, RTC\_TIME\_DATA\_T \*sPt)

### Description

Write 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

```
/* Set the current time */
RTC_Write(RTC_CURRENT_TIME, &sCurTime);
```

### RTC\_WriteEnable

#### Synopsis

UINT32 RTC\_WriteEnable (VOID)

#### Description

Access PW to AER to make access other register enable

#### Parameter

None

#### Return Value

E_SUCCESS	- Success
E_RTC_ERR_EIO	- Access RTC Failed.

#### Example

```
RTC_WriteEnable();
```

### RTC\_SetFrequencyCompensation

#### Synopsis

UINT32 RTC\_SetFrequencyCompensation (FLOAT fnumber)

#### Description

Set Frequency Compensation Data if RTC crystal frequency isn't accurate.

#### Parameter

fnumber                      The actual RTC crystal frequency

#### Return Value

E_SUCCESS	- Success
E_RTC_ERR_FCR_VALUE	- Wrong Compensation value

#### Example

```
/* If actual RTC crystal is 32773.65Hz */
```

```
RTC_SetFrequencyCompensation(32773.65)
```

## 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 supported commands and arguments listed in the table below (Argument 1 is reserved for feature use).

Command	Argument 0	Comment
RTC_IOC_IDENTIFY_LEAP_YEAR	Unsigned integer pointer to store the return leap year value	Get the leap year
RTC_IOC_SET_TICK_MODE	Unsigned integer stores the tick mode data	Set Tick mode
RTC_IOC_GET_TICK	Unsigned integer pointer to store the return tick number	Get the tick counter
RTC_IOC_RESTORE_TICK	None	Restore the tick counter
RTC_IOC_ENABLE_INT	interrupt type	Enable interrupt
RTC_IOC_DISABLE_INT	interrupt type	Disable interrupt
RTC_IOC_SET_CURRENT_TIME	None	Set current time
RTC_IOC_SET_ALAMRM_TIME	None	Set alarm time
RTC_IOC_SET_FREQUENCY	Unsigned integer stores the Frequency Compensation value	Set Frequency Compensation Data
RTC_IOC_SET_PWRON	None	Set Power on
RTC_IOC_SET_PWROFF	None	Set Power off
RTC_IOC_SET_POWER_OFF_PERIOD	Unsigned integer stores the power off period value : 0~15	Set Power Off Period
RTC_IOC_ENABLE_HW_POWEROFF	None	Enable H/W Power Off
RTC_IOC_DISABLE_HW_POWEROF	None	Disable H/W Power Off
RTC_IOC_GET_POWERKEY_STATUS	Unsigned integer pointer to store the return Power Key status	Get Power Key Status
RTC_IOC_SET_PSWI_CALLBACK	The call back function pointer for Power Switch Interrupts	Set Power Switch Interrupt Callback function

### Parameter

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);
```

### Example code

The demo code tests “Time display”, “Alarm” , “Power down Wakeup”, “Software Power Off (Normal Case) Control Flow”, “Hardware Power Off (System Crash) Control Flow”, and “Software Force to Power Off”. Please refer to the RTC sample code of BSP Non-OS.

### 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.

## 12 SIC Library

### 12.1 Overview

N9H20 Non-OS library consists of some 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), 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 N9H20 micro processor.

These libraries are created by using Keil uVision IDE. Therefore, they only can be used in Keil environment.

### 12.2 Storage Interface Controller Library

This library is designed to make user application access N9H20 Storage Interface Controller (SIC) more easily. This interface can directly connect to SD card and NAND Flash.

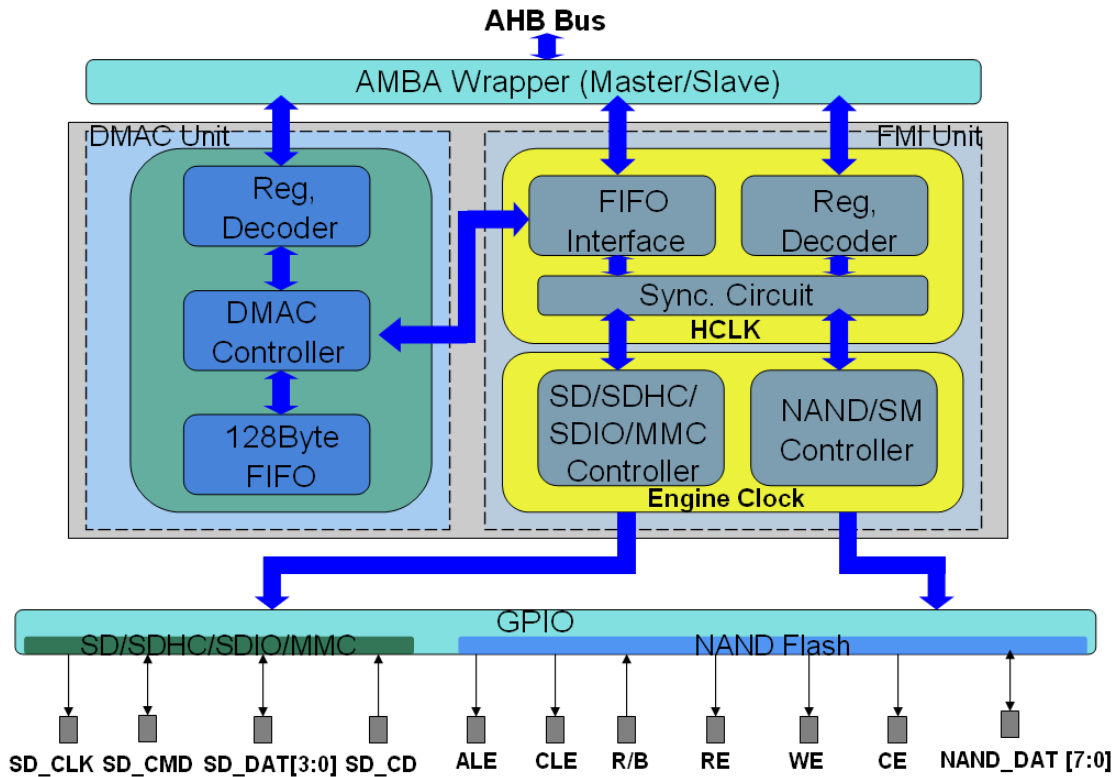
The SIC library has the following features:

- Support single DMA channel and address in non-word boundary.
- Support SD/SDHC/SDIO/MMC card.
- Support SLC and MLC NAND type Flash.
- Adjustable NAND page sizes. (512 / 2048 / 4096 / 8192 bytes + spare area)
- Support up to 4bit/8bit/12bit/15bit hardware ECC calculation circuit to protect data communication.
- Programmable NAND/SM timing cycle.

### 12.3 Programming Guide

#### 12.3.1 System Overview

The Storage Interface Controller (SIC) of N9H20 chip has SIC\_DMAM unit and SIC\_FMI unit. The SIC\_DMAM 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 serial interface controller can support SD/SDHC/SDIO/MMC card and NAND-type flash and the FMI is cooperated with DMAM to provide a fast data transfer between system memory and cards. The block diagram of SIC controller is shown as following:



### 12.3.2 NAND Driver and GNAND Library

The SIC library provides NAND driver API to access NAND chip directly. However, the NAND driver doesn't support management features for NAND chip that doesn'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 use GNAND library before using SIC NAND driver.

## 12.4 SIC APIs Specification

### 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 supported 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_CARD)	SD card remove callback function	SD card insert callback function
SIC_GET_CARD_STATUS	Pointer to return value of SD card status.	None	None
SIC_SET_CARD_DETECT	TRUE to enable SD card detect feature; FALSE to disable it.	None	None

### Parameter

sicFeature SIC\_SET\_CLOCK, SIC\_SET\_CALLBACK,  
SIC\_GET\_CARD\_STATUS, SIC\_SET\_CARD\_DETECT

sicArg0            Depend on feature setting

sicArg1            Depend on feature setting

sicArg2            Depend on feature setting

#### Return Value

For SIC\_GET\_CARD\_STATUS, return TRUE means SD card inserted; return FALSE means SD card removed.

#### Example

Refer to the example code of sicOpen().

## 12.5 SIC / SD APIs Specification

### 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 initializes the SD host interface and programs the SD card from identify mode to stand-by mode.

#### Parameter

None

#### Return Value

>0                    – Total sector number of SD card

Otherwise            – Refer to the error code defined in Error Code Table

#### Example

```
if (sicSdOpen0() <= 0) // Open SD port 0
{
    printf("Error in initializing SD card !! \n");
    sicSdClose0();
    /* 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 closes 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 number which the data is from the address
sdSectorCount	Sector count of this access
sdTargetAddr	The address which data uploads 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];
status = sicSdRead(3000, FMI_TEST_SIZE/512, (UINT32)
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 writes the data into SD card.

### Parameter

sdSectorNo	Sector number which the data puts the address
sdSectorCount	Sector count of this access
sdSourcetAddr	The address which downloads 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];
status = sicSdWrite(3000, FMI_TEST_SIZE/512, (UINT32)fmiFlash_Buf);
```

## 12.6 SIC / NAND APIs Specification

### 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 initialize DMAC and FMI to NAND mode. It also initializes the internal data structure for the future use. Since the different NAND chips need different parameters, nandInit0() also reads the product ID from NAND chip to try to configure correct parameters for it. This function is dependent on NAND chip. 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 initialize it and return to caller.

#### Return Value

0                          - Success  
 Otherwise                - Refer to the error code defined in Error Code Table

#### Example

```
NDISK_T *ptMassNDisk;
NDISK_T MassNDisk;
ptMassNDisk = (NDISK_T *)&MassNDisk;
sicOpen();
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 nandpread0 (INT PBA, INT page, UINT8 \*buff) for NAND chip 0  
 INT nandpread1 (INT PBA, INT page, UINT8 \*buff) for NAND chip 1

#### Description

This function reads a page of data from NAND.

#### Parameter

PBA	physical block address of NAND which data is 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 to the 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 = nandpread0(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 writes a page of data to NAND.

### Parameter

PBA	physical block address of NAND which the data is written to
page	page number in PBA block to write data.
buff	the RAM address to get the writing data.

### Return Value

0	- Success
Otherwise	- Refer to the 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 checks the redundancy area of the NAND page and return the dirty status to indicate whether a page to be 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
page	page number in PBA block which checks 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 checks the redundancy area of the NAND block and return the valid status to indicate whether a block to be 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 an invalid block always since it could be a bad block.

### Parameter

PBA	physical block address of NAND which checks the valid
status	

### Return Value

0	- Invalid block that cannot write data into it
1	- Valid block that can write data into it directly or indirectly

### Example

```
/* check valid status for NAND block 5 */
status = nand_is_valid_block0(5);
if (status == 1)
{
    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 erases 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 which is erased
-----	--

### Return Value

0	- Erase block successfully
Otherwise	- Refer to the 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 erases all blocks in NAND chip. All data in chip will be lost.

### Parameter

None

### Return Value

- 0 - Erase chip successfully
- Otherwise - Refer to the 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");
}
```

## 12.6.1 Example code

The example code tests the NAND flash and SD card by using the read/write/compare, please refer to the SIC example code of BSP Non-OS.

## 12.6.2 Error Code Table

Code Name	Value	Description
FMI_TIMEOUT	0xFFFF0101	Access timeout
FMI_NO_MEMORY	0xFFFF0102	No available memory
Error Code for SD Card		
FMI_NO_SD_CARD	0xFFFF0110	No SD card insert
FMI_ERR_DEVICE	0xFFFF0111	Unknown device type
FMI_SD_INIT_TIMEOUT	0xFFFF0112	SD command/response timeout
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	SD command/response error
FMI_SD_CRC16_ERROR	0xFFFF0117	Data transfer error
FMI_SD_CRC_ERROR	0xFFFF0118	Data transfer error
FMI_SD_CMD8_ERROR	0xFFFF0119	SD command 8 response error
Error Code for NAND Flash		
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

## 13 SPI Library

### 13.1 Overview

This library provides APIs for programmers to access SPI device connecting with N9H20 SPI interfaces. The SPI library will get the APB clock frequency from system library; application must set the CPU clock before using SPI library.

### 13.2 SPI Library APIs Specification

#### spiOpen

##### Synopsis

```
int spiOpen(SPI_INFO_T *pInfo)
```

##### Description

This function initializes the SPI interface.

##### Parameter

```
typedef struct _spi_info_t
{
    INT32  nPort;           /* select SPI0 (0) or SPI1 (1) */
    BOOL   blsSlaveMode;    /* set the interface mode - master mode or
                             slave mode */
    BOOL   blsClockIdleHigh; /* set the clock idle state – high or low */
    BOOL   blsLSBFirst;     /* set LSB transfer first or MSB first */
    BOOL   blsAutoSelect;   /* set automatically active / inactive CS pin
    */
    BOOL   blsActiveLow;    /* define the active level of device select
                             signal */
    BOOL   blsTxNegative;   /* set the Tx signal changed on rising edge
    or
                             falling edge */
    BOOL   blsLevelTrigger; /* set slave select signal is edge-trigger or
                             level-trigger */
} SPI_INFO_T;
```

##### Return Value

```
= 0      Success
< 0      Fail
```

##### Example

```
/* Set SPI0 as master mode, clock idle low, MSB transfer first, not
auto CS, receive on negative edge, slave select signal is active Low
and edge-trigger */
```

```
SPI_INFO_T spi0;
spi0.nPort = 0;
spi0.bIsSlaveMode = FALSE;
spi0.bIsClockIdleHigh = FALSE;
spi0.bIsLSBFirst = FALSE;
spi0.bIsAutoSelect = FALSE;
spi0.bIsActiveLow = TRUE;
spi0.bIsTxNegative = FALSE;
spi0.bIsLevelTrigger = FALSE;
spiOpen(&spi0);
```

## 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 spiloctl(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

None

### Example

```
/* apb clock is 48MHz, output clock is 10MHz */
spiIoctl(0, SPI_SET_CLOCK, 48, 10000);
```

## spiEnable

### Synopsis

INT spiEnable(INT32 spiPort)

### Description

The function will be 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 be 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 and store it into the buffer pDst.

#### Parameter

port	select SPI0 (0) or SPI1 (1)
RxBitLen	set the receive bit length. <i>SPI_8BIT, SPI_16BIT, SPI_32BIT</i>
len	data count. SPI_8BIT is byte count; SPI_16BIT is half-word count;
	SPI_32BIT is word count.
pDst	The buffer stores the read back data.

#### 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. <i>SPI_8BIT, SPI_16BIT, SPI_32BIT</i>
len	data count. SPI_8BIT is byte count; SPI_16BIT is half-word count;
	SPI_32BIT is word count
pSrc	The buffer stores the data that is written into SPI interface.

#### Return Value

0 Success

#### Example

```
/* write 1 half-word to SPI device */
wdata = 0x80ff;
spiWrite(0, SPI_16BIT, 1, (CHAR *)&wdata);
```

### spiTransfer

### Synopsis

INT spiTransfer(UINT32 port, UINT32 TxBitLen, UINT32 len, PUINT8 RxBuf , PUINT8 TxBuf)

### Description

This function is used to read/write the data from/to the SPI interface.

### Parameter

port	select SPI0 (0) or SPI1 (1)
TxBitLen	set the transmit bit length. <i>SPI_8BIT, SPI_16BIT, SPI_32BIT</i>
len	data count. SPI_8BIT is byte count; SPI_16BIT is half-word count; SPI_32BIT is word count.
RxBuf	The buffer stores the read back data.
TxBuf	The buffer stores the data that is written into SPI interface.

### Return Value

0 Success

### Example

```
/* write 1 byte to SPI device and read 1 byte data from SPI device */
spiRead(0, SPI_8BIT, 1, (PUINT8)&rdata, (PUINT8)&wdata);
```

## spiSSEnable

### Synopsis

INT spiSSEnable(UINT32 spiPort, UINT32 SSPin, UINT32 ClockMode)

### Description

The function will be set specified SS pin of the SPI interface to active state and set transfer timing.

### Parameter

spiPort	select SPI0 (0) or SPI1 (1)
SSPin	select SS0 (0) or SS1 (1)
ClockMode	Decides the transfer timing. (SPI_CLOCK_MODE 0~3 )

### Return Value

0 Success

### Example

```
/* Set SS0 pin of SPI0 to active state and transfer timing as SPI mode 3 */
#define SPI_CLOCK_MODE 3
```

```
spiSSEnable(0, 0, SPI_CLOCK_MODE);
```

## spiSSDisable

### Synopsis

INT spiSSDisable(UINT32 spiPort, UINT32 SSPin)

### Description

The function will be set specified SS pin of the SPI interface to inactive state.

### Parameter

spiPort	select SPI0 (0) or SPI1 (1)
SSPin	select SS0 (0) or SS1 (1)

### Return Value

0 Success

### Example

```
/* Set SS0 pin of SPI0 to inactive state */
spiSSDisable(0, 0);
```

## spiInstallCallBack

### Synopsis

INT32 spiInstallCallBack(UINT8 u8Port, PFN\_DRV\_SPI\_CALLBACK  
pfncallback, PFN\_DRV\_SPI\_CALLBACK \*pfnOldcallback)

### Description

This function is used to install the specified SPI port 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

```
/* Install Callback function */
spiInstallCallBack (0, SPI0_Handler, &pfnOldcallback);
```

## spilsBusy

### Synopsis

BOOL spIsBusy(UINT8 u8Port)

#### Description

This function is used to get the SPI busy state.

#### Parameter

u8Port                      select SPI0 (0) or SPI1 (1)

#### Return Value

TURE                      - The specified SPI is busy.  
FALSE                      - The specified SPI is not busy.

#### Example

```
/* Wait for SPI0 transfer finish */
while(spiIsBusy(0) != FALSE);
```

### 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

```
/* Enable Interrupt Sources of SPI0 */
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

```
/* Disable Interrupt Sources of SPI0 */
spiDisableInt(0);
```

## spiSetGo

### Synopsis

VOID spiSetGo(UINT8 u8Port)

### Description

This function is used to set SPI starts to transfer data.

### Parameter

u8Port                      select SPI0 (0) or SPI1 (1)

### Return Value

None

### Example

```
/* Set SPI0 starts to transfer data */
spiSetGo(0);
```

## spiSetByteEndin

### Synopsis

VOID spiSetByteEndin(UINT8 u8Port, E\_DRVSPi\_OPERATION eOP)

### Description

This function is used to enable or disable byte endin.

### Parameter

u8Port                      Select SPI0 (0) or SPI1 (1)  
eOP                          Select enable or disable the byte endin

### Return Value

None

### Example

```
/* Set SPI0 disable byte endin */
spiSetByteEndin(0, eDRVSPi_DISABLE);
```

## 14 SPI\_SecureIC Library

### 14.1 Overview

This library provides APIs for programmers to access Winbond SPI Flash (Only for W74M) 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.

### 14.2 API Functions

#### RPMC\_ReadJEDECID

##### Synopsis

INT32 RPMC\_ReadJEDECID(PUINT8 data)

##### Description

This function reads JEDEC ID.

##### Parameter

data                      the JEDEC ID pointer

##### Return Value

= 0                      Success

< 0                      Fail

##### Example

```
if ((RPMC_ReadJEDECID(u8JID)) == -1)
{
    sysprintf("read id error !!\n");
    return -1;
}
```

#### RPMC\_ReadUID

##### Synopsis

INT16 RPMC\_ReadUID (PUINT8 data)

##### Description

This function reads Unique ID.

##### Parameter

data                      the Unique ID pointer

##### Return Value

= 0                      Success

< 0                      Fail

**Example**

```
if ((RPMC_ReadUID(u8UID)) == -1)
{
    sysprintf("read id error !!\n");
    return -1;
}
```

**RPMC\_ReadCounterData****Synopsis**

unsigned int RPMC\_ReadCounterData(void)

**Description**

This function read counter data in stoage in public array counter[], data is available if RPMC\_IncCounter() operation succeeded

**Parameter**

None

**Return Value**

Counter data

**Example**

```
/* counter data in stoage in public array counter[], data is
available if RPMC_IncCounter() operation succeeded */
RPMC_counter = RPMC_ReadCounterData();
```

**RPMC\_WrRootKey****Synopsis**

unsigned int RPMC\_WrRootKey(unsigned int cadr,unsigned char \*rootkey)

**Description**

This function writes root key to W74M.

**Parameter**

cadr	Key index (0~3)
rootkey	Root key pointer

**Return Value**

0x80	Success
Other	Fail

**Example**

```

RPMCStatus = RPMC_WrRootKey(KEY_INDEX, ROOTKey);          /* initial
Rootkey, use first rootkey/counter pair */
if(RPMCStatus == 0x80)
{
    /* Write rootkey success */
    sysprintf("RPMC_WrRootKey Success - 0x%02X!!\n",RPMCStatus );
}
else
{
    /* write rootkey fail, check datasheet for the error bit */
    sysprintf("RPMC_WrRootKey Fail - 0x%02X!!\n",RPMCStatus );
}

```

## RPMC\_UpHMACkey

### Synopsis

unsigned int RPMC\_UpHMACkey(unsigned int cadr,unsigned char  
\*rootkey,unsigned char \*hmac4,unsigned char \*hmackey);

### Description

This function updates HMAC Key

### Parameter

cadr	Key index (0~3)
rootkey	Root key pointer
hmac4	HMAC Key message pointer
hmackey	HMAC Key pointer (generated by Rootkey and hmac4)

### Return Value

0x80	Success
Other	Fail

### Example

```

RPMCStatus = RPMC_UpHMACkey(KEY_INDEX, ROOTKey, HMACMessage,
HMACKey);
if(RPMCStatus == 0x80)
{
    /* update HMACkey success */
    sysprintf("RPMC_UpHMACkey Success - 0x%02X!!\n",RPMCStatus );
}
else

```

```
{
    /* write HMACkey fail, check datasheet for the error bit */
    sysprintf("RPMC_UpHMACkey Fail - 0x%02X!!\n",RPMCStatus );
}
```

## RPMC\_IncCounter

### Synopsis

```
unsigned int RPMC_IncCounter(unsigned int cadr,unsigned char
*hmackey,unsigned char *input_tag);
```

### Description

This function is used to increase counter data

### Parameter

cadr	Key index (0~3)
hmackey	HMAC Key pointer (generated by Rootkey and input_tag)
input_tag	Message for increase counter

### Return Value

0x80	Success
Other	Fail

### Example

```
RPMCStatus = RPMC_IncCounter(KEY_INDEX, HMACKey, Input_tag);
if(RPMCStatus == 0x80)
{
    /* increase counter success */
    sysprintf("RPMC_IncCounter Success - 0x%02X!!\n",RPMCStatus );
}
else
{
    /* increase counter fail, check datasheet for the error bit */
    sysprintf("RPMC_IncCounter Fail - 0x%02X!!\n",RPMCStatus );
}
```

## RPMC\_Challenge

### Synopsis

```
unsigned char RPMC_Challenge(unsigned int cadr,unsigned char
*hmackey,unsigned char *input_tag);
```

### Description

This function is used to do authentication check.

### Parameter

cadr	Key index (0~3)
hmackey	HMAC Key pointer (generated by Rootkey and input_tag)
input_tag	Message for authentication

### Return Value

0x80	Success
Other	Fail

### Example

```
/* Main security operation call challenge*/
while(1)
{
    if(RPMC_Challenge(KEY_INDEX, HMACKey, Input_tag)!=0)
    {
        sysprintf("RPMC_Challenge Fail!!\n" );
        /* return signature miss-match */
        return 0;
    }
    else
    {
        if(count > 500)
        {
            sysprintf("\n" );
            RPMCStatus = RPMC_IncCounter(KEY_INDEX, HMACKey,
Input_tag);
            if(RPMCStatus == 0x80)
            {
                /* increase counter success */
                sysprintf("RPMC_IncCounter Success -
0x%02X!!\n",RPMCStatus );
            }
            else
            {
                /* increase counter fail, check datasheet for the
error bit */
                sysprintf("RPMC_IncCounter Fail -
0x%02X!!\n",RPMCStatus );
                while(1);
            }
        }
    }
}
```

```
        /* counter data in stoage in public array counter[],
data is available if RPMC_IncCounter() operation succeeded */
        RPMC_counter = RPMC_ReadCounterData();

        /* increase RPMC counter done*/
        sysprintf("RPMC_counter 0x%X\n",RPMC_counter);
        count = 0;
    }
    else
        count++;
    sysprintf("." );
}
}
```

## 15 SPU Library

### 15.1 Overview

This library provides APIs for programmers to 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.

### 15.2 SPU Library APIs Specification

#### 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    The pointer for Call back function

data                      The pointer for Source PCM audio data

##### 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 ();
```

## spuloctl

### Synopsis

VOID spuloctl(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	The first argument of the command
arg1	The 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 be 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 ();
```

## 16 I2S Library

### 16.1 overview

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

### 16.2 APIs Specification

### 16.3 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_DRVI2S_PLAY* psPlayStruct  
)
```

## DrvI2S\_EnableInt

### Synopsis

VOID DrvI2S\_EnableInt (UINT32 u32InterruptFlag, PFN\_DRVI2S\_CB\_FUNC\* pfnCallBack)

### Description

Enable the selected interrupt and setup the callback function, respectively.

### Parameter

u32InterruptFlag    selected interrupt  
pfnCallBack                callback function

### Return Value

None

### Example

```
DrvI2S_EnableInt (DRVI2S_IRQ_PLAYBACK, audio_PlayCallBack);
```

## DrvI2S\_DisableInt

### Synopsis

VOID DrvI2S\_DisableInt (UINT32 u32InterruptFlag)

### Description

Disable the related interrupt

### Parameter

u32InterruptFlag    selected interrupt

### Return Value

None

### Example

```
DrvI2S_DisableInt (DRVI2S_IRQ_PLAYBACK);
```

## DrvI2S\_ClearInt

### Synopsis

VOID DrvI2S\_ClearInt (UINT32 u32InterruptFlag)

### Description

Clear the related interrupt flag

### Parameter

u32InterruptFlag    interrupt flag

#### Return Value

None

#### Example

```
DrvI2S_ClearInt (DRVI2S_IRQ_RECORD);
```

### DrvI2S\_PollInt

#### Synopsis

```
VOID DrvI2S_PollInt (UINT32 u32InterruptFlag)
```

#### Description

Get the status of related interrupt flag

#### Parameter

u32InterruptFlag    selected interrupt flag

#### Return Value

Current status of selected interrupt flag

#### Example

```
IntStatus = DrvI2S_PollInt (DRVI2S_IRQ_RECORD);
```

### DrvI2S\_StartPlay

#### Synopsis

```
VOID DrvI2S_StartPlay(S_DRVI2S_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_DRVI2S_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_DRVI2S_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_DRVI2S_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_DRVI2S_SAMPLING eSamplaerate)
```

### Description

Set Play/Record sampling rate.

### Parameter

eSampleRate      Given sampling rate.

### Return Value

E\_SUCCESS      set up the sampling rate successfully

E\_FAIL              fail in setting up the sampling rate

### Example

```
DrvI2S_SetSampleRate((E_DRVI2S_SAMPLING) eDRVI2S_FREQ_44100
```

## 17 System Library

### 17.1 Overview

The N9H20 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 N9H20 demo board or evaluation board.

This library is created by using Keil uVision IDE. Therefore, it only can be used in Keil environment.

### 17.2 System Library APIs Specification

#### 17.3 Timer Functions

##### 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 or timer 1. The event function which indicated by *uTimeEventNo* shall be cleared.

###### Parameter

nTimeNo   TIMER0, TIMER1

uTimeEventNo   Event number which want to clear

###### 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 count. When interrupt occurred, the system will reset after 1024 clock cycles. Clear the 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 the selected interrupt to be FIQ or IRQ, and level group 0 ~ 7. *pvNewISR* is the pointer of own interrupt service routine.

### Parameter

nIntTypeLevel FIQ\_LEVEL\_0, IRQ\_LEVEL\_1 ~ IRQ\_LEVEL\_7  
pvNewISR The pointer of watch dog timer interrupt service routine

### Return value

The pointer which points 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.

### 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);
```

## 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 or timer 1. The event function which pointed by *pvFun* shall be executed after *nTimeTick* system timer tick.

### Parameter

nTimeNo   TIMER0, TIMER1  
nTimeTick   Tick count before event executed  
pvFun       Event function pointer

### Return Value

event number

### Example

```
/* Set event function "hello" after 100 tick */
INT nEventNo;
VOID hello(VOID)
{
    sysPrintf("Hello World!\n");
}
nEventNo = sysSetTimerEvent (TIMER0, 100, (PVOID)hello);
```

## sysSetTimerReferenceClock

### Synopsis



```
INT32 sysSetTimerReferenceClock(UINT32 nTimeNo, UINT32 uClockRate);
```

### Description

This function used to set the reference clock of timer. The default reference clock is system clock (15MHz).

### 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.

nWdtInterval	Interrupt Timeout	Reset Timeout	Real Time Interval
WDT_INTERVAL_0	$2^{14}$ clocks	$2^{14} + 1024$ clocks	0.28 sec.
WDT_INTERVAL_1	$2^{16}$ clocks	$2^{16} + 1024$ clocks	1.12 sec.
WDT_INTERVAL_2	$2^{18}$ clocks	$2^{18} + 1024$ clocks	4.47 sec.
WDT_INTERVAL_3	$2^{20}$ clocks	$2^{20} + 1024$ clocks	17.9 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

sysStartTimer will start Timer 0 or Timer 1. *nTimeNo* is used to select timer 0 or timer 1. Because W90P710 timer has three operation modes, the *nOpMode* is used to set the operation mode. *uTicksPerSecond* indicates that how many ticks per second.

### Parameter

nTimeNo TIMER0, TIMER1

nTickPerSecond Tick numbers per second

nOpMode ONE\_SHOT\_MODE, PERIODIC\_MODE, TOGGLE\_MODE

### 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 Timer 0 or Timer 1. *nTimeNo* is used to select timer 0 or timer 1. After disabling timer, this function will restore the interrupt service routine.

### Parameter

nTimeNo TIMER0, TIMER1

### 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     The value of tick counter

### Return Value

Successful

### Example

```
/* update timer 0's tick counter as 3000 */  
sysUpdateTickCount (TIMER0, 3000);
```

## 17.4 UART Function

### sysGetChar

#### Synopsis

```
CHAR sysGetChar(VOID);
```

#### Description

This function is used to obtain the next available character from the UART. Nothing is echoed. When no any available character is found, the function waits until a character is found from UART.

#### Parameter

None

#### Return Value

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 follows:

```
typedef struct UART_INIT_STRUCT
{
    UINT32 freq;
    UINT32 baud_rate;
    UINT32 data_bits;
    UINT32 stop_bits;
    UINT32 parity;
    UINT32 rx_trigger_level;
} WB_UART;
```

*uart->freq* is UART reference clock. Default is 15MHz. If user sets the different reference clock, use this parameter to change it.

*uart->baud\_rate* is used to set the baudrate of COM port. 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->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
```

## Return Value

Successful/ WB\_INVALID\_PARITY/ WB\_INVALID\_DATA\_BITS/  
WB\_INVALID\_STOP\_BITS/ WB\_INVALID\_BAUD

## Example

```
WB_UART_T uart;
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 wants 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 wants to display

### Return Value

None

### Example

```
sysPutChar("A");
```

## sysUartEnableInt

### Name

sysUartEnableInt– Enable high speed UART time out or data ready interrupt.

### Synopsis

```
VOID sysUartEnableInt (INT32 eIntType );
```

### Description

The function enables the specified UART interrupt.

### Parameter

eIntType:

Table 17-1 UART Interrupt Type

u32IntType	Value	Meaning
UART_INT_RDA	1	Data has ready in buffer.
UART_INT_RDTO	2	A moment that over the time of time out after receive UART data
UART_INT_NONE	255	Disable UART interrupt

### Return Value

None

### Example

```
sysUartEnableInt(UART_INT_RDA);
/* Enable receive data ready interrupt */
sysUartEnableInt(UART_INT_RDTO);
/* Enable receive time out interrupt */
```

## sysUartInstallcallback

### Name

sysUartInstallcallback – install callback function for upper to process UART events.

### Synopsis

```
VOID sysUartInstallcallback (UINT32 u32IntType,
PFN_SYS_UART_CALLBACK fnCallback);
```

### Description

Generally, the function is used for upper layer to processing the UART events.

### Parameter

u32IntType	UART interrupt type. Please reference Table 17-1 UART Interrupt Type
------------	--

### Return value

None

### Return Value

None

### Example

```
void UartDataValid_Handler(UINT8* buf, UINT32 u32Len)
{
    UINT32 u32Idx = 0;
    g_u32Len = u32Len;
    g_u32Valid = g_u32Valid+1;
    memcpy(&(pi8UartBuf[g_u32Idx]), buf, u32Len);
    g_u32Idx = g_u32Idx+u32Len;
    while(u32Idx++<u32Len)
    {
        if(*buf++ == 'q')
        {
            bIsTimeOut = 1;
            break;
        }
    }
}

void UartDataTimeOut_Handler(UINT8* buf, UINT32 u32Len)
{

```

```

UINT32 u32Idx = 0;
g_u32Timeout = g_u32Timeout+1;
memcpy(&(pi8UartBuf[g_u32Idx]), buf, u32Len);
g_u32Idx = g_u32Idx+u32Len;

while(u32Idx++<u32Len)
{
    if(*buf++ == 'q')
    {
        bIsTimeOut = 1;
        break;
    }
}
}
int DemoAPI_HUART(void)
{
...
    sysUartInstallcallback(0, UartDataValid_Handler);
    sysUartInstallcallback(1, UartDataTimeOut_Handler);
...
}

```

## sysUartTransfer

### Name

sysUartTransfer – Transfer mass data through UART.

### Synopsis

VOID sysUartTransfer (char\* pu8buf, UINT32 u32Len);

### Description

Transfer mass data through UART port.

### Parameter

pu8bufBuffer	Pointer for transferring data.
u32Len	Length for transferring data. Unit: Byte.

### Return value

None

### Example

```
int DemoAPI_HUART(void)
```



```
{
...
    sysUartTransfer(pi8UartBuf, u32Count);
    /* Transfer out u32Count byte in i8UartBuf */
...
}
```

## register\_uart\_device

### Name

register\_uart\_device – Register UART device

### Synopsis

```
INT32 register_uart_device(UINT32 u32port, UARTDEV_T* pUartDev);
```

### Description

Registered an UART device through the API. Programmer could use same API interface to operation both UART devices in the same time.

### Parameter

u32port	Specified UART port number for registering.
pUartDev	A pointer for registering UART port. The pointer points to the APIs interface

### Return value

None

### Example

```
UARTDEV_T UART0; /*High speed */
UARTDEV_T* pUART0;
int DemoAPI_HUART(void)
{
...
    register_uart_device(0, &UART0);
    pUART0 = &UART0;
    pUART0->UartPort(0);
    uart.uiFreq = u32ExtFreq*1000;
    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;
```

```
pUART0->UartInitialize(&uart);
pUART0->UartEnableInt(UART_INT_NONE);
pUART0->UartInstallcallback(0, UartDataValid_Handler);
pUART0->UartInstallcallback(1, UartDataTimeOut_Handler);
...
}
```

## 17.5 AIC Functions

### 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

#### 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.

#### 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 interrupt which saves in AIC\_IMR register.

### Parameter

None

### Return Value

value of AIC\_IMR register

### Example

```
/* Set AIC as software mode */
UINT32 uIMRValue;
uIMRValue = sysGetInterruptEnableStatus();
```

## 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
pNewHandler	Pointer of the new handler

### Return Value

a pointer which points 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);
```

## sysInstallIrrqHandler

### Synopsis

PVOID sysInstallIrrqHandler(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 points to old ISR

#### Example

```
/* Setup own IRQ handler */
PVOID oldVect;
oldVect = sysInstallIrqHandler(pNewIrqISR);
```

### sysInstallISR

#### Synopsis

```
PVOID sysInstallISR(INT32 intTypeLevel, INT32 intNo, PVOID pNewISR,
PVOID pParam);
```

#### Description

W90P710 interrupt group 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  
intNo      Interrupt source number  
pNewISR      Function pointer of new ISR  
pParam      Parameter for ISR

#### Return Value

a pointer which point to old ISR

#### Example

```
/* Setup timer 0 handler */
PVOID oldVect;
oldVect = sysInstallISR(IRQ_LEVEL_1, 7, pTimerISR, param);
```

### 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**

Enable / disable all interrupt sources.

**Parameter**

intState    ENABLE\_ALL\_INTERRUPTS, DISABLE\_ALL\_INTERRUPTS

**Return Value**

Successful

**Example**

```
/* Disable all interrupt */  
INT32 status;  
status = sysSetGlobalInterrupt(DISABLE_ALL_INTERRUPTS);
```

**sysSetInterruptPriorityLevel****Synopsis**

```
INT32 sysSetInterruptPriorityLevel(UINT32 intNo, UINT32 intLevel);
```

**Description**

W90P710 interrupt has 8 group levels. The highest is 0, and the lowest is 7.  
Use this function can change the priority level after installing the ISR.

**Parameter**

intNo       Interrupt source number

intLevel    FIQ\_LEVEL\_0, IRQ\_LEVEL\_1 ~ IRQ\_LEVEL\_7

**Return Value**

Successful or Fail.

### Example

```
/* Change timer 0 priority to level 4 */
INT32 status;
status = sysSetInterruptPriorityLevel(7, 4);
```

## sysSetInterruptType

### Synopsis

```
INT32 sysSetInterruptType(UINT32 intNo, UINT32 intSourceType);
```

### Description

W90P710 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

intSourceType    LOW\_LEVEL\_SENSITIVE, HIGH\_LEVEL\_SENSITIVE,  
NEGATIVE\_EDGE\_TRIGGER, POSITIVE\_EDGE\_TRIGGER

### Return Value

Successful or Fail.

### Example

```
/* Change timer 0 source type to be positive edge trigger */
INT32 status;
status = sysSetInterruptType(7, POSITIVE_EDGE_TRIGGER);
```

## sysSetLocalInterrupt

### Synopsis

```
INT32 sysSetLocalInterrupt(INT32 intState);
```

### Description

When using interrupt, the CPSR I bit and F bit need to be enabled or disabled. This function is used to enable / disable I bit and F bit.

### Parameter

intState    ENABLE\_IRQ, ENABLE\_FIQ, ENABLE\_FIQ\_IRQ, DISABLE\_IRQ,  
DISABLE\_FIQ, DISABLE\_FIQ\_IRQ

### Return Value

Successful

### Example

```
/* Enable I bit of CPSR */
INT32 state;
state = sysSetLocalInterrupt(ENABLE_IRQ);
```

## 17.6 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      I\_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);
```

## 17.7 Clock Control Function

### sysGetExternalClock

#### Synopsis

```
UINT32 sysGetExternalClock(void);
```

#### Description

This function is used to get external clock setting. N9H20 IBR only supports 2 kinds of external clock frequency. 12MHz or 27MHz. So external clock will be 12MHz or 27MHz.

#### Parameter

None

#### Return Value

External clock. Unit : KHz.

#### Example

```
/* Read system clock setting */
UINT32 u32ExtFreq;

u32ExtFreq = sysGetExternalClock();
```

### sysSetSystemClock

#### Synopsis

```
UINT32 sysSetSystemClock(E_SYS_SRC_CLK eSrcClk,
                        UINT32 u32PllKHz,
                        UINT32 u32SysKHz,
                        UINT32 u32CpuKHz,
                        UINT32 u32HclkKHz,
                        UINT32 u32ApbKHz);
```

#### Description

This function is used to write system clock setting includes PLL output frequency, System, CPU, AHB and APB clock.

#### Parameter

eSrcClk Sytem clock source.

It could be eSYS\_EXT, eSYS\_APLL and eSYS\_UPLL. They mean the system clock source comes from external clock, APLL and UPLL respectively.

u32PllKHz Set the APLL or UPLL output frequency. Unit: KHz.

u32SysKHz Set the system clock output frequency.

Unit: KHz. The system clock source can be external, APLL or UPLL.

u32CpuKHz Set the CPU working frequency.

Unit: KHz.

u32HclkKHz Set the DDR/SDRAM working frequency.

Unit: KHz. It is always equal to u32SysClk/2

u32ApbKHz Set the APB output frequency.

Unit: KHz.

There are some limitations in the clock function due to hardware's limitation.

1. These clocks exist multiplication factor. It means  $PLL \geq n * SYS$ .  $SYS \geq m * CPU$ .  $SYS = 2 * HCLK$ .  $SYS \geq x * APB$ . Where n, m, x are all integer.
2. PLL clock must under or equal to 240MHz.
3. System clock must under or equal to the source clock.
4. CPU clock source is system clock. It can only be equal to system clock or half of system clock.
5. HCLK clock can only be half of system clock.

APB clock source comes from system clock. It is can only smaller than system clock.

#### Return Value

Successful or Error code.

#### Example

```
/* Write system clock setting */
sysSetSystemClock(eSYS_UPLL,          // system clock come from UPLL
                  240000,              // UPLL = 240MHz
                  240000,              // SYS = 240MHz
                  120000,              // CPU = 120MHz
                  120000,              // HCLK = 120MHz,
                  60000);              // APB = 60MHz
```

#### sysGetSystemClock

#### Synopsis

```
void sysGetSystemClock(E_SYS_SRC_CLK* peSrcClk,
    PUINT32 pu32PllKHz,
    PUINT32 pu32SysKHz,
    PUINT32 pu32CpuKHz,
    PUINT32 pu32HclkKHz,
    PUINT32 pu32ApbKHz);
```

### Description

This function is used to read system clock setting including PLL output frequency, System, CPU, AHB and APB clock. The function must be called after function-sysSetSystemClock. The clocks are just only a record after function- sysGetSystemClock.

### Parameter

peSrcClk Sytem clock source.

It could be eSYS\_EXT=0, eSYS\_APLL= 2 and eSYS\_UPLL = 3.

pu32PllKHz : APLL or UPLL output frequency.

Unit: KHz.

pu32SysKHz System clock output frequency.

Unit: KHz. The system clock source can be external, APLL or UPLL.

pu32CpuKHz CPU working frequency.

Unit: KHz.

pu32HclkKHz DDR/SDRAM working frequency.

Unit: KHz.

pu32ApbKHz APB output frequency.

Unit: KHz.

### Return Value

None.

### Example

```
/* Write system clock setting */
E_SYS_SRC_CLK eSrcClk;
UINT32 u32PllKHz, u32SysKHz, u32CpuKHz, u32HclkKHz, u32ApbKHz;
sysSetSystemClock(eSYS_UPLL, // system clock come from UPLL
    240000, // UPLL = 240MHz
    240000, // SYS = 240MHz
    120000, // CPU = 120MHz
    120000, // HCLK = 120MHz,
    60000); // APB = 60MHz
/* Read system clock setting */
```

```
sysGetSystemClock(&eSrcClk,
                  &u32P11KHz,
                  &u32SysKHz,
                  &u32CpuKHz,
                  &u32HclkKHz,
                  &u32ApbKHz);
```

## sysSetPllClock

### Synopsis

```
UINT32 sysSetPllClock(E_SYS_SRC_CLK eSrcClk,
                     UINT32 u32TargetKHz);
```

### Description

There are two PLLs in N9H20. User can assign one PLL as system clock source. The other PLL can be assigned the output frequency through the function.

### Parameter

eSrcClk: eSYS\_APLL = 2 or eSYS\_UPLL = 3.

u32TargetKHz: Target PLL output frequency. Unit : KHz.

### Return Value

Specified PLL output frequency. Unit : KHz. The return value may not be the same as the specified value due to hardware's limitation. If it could not meet the hardware SPEC, library will automatically search the nearest frequency.

### Example

```
/* Write system clock setting */
E_SYS_SRC_CLK eSrcClk;
sysSetSystemClock(eSYS_UPLL,          // system clock come from UPLL
                  240000,              // UPLL = 240MHz
                  240000,              // SYS = 240MHz
                  120000,              // CPU = 120MHz
                  120000,              // HCLK = 120MHz,
                  60000);              // APB = 60MHz

/*Specified APLL clock */
sysSetPllClock(eSYS_APLL,
               192000);
```

## sysClockDivSwitchStart

### Synopsis

```
INT32 sysClockDivSwitchStart(UINT32 u32SysDiv);
```

### Description

The function is used to set system divider quickly. It doesn't change PLL clock. If the system divider is not zero after the function was called. User need to calculate the u32SysDiv by himself after call function-sysSetSystemClock().

### Parameter

U32SysDiv            System divider. The value is from 0 ~ 7.

### Return Value

Sussessful.

### Example

```
/* Write system clock setting */
E_SYS_SRC_CLK eSrcClk;
sysSetSystemClock(eSYS_UPLL,            // system clock come from UPLL
                  240000,            // UPLL = 240MHz
                  240000,            // SYS = 240MHz
                  120000,            // CPU = 120MHz
                  120000,            // HCLK = 120MHz,
                  60000);            // APB = 60MHz

/* Specified APLL clock */
sysSetPllClock(eSYS_APLL,
               192000);
sysClockDivSwitchStart(240000/48000-1); /* Change system clock to 48MHz
*/
```

## sysCheckPllConstraint

### Synopsis

```
VOID sysCheckPllConstraint (BOOL blsCheck);
```

### Description

There are some constraints in PLL formula. The function is used to enable or disable these constraints.

### Parameter

blsCheck TRUE:

Check PLL constraints as call function-sysSetSystemClock or sysSetPllClock.

FALSE:

Not to check PLL constraints as call function-sysSetSystemClock or sysSetPllClock.

### Return Value

None.

### Example

```
sysCheckPllConstraint(FALSE); // Not to check PLL constraint
sysSetPllClock(eSYS_APLL,      // APLL for audio recording,
               153600);        // UINT32 u32PllKHz,
sysCheckPllConstraint(TRUE); // Check PLL constraint
```

## sysSetCPUClock

### Synopsis

```
UINT32 sysSetCPUClock(UINT32 u32CPUClockKHz);
```

### Description

This function is used to set CPU clock.

### Parameter

u32CPUClockKHz Specified CPU clock

### Return Value

The CPU clock after setting. Due to some constraints of clock tree, the finally CPU clock may not equal to the specified CPU clock.

### Example

```
UINT32 u32CPUClock;
u32CPUClock = sysSetCPUClock (96000);
```

## sysGetCPUClock

### Synopsis

```
UINT32 sysGetCPUClock(VOID);
```

### Description

This function is used to get CPU clock.

### Parameter

None

### Return Value

The CPU clock

### Example

```
UINT32 u32CPUClock;
u32CPUClock = sysGetCPUClock ();
```



**sysSetAPBClock****Synopsis**

```
UINT32 sysSetAPBClock(UINT32 u32APBclockKHz);
```

**Description**

This function is used to set APB clock.

**Parameter**

u32APBclockKHz Specified APB clock

**Return Value**

The APB clock after setting. Due to some constraints of clock tree, the finally APB clock may not equal to the specified APB clock.

**Example**

```
UINT32 u32APBclock;  
u32APBclock = sysSetAPBClock (48000);
```

**sysGetAPBClock****Synopsis**

```
UINT32 sysGetAPBClock(VOID);
```

**Description**

This function is used to get APB clock.

**Parameter**

None

**Return Value**

The APB clock

**Example**

```
UINT32 u32APBclock;  
u32APBclock = sysGetAPBClock ();
```

## 17.8 Power management Function

**sysPowerDown****Synopsis**

```
INT32 sysPowerDown(WAKEUP_SOURCE_E eWakeUpSrc);;
```

**Description**

This function forces system to enter standby mode. All of IP clock were turned off except RTC and DDR enter self-refresh mode.

#### Parameter

eWakeUpSrc: Specified the wake up source.

eWakeUpSrc	Value	Meaning
WE_GPIO	0x1	Wake up from GPIO
WE_RTC	0x2	Wake up from RTC
WE_SDH	0x4	Wake up from SD Host
WE_UART	0x8	Wake up from UART (UART high speed CTS pin)
WE_UDC	0x10	Wake up from USB device (Host issue resume command)
WE_UHC	0x20	Wake up from USB host (USB device remote wake up)
WE_ADC	0x40	Wake up from ADC (Touch panel)
WE_KPI	0x80	Wake up from KPI.

#### Return Value

None

#### Example

```
/* Reset PM IRQ status*/
...
/* Configuration GPIO for wake up */
sysPowerDown(WE_GPIO);
/* Force system enter power down. Wake up source from GPIO */
```

### sysPowerDownPLLDuringSysPowerDown

#### Synopsis

```
void sysPowerDownPLLDuringSysPowerDown(BOOL blsPowerDownPLL)
```

#### Description

Calling this function will disable or enable PLL clock as system enter power down mode. It is used to save more power if disable PLL under power down mode. However, it will take more than 25ms for system wake up. If enable PLL as power down mode, the wake up time will be shrunk to 3ms

#### Parameter

blsPowerDownPLL      1: Power down PLL as power down mode  
0: Keep to enable PLL as power down mode

### Return Value

None

### Example

```
...
/* enable power down as enter power mode */
sysPowerDownPLLDuringSysPowerDown (1);
/* Configuration GPIO for wake up. PLLs will be disable as power down
mode */
sysPowerDown(WE_GPIO);
```

## 17.9 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

## 18 UDC Library

### 18.1 overview

This library is designed to make user application to use N9H20 UDC more easily.

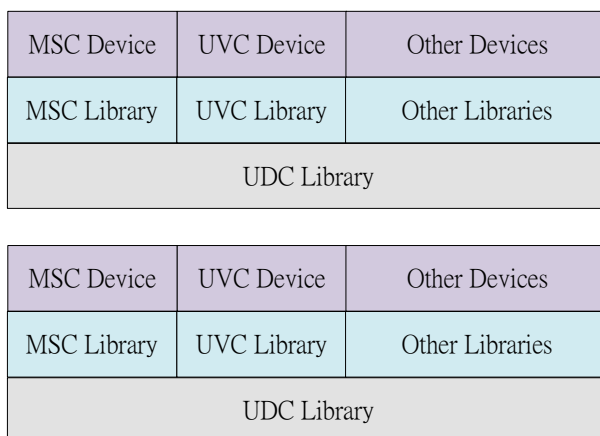
The UDC library has the following features:

- Support all basic USB operations.
- Pass USB-IF Chapter 9 test.

BSP 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.
- 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.



## 18.2 Programming Guide

### 18.2.1 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.

### 18.2.2 Features

- USB Specification version 2.0 compliant.
- Interfaces between USB 2.0 bus and the AHB bus.

- Support 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.
- Support 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 endpoint buffer.
- DMA operation is carried out by AHB master
- Supports Endpoint Maximum Packet Size up to 1024 bytes.

### 18.2.3 UDC Library Property Definition

The UDC library provides property structure to set UDC property more easily.

USBD\_INFO\_T (The fields for internal usage are not in the table)

Name	Description
Descriptor pointer	
pu32DevDescriptor	Device Descriptor pointer
pu32QulDescriptor	Device Qualifier Descriptor pointer
pu32HSConfDescriptor	Standard Configuration Descriptor pointer for High speed
pu32FSConfDescriptor	Standard Configuration Descriptor pointer for Full speed
pu32HOSConfDescriptor	Other Speed Configuration Descriptor pointer for High speed
pu32FOSConfDescriptor	Other Speed Configuration Descriptor pointer for Full speed
pu32HIDDescriptor	HID Device Descriptor pointer
pu32HIDRPTDescriptor	HID Device Report Descriptor pointer
pu32StringDescriptor[5]	String Descriptor pointer
Descriptor length	
u32DevDescriptorLen	Device Descriptor Length
u32QulDescriptorLen	Device Qualifier Descriptor pointer Length
u32HSConfDescriptorLen	Standard Configuration Descriptor Length for High speed
u32FSConfDescriptorLen	Standard Configuration Descriptor Length for Full speed
u32HOSConfDescriptorLen	Other Speed Configuration Descriptor Length for High speed
u32FOSConfDescriptorLen	Other Speed Configuration Descriptor Length for Full speed
u32HIDDescriptorLen	HID Device Descriptor Length
u32HIDRPTDescriptorLen	HID Device Report Descriptor Length
u32StringDescriptorLen[5]	String Descriptor Length
USBD Init	

pfnHighSpeedInit	High speed USB Device Initialization function
pfnFullSpeedInit	Full speed USB Device Initialization function
Endpoint Number	
i32EPA_Num	Endpoint Number for EPA (-1 : Not used)
i32EPB_Num	Endpoint Number for EPB (-1 : Not used)
i32EPC_Num	Endpoint Number for EPC (-1 : Not used)
i32EPD_Num	Endpoint Number for EPD (-1 : Not used)
Endpoint Call Back	
pfnEPACallBack	Callback function pointer for Endpoint A Interrupt
pfnEPBCallBack	Callback function pointer for Endpoint B Interrupt
pfnEPCCallBack	Callback function pointer for Endpoint C Interrupt
pfnEPDCallBack	Callback function pointer for Endpoint D Interrupt
Class Call Back	
pfnClassDataINCallBack	Callback function pointer for Class Data IN
pfnClassDataOUTCallBack	Callback function pointer for Class Data OUT
pfnDMACompletion	Callback function pointer for DMA Complete
pfnReset	Callback function pointer for USB Reset Interrupt
pfnSOF	Callback function pointer for USB SOF Interrupt
pfnPlug	Callback function pointer for USB Plug Interrupt
pfnUnplug	Callback function pointer for USB Un-Plug Interrupt
VBus status	
u32VbusStatus	VBus Status

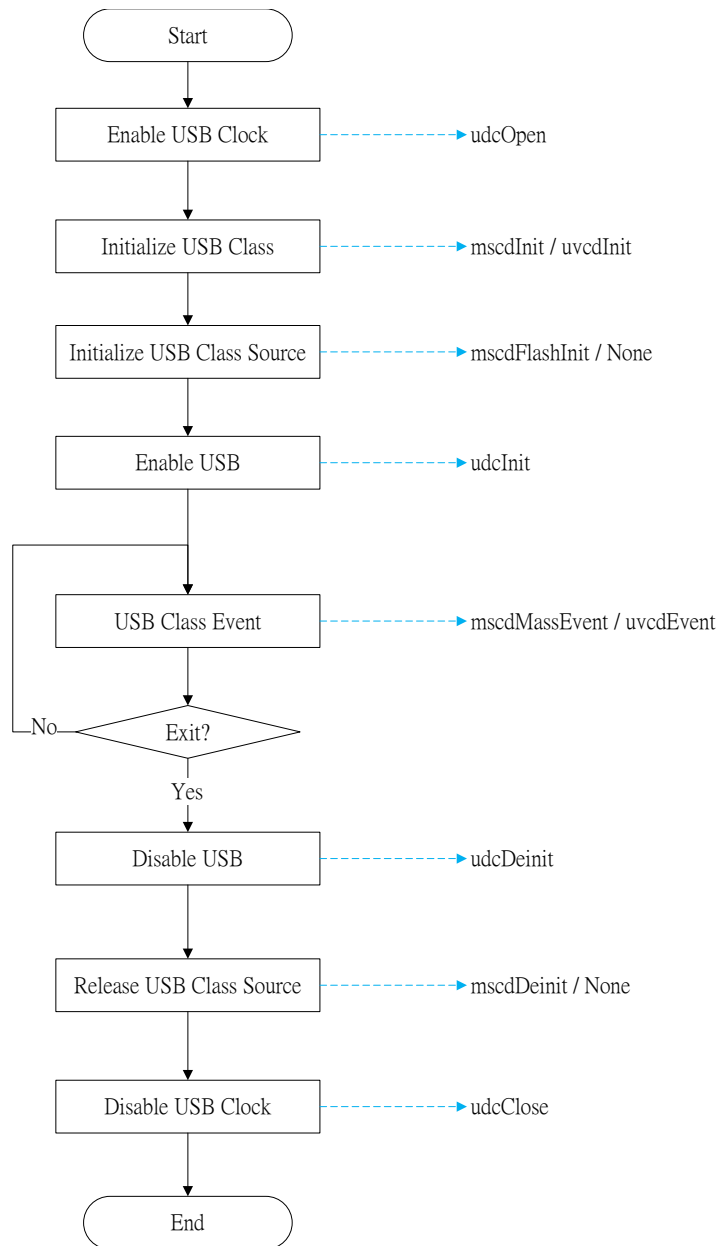
The USB Device initialization function initializes the basic setting of USB device controller including endpoints buffer allocation, endpoint number, endpoint type, speed mode, and interrupt, etc. User can modify the function to change USB speed and endpoint properties.

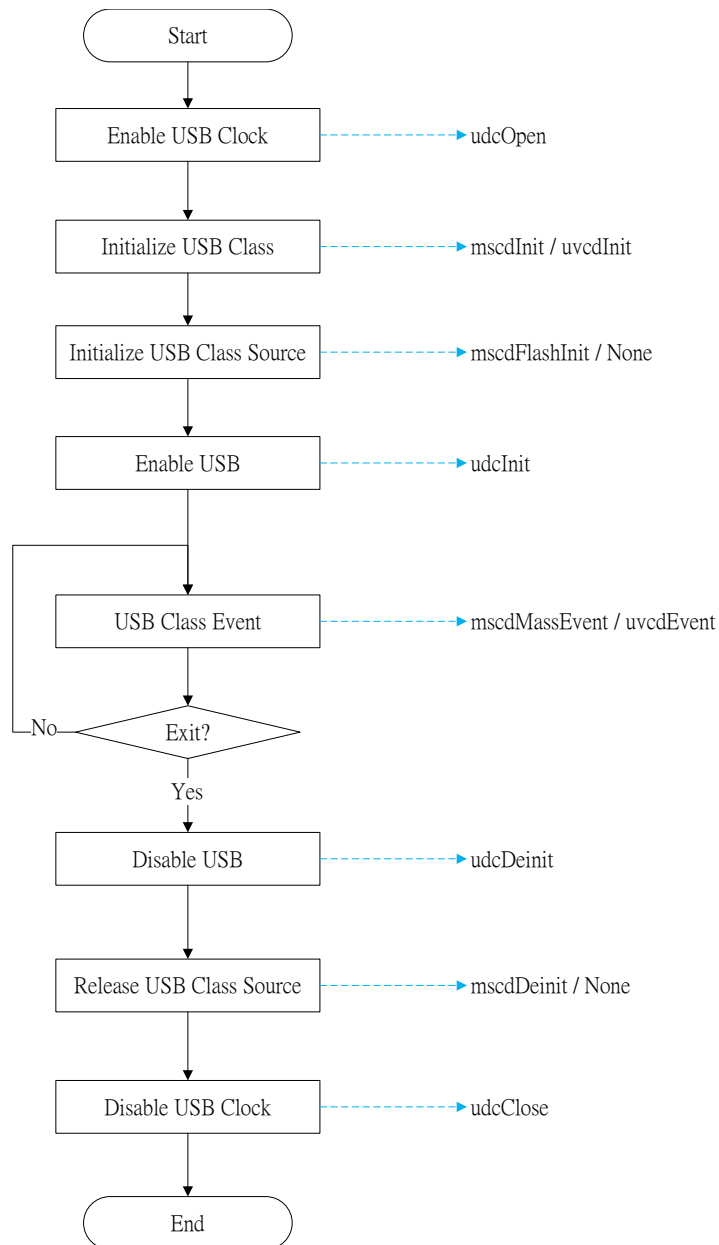
- pfnHighSpeedInit
- mscdHighSpeedInit
- pfnFullSpeedInit
- mscdFullSpeedInit

PC classifies USB derive according to the descriptors. With Non-OS BSP structure, the descriptors are initialized in the class initialize functions. The functions set proper descriptors and the callback functions.

- mscdInit

## 18.2.4 Programming Flow





## 18.3 USB Device (UDC) APIs Specification

### 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 ();
```

## udclsAttached

### Synopsis

BOOL udclsAttached(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");
```

## udclsAttachedToHost

### Synopsis

BOOL udclsAttachedToHost (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 send 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.

### udcSetSuspendCallback

#### Synopsis

VOID udcSetSuspendCallBack (PFN\_USBD\_CALLBACK pfun)

#### Description

This function is to install the Suspend call back function

#### Parameter

pfun                      The Suspend Call back function pointer

#### Return Value

None

#### Example

```
udcSetSuspendCallBack(Demo_PowerDownWakeUp);
```

## 18.4 Mass Storage Class (MSCD) APIs Specification

### 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 releases software source (allocated by mscdInit)

#### Parameter

None

#### Return Value

None

#### Example

```
mscdDeinit ();
```

### mscdFlashInit

#### Synopsis

UINT8 mscdFlashInit (char \*pDisk, INT SDsector)

#### Description

Initialize the Flash capacity for usb device controller use.

#### Parameter

pDisk                      The internal data for NAND disk information  
SDsector    The total sector number of SD card

#### Return Value

0                            - Fail  
1                            - Success

#### Example

```
NDISK_T MassNDisk;  
INT32 status;  
status = sicSdOpen0();  
if(status < 0)  
sicSdClose0();
```

```
mscdFlashInit(&MassNDisk, status);
```

#### Note

1. User can modify the mscd.c and rebuild library to select the flash types you want
  - TEST\_RAM: Ram Disk
    - Change RAM\_DISK\_SIZE to change RAM Disk size
      - ◆ RAMDISK\_1M / RAMDISK\_2M / RAMDISK\_4M / RAMDISK\_8M/ RAMDISK\_16M / RAMDISK\_32M
  - TEST\_SM: NAND Disk
  - TEST\_SD: SD Card Reader
2. The pDisk is used only when TEST\_SM is defined.

### mscdFlashInitCDROM

#### Synopsis

```
UINT8 mscdFlashInitCDROM (
    NDISK_T *pDisk,
    INT SDsector,
    PFN_MSCD_CDROM_CALLBACK pfnCallBack,
    INT CdromSizeInByte
)
```

#### Description

Initialize the Flash capacity for usb device controller use.

#### Parameter

pDisk	The internal data for NAND disk information.
SDsector	The total sector number of SD card
pfnCallBack	The callback function for CDROM read function
CdromSizeInByte	The size of CDROM size

#### Return Value

0	- Fail
1	- Success

#### Example

```
NDISK_T MassNDisk;
mscdFlashInitCDROM(&MassNDisk, NULL, CDROM_Read, u32CdromSize);
```

#### Note

3. User can modify the mscd.c and rebuild library to select the flash types you want
  - TEST\_RAM: Ram Disk
    - Change RAM\_DISK\_SIZE to change RAM Disk size
      - ◆ RAMDISK\_1M / RAMDISK\_2M / RAMDISK\_4M / RAMDISK\_8M/ RAMDISK\_16M / RAMDISK\_32M
  - TEST\_SM: NAND Disk
  - TEST\_SD: SD Card Reader
4. The pDisk is used only when TEST\_SM is defined.

### mscdSdPortSelect

#### Synopsis

VOID mscdSdPortSelect (UINT32 u32Port)

#### Description

This function can change the SD port used by MSC library before mscdFlashInit

#### Parameter

u32Port                      SD port index (0/1/2)

#### Return Value

None

#### Example

```
mscdSdPortSelect (0);
```

### mscdBlcokModeEnable

#### Synopsis

VOID mscdBlcokModeEnable (BOOL bEnable)

#### Description

This function can set MSC to Block mode or Non-Block mode.

#### Parameter

bEnable                      TRUE / FALSE

#### Return Value

None

#### Example

```
#ifdef NON_BLOCK_MODE
```

```
mscdBlcokModeEnable(FALSE);    /* Non-Block mode */
while(1)
{
    if(!PlugDetection())
        break;
    mscdMassEvent(NULL);
}
#else
    mscdMassEvent(PlugDetection); /* Default : Block mode */
#endif
```

## 19 USB Core Library

### 19.1 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.

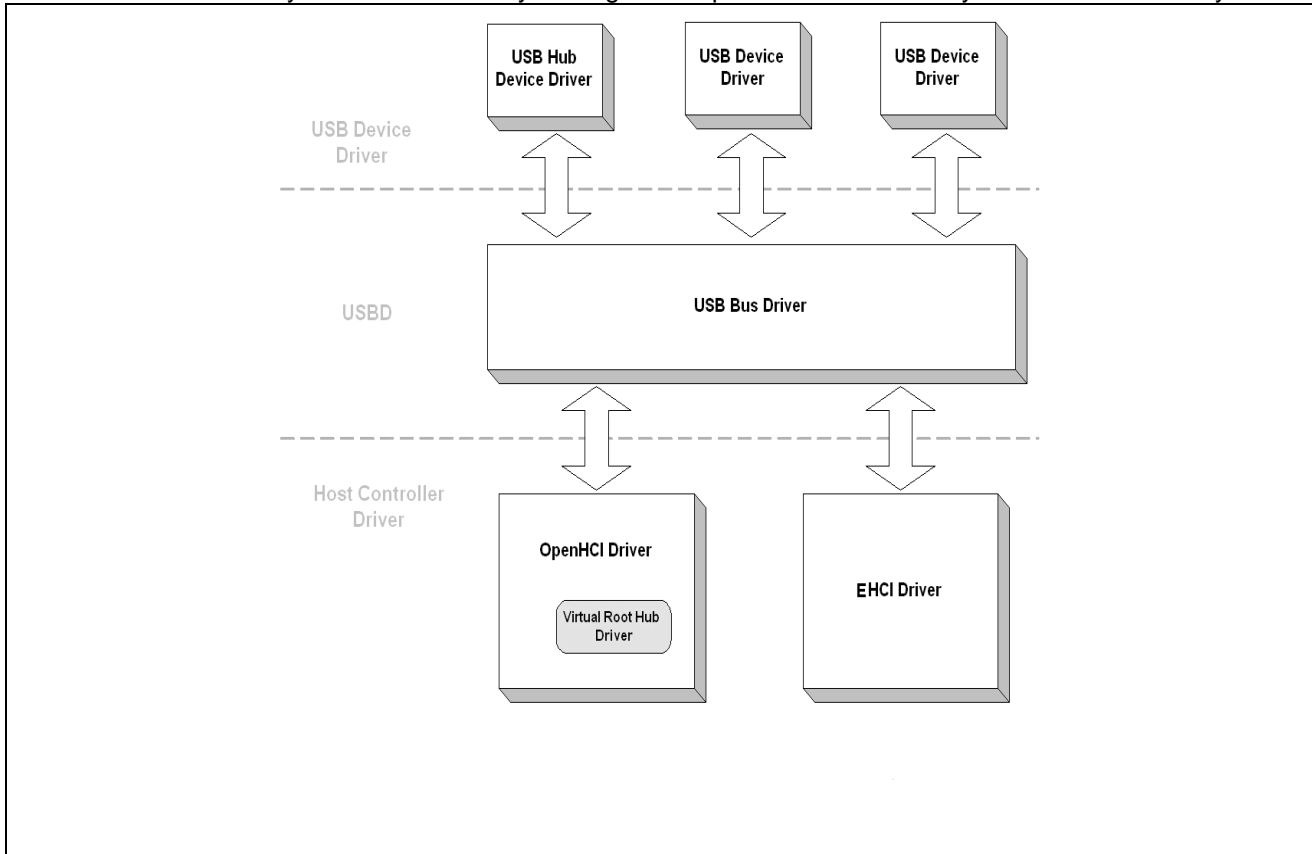


Figure 19-1 USB driver layer of USB library

### 19.2 Data Structures

The USB Core library includes 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 which you may need. These data structures are all defined in header file <usb.h>.

#### 19.2.1 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 19-1: 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;
```

Member	Description
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)
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

Table 19-1: Members of USB\_DEV\_T

## 19.3 Descriptor Structures

In the USB\_DEV\_T structure, device descriptor, configuration descriptor 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 the original descriptors list returned from the device. Table 2-2, Table 2-3, Table 2-4, and Table 2-5 describe the structures defined for device descriptor, 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.

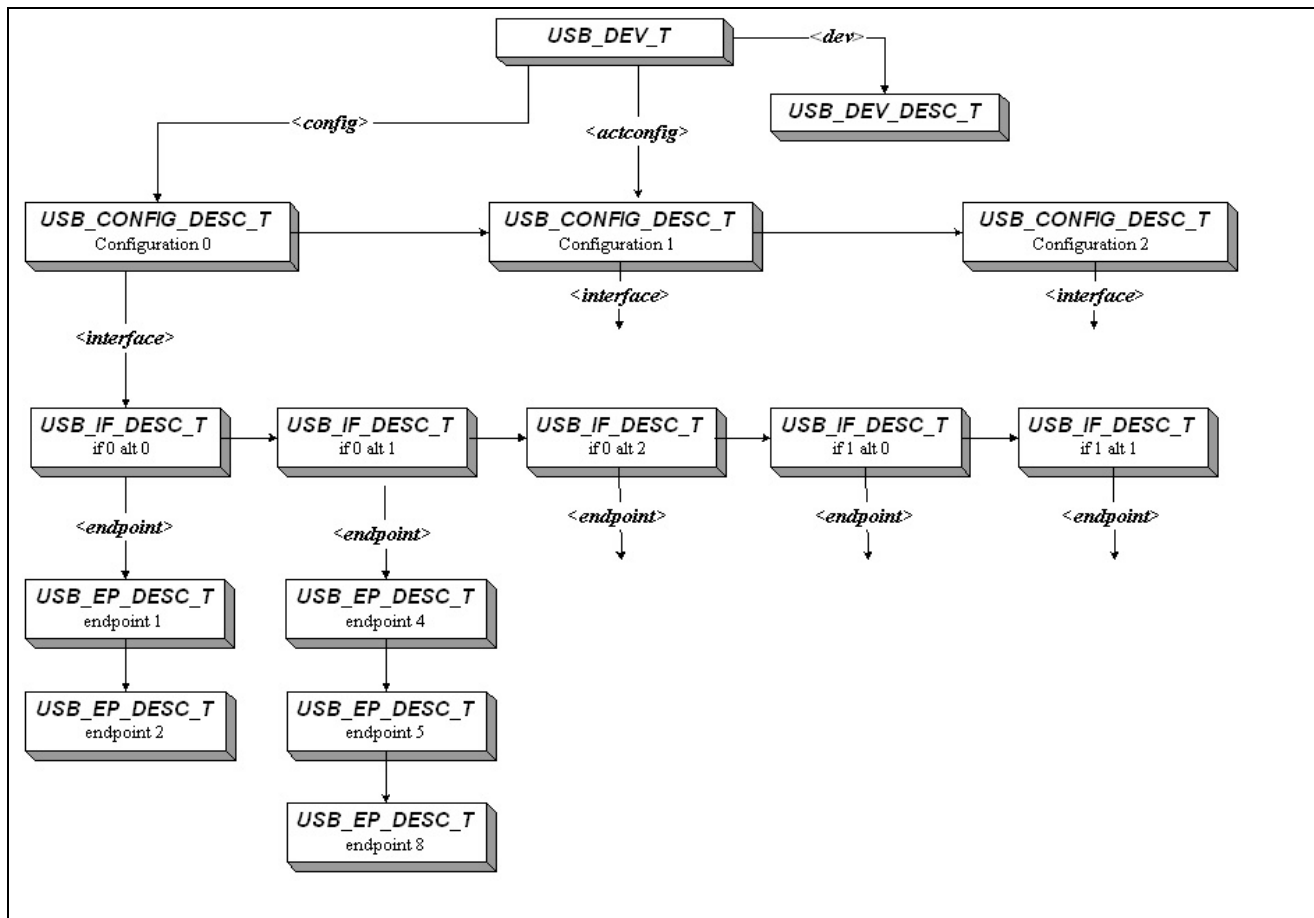


Figure 19-2:Descriptors relationship

```

/* 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 19-2: 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 19-3: 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 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 to 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 is 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 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 19-4: Members of USB\_IF\_DESC\_T

Member	Description
bLength	Size of the descriptor in bytes
bDescriptorType	INTERFACE descriptor type (0x04)
bInterfaceNumber	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)
bInterfaceClass	Class code
bInterfaceSubClass	Subclass code
bInterfaceProtocol	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 19-5: 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-byte 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 19-6: 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 employs 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 19-7: 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, 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 19-8: 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

## URB\_T

USB specification defines 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.

## 19.4 Data Transfer

USB specification defines four transfer types, control, bulk, interrupt, and isochronous. The USB device driver performs data transfer by preparing an URB and transfers 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 passes 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 designs 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 provides 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.

## Pipe Control

Before delivering a URB, the USB device driver must determine which device and 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 defines pipe structure with a 32-bits unsigned integer. The USB library defines several useful macros for pipe control. The pipe is defined as follows:

31	30	29	28	27	26	25	24
Pipe Type		Reserved			Speed	Reserved	
23	22	21	20	19	18	17	16
Reserved				Data0/1	Endpoint		
15	14	13	12	11	10	9	8
Device							
7	6	5	4	3	2	1	0
Direction	Reserved					Max Size	

Member	Description
Max Size [1 .. 0]	The maximum packet size. This field has been obsoleted. Now the maximum packet size is recorded in <code>&lt;epmaxpacketin&gt;</code> and <code>&lt;epmaxpacketout&gt;</code> fields of <code>USB_DEV_T</code> .
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 <code>SET_ADDRESS</code> 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.

Table 19-9: Members of Pipe Control

The USB library provides 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)
```

## Control Transfer

In this section, we will introduce how to make control transfer 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 provides 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 contains 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 uses <**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 delivers the URB. The callback function may also check the status of the URB to determine the transfer to be 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;
#ifdef LITTLE_ENDIAN
    pegasus->dr->value = 0;
    pegasus->dr->index = indx;
    pegasus->dr->length = size;
#else
    pegasus->dr->value = USB_SWAP16(0);
    pegasus->dr->index = USB_SWAP16(indx);
    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 prepares the device request command in `<pegasus->dr>`, which was later referred to by `<urb->setup_packet>`. It requests a buffer for DMA transfer by **USB\_malloc()**. Note that **USB\_malloc()** will allocate a non-cacheable memory buffer. It then creates a Control-In pipe by using **usb\_rcvctrlpipe** macro, and the endpoint number is 0. The device driver uses 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 suspends 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.

## Bulk Transfer

In this section, we will introduce how to make bulk transfers by URBs. The URB provides `<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 delivers the URB. The callback function may also check the status of the URB to determine the transfer to be 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;
}

```

## Interrupt Transfer

In this section, we will introduce how to make interrupt transfer by URBs. The URB provides **<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 the endpoint of target interrupt.

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 the received interrupt data. The USB device driver has 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);

```

## 19.5 USB Core Library APIs Specification

### USB\_PortInit

#### Synopsis

```
INT USB_PortInit (UINT32 u32PortType);
```

#### Description

The function is used to initialize USB host port type.

#### Parameter

u32PortType:

Table 19-10: 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 blsDisPort0, BOOL blsDisPort1);
```

### Description

The function is used to disable USB host ports if the port is useless.

### Parameter

blsDisPort0	TRUE to disable port 0. FALSE to enable port 0
blsDisPort1	TRUE to disable port 1. FALSE to enable port 1

### Return Value

None

### Example

```
/* In/out through host like port 0 and disable 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 function execute. 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 NVT FAT file system, USB core system, and USB mass
storage driver
*/
fsInitFileSystem();
USB_PortInit(HOST_LIKE_PORT0);
USB_PortDisable(FALSE, TRUE);
InitUsbSystem();
UMAS_InitUmasDriver();
```

**DelInitUsbSystem**

**Synopsis**

INT DelInitUsbSystem(VOID)

**Description**

De-Initialize the USB hardware and USB core library.

**Parameter**

None

**Return Value**

0 – Success

**Example**

```
/*
Initialize NVT 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();
```

## umass\_register\_connect

### Synopsis

```
VOID umass_register_connect(PFN_PORT_MS_CALLBACK pfnCallback);
```

### Description

Register connection call back function for mass storage device plug in.

### Parameter

pfnCallback      Call back function as mass storage device plug in.

### Return Value

None

## umass\_register\_disconnect

### Synopsis

```
VOID umass_register_disconnect(PFN_PORT_MS_CALLBACK pfnCallback);
```

### Description

Register disconnection call back function for mass storage device plug out.

### Parameter

pfnCallback      Call back function as mass storage device plug out.

### Return Value

None

### Example

```
VOID MassStotrageConnection(void* umas)
{
    sysprintf("Umas driver connect 0x%x\n", (UINT32)umas);
    ...
}
VOID MassStotrageDisconnection(void* umas)
{
    sysprintf("Umas driver disconnect 0x%x\n", (UINT32)umas);
    ...
}
void PenDriverConnectTest(void)
{
    umass_register_connect(MassStotrageConnection);
}
```

```
umass_register_disconnect(MassStorageDisconnection);
InitUsbSystem();
UMAS_InitUmasDriver();
...
while(1);
}
```

## 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 NVT FAT file system automatically.

### Parameter

None

### Return Value

0 – Success  
Otherwise – Failure

### Example

```
/*
Initialize NVT FAT file system, USB core system, and USB mass
storage driver
*/
fsInitFileSystem();
USB_PortInit(HOST_LIKE_PORT0);
USB_PortDisable(FALSE, TRUE);
InitUsbSystem();
UMAS_InitUmasDriver();
```

## UMAS\_RemoveUmasDriver

### Synopsis

INT UMAS\_RemoveUmasDriver (VOID)

### Description

Deinitialize the USB mass storage driver.

### Parameter

None

### Return Value

0 – Success  
Otherwise – Failure

### Example

```
/*
Initialize NVT FAT file system, USB core system, and USB mass
storage driver
*/
fsInitFileSystem();
USB_PortInit(HOST_LIKE_PORT0);
USB_PortDisable(FALSE, TRUE);
InitUsbSystem();
UMAS_InitUmasDriver();
.....
.....
UMAS_RemoveUmassDriver();
```

## 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 is registered with USB core library

### Return Value

0 – Success  
Otherwise – Failure

### Example

```
static USB_DRIVER_T  usb1p_driver =
{
  "usb1p",
  usb1p_probe,
```



```

usb1p_disconnect,
{NULL,NULL},
{0},
NULL,
usb1p_ids,
NULL,
NULL
};

INT  UsbPrinter_Init() {
    if (USB_RegisterDriver(&usb1p_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 is deregistered

### Return Value

0 – Success  
Otherwise – Failure

### Example

```

VOID  UsbPrinter_Exit()
{
    USB_DeregisterDriver(&usb1p_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 within 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

Free 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

Build a bulk URB, send it off and wait 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 manages 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("usb1p_ctrl_msg - Memory not enough!\n");
        return -1;
    }
    retval = USB_SendControlMessage(usb1p->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 free.

### Return Value

None

### Example

Same as USB\_malloc()

## 20 VPOST Library

### 20.1 Overview

The VPOST consists of LCD and TV encoder controller and 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 synchronized with TV (NTSC/PAL non-interlace timing) or set by the LCD timing control register. The video/image data source may be come from the frame buffer, color bar or register settings. In general, the frame buffer which is stored in system memory is used as the image source.

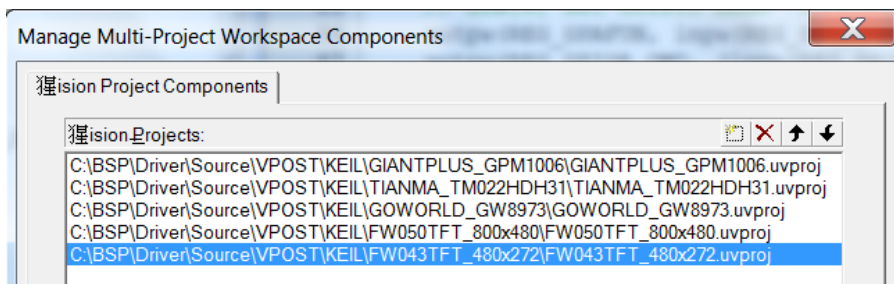
#### 20.1.1 How to build VPOST library

There are several panels can be supported by the VPOST library. If the current driver cannot support the selected panel, the user can add the desired one in the following procedure.

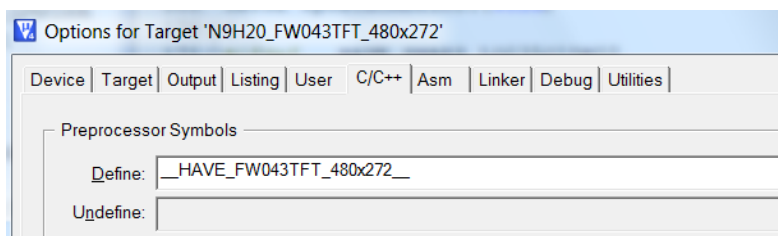
Prepare two functions, i.e., “vpostLCMInit\_PanelName()” and “vpostLCMDeInit\_PanelName()” in the desired panel driver.

In N9H20\_VPOST.c, please add the statement “vpostLCMInit\_PanelName();” and “vpostLCMDeInit\_PanelName();” to functions of vpostLCMInit () and vpostLCMDeInit(), respectively.

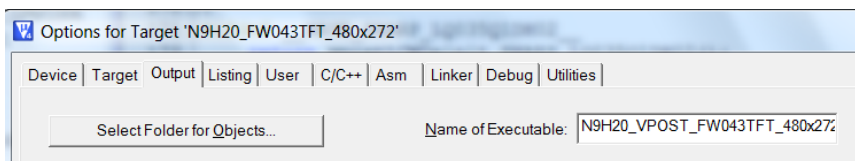
Add the panel project to the VPOST multi-project library.



Add the preprocessor symbol of “\_\_HAVE\_PanelName\_\_” in the panel project.



Give the output name for the panel.



According to above procedure, the user can create the desired driver for selected panel.

### 20.2 VPOST APIs Specification

## 20.3 Enumeration

Name	Value	Description
E_DRVPOST_TIMING_TYPE		
eDRVPOST_SYNC_TV	0x0	LCD timing sync with TV
eDRVPOST_ASYNC_TV	0x1	LCD timing not sync with TV
E_DRVPOST_IMAGE_SOURCE		
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
E_DRVPOST_IMAGE_SCALING		
eDRVPOST_DUPLICATED	0x0	Duplicate for TV Line buffer scaling
eDRVPOST_INTERPOLATION	0x1	Interpolation for TV line buffer scaling
E_DRVPOST_LCM_TYPE		
eDRVPOST_HIGH_RESOLUTION_SYNC	0x0	High resolution LCD device type
eDRVPOST_SYNC	0x1	Sync-type TFT LCD
eDRVPOST_MPU	0x3	MPU-type LCD
E_DRVPOST_MPU_TYPE		
eDRVPOST_I80	0x0	80-series MPU interface
eDRVPOST_M68	0x1	68-series MPU interface
E_DRVPOST_8BIT_SYNCLCM_INTERFACE		
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
E_DRVPOST_CCIR656_MODE		
eDRVPOST_CCIR656_360	0x0	720Y 360CbCr mode for CCIR656 horizontal active width
eDRVPOST_CCIR656_320	0x1	640Y 320CbCr mode for CCIR656 horizontal active width
E_DRVPOST_ENDIAN		
eDRVPOST_YUV_BIG_ENDIAN	0x0	Big Endian for YCbCr
eDRVPOST_YUV_LITTLE_ENDIAN	0x1	Little Endian for YCbCr
E_DRVPOST_SERAIL_SYNCLCM_COLOR_ORDER		
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_RBG	0x3	Data in RBG order



E_DRVVPPOST_PARALLEL_SYNCLCM_INTERFACE		
eDRVVPPOST_PRGB_16BITS	0x0	16 pin parallel RGB data bus
eDRVVPPOST_PRGB_18BITS	0x1	18 pin parallel RGB data bus
eDRVVPPOST_PRGB_24BITS	0x2	24 pin parallel RGB data bus
E_DRVVPPOST_SYNCLCM_DATABUS		
eDRVVPPOST_SYNC_8BITS	0x0	8 bit sync-type LCD
eDRVVPPOST_SYNC_9BITS	0x1	9 bit sync-type LCD
eDRVVPPOST_SYNC_16BITS	0x2	16 bit sync-type LCD
eDRVVPPOST_SYNC_18BITS	0x3	18 bit sync-type LCD
eDRVVPPOST_SYNC_24BITS	0x4	24 bit sync-type LCD
E_DRVVPPOST_MPULCM_DATABUS		
eDRVVPPOST_MPU_8_8	0x0	Transfer in 8-8 format for 16 bit color in 8 bit bus width
eDRVVPPOST_MPU_2_8_8	0x1	Transfer in 2-8-8 format for 18 bit color in 8 bit bus width
eDRVVPPOST_MPU_6_6_6	0x2	Transfer in 6-6-6 format for 18 bit color in 8 bit bus width
eDRVVPPOST_MPU_8_8_8	0x3	Transfer in 8-8-8 format for 24 bit color in 8 bit bus width
eDRVVPPOST_MPU_9_9	0x4	Transfer in 9-9 format for 18 bit color in 9 bit bus width
eDRVVPPOST_MPU_16	0x5	Transfer in 16 format for 16 bit color in 16 bit bus width
eDRVVPPOST_MPU_16_2	0x6	Transfer in 16-2 format for 18 bit color in 16 bit bus width
eDRVVPPOST_MPU_2_16	0x7	Transfer in 2-16 format for 18 bit color in 16 bit bus width
eDRVVPPOST_MPU_16_8	0x8	Transfer in 16-8 format for 24 bit color in 16 bit bus width
eDRVVPPOST_MPU_18	0x9	Transfer in 18 format for 18 bit color in 18 bit bus width
eDRVVPPOST_MPU_18_6	0xA	Transfer in 18-6 format for 124 bit color in 18 bit bus width
eDRVVPPOST_MPU_24	0xB	Transfer in 24 format for 24 bit color in 24 bit bus width
E_DRVVPPOST_FRAME_DATA_TYPE		
eDRVVPPOST_FRAME_RGB555	0x0	RGB555 Frame buffer data format
eDRVVPPOST_FRAME_RGB565	0x1	RGB565 Frame buffer data format
eDRVVPPOST_FRAME_RGBX888	0x2	RGB_Dummy888 Frame buffer data format
eDRVVPPOST_FRAME_RGB888X	0x3	RGB888_Dummy Frame buffer data format
eDRVVPPOST_FRAME_CBYCRY	0x4	Cb0Y0Cr0Y1 Frame buffer data format
eDRVVPPOST_FRAME_YCBYCR	0x5	Y0Cb0Y1Cr0 Frame buffer data format
eDRVVPPOST_FRAME_CRYCBY	0x6	Cr0Y0Cb0Y1 Frame buffer data format
eDRVVPPOST_FRAME_YCRYCB	0x7	Y0Cr0Y1Cb0 Frame buffer data format

E_DRVVPPOST_DATABUS		
eDRVVPPOST_DATA_8BITS	0x0	8 bits data bus
eDRVVPPOST_DATA_9BITS	0x1	9 bits data bus
eDRVVPPOST_DATA_16BITS	0x2	16 bits data bus
eDRVVPPOST_DATA_18BITS	0x3	18 bits data bus
eDRVVPPOST_DATA_24BITS	0x4	24 bits data bus
E_DRVVPPOST_OSD_DATA_TYPE		
eDRVVPPOST_OSD_RGB555	0x8	RGB555 OSD data format
eDRVVPPOST_OSD_RGB565	0x9	RGB565 OSD data format
eDRVVPPOST_OSD_RGBx888	0xA	RGB_Dummy888 OSD data format
eDRVVPPOST_OSD_RGB888x	0xB	RGB_888Dummy OSD data format
eDRVVPPOST_OSD_ARGB888	0xC	RGB_Alfa888 OSD data format
eDRVVPPOST_OSD_CB0Y0CR0Y1	0x0	CB0Y0CR0Y1 OSD data format
eDRVVPPOST_OSD_Y0CB0Y1CR0	0x1	Y0CB0Y1CR0 OSD data format
eDRVVPPOST_OSD_CR0Y0CB0Y1	0x2	CR0Y0CB0Y1 OSD data format
eDRVVPPOST_OSD_Y0CR0Y1CB0	0x3	Y0CR0Y1CB0 OSD data format
eDRVVPPOST_OSD_Y1CR0Y0CB0	0x4	Y1CR0Y0CB0 OSD data format
eDRVVPPOST_OSD_CR0Y1CB0Y0	0x5	CR0Y1CB0Y0 OSD data format
eDRVVPPOST_OSD_Y1CB0Y0CR0	0x6	Y1CB0Y0CR0 OSD data format
eDRVVPPOST_OSD_CB0Y1CR0Y0	0x7	CB0Y1CR0Y0 OSD data format
E_DRVVPPOST_OSD_TRANSPARENT_DATA_TYPE		
eDRVVPPOST_OSD_TRANSPARENT_RGB565	0x0	RGB565 transparent color format
eDRVVPPOST_OSD_TRANSPARENT_YUV	0x1	YUV transparent color format
eDRVVPPOST_OSD_TRANSPARENT_RGB888	0x2	RGB888 transparent color format

## 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 20-1: LCDFORMAT\_EX structure

Field	Type	Description
u8PulseWidth	UINT8	Horizontal sync pulse width
u8BackPorch	UINT8	Horizontal back porch
u8FrontPorch	UINT8	Horizontal front porch

Table 20-2: S\_DRVPOST\_SYNCLCM\_HTIMING structure

Field	Type	Description
u8PulseWidth	UINT8	Vertical sync pulse width
u8BackPorch	UINT8	Vertical back porch
u8FrontPorch	UINT8	Vertical front porch

Table 20-3: S\_DRVPOST\_SYNCLCM\_VTIMING 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 20-4: S\_DRVPOST\_SYNCLCM\_WINDOW structure

Field	Type	Description
bIsVsyncActiveLow	BOOL	Vsync polarity
bIsHsyncActiveLow	BOOL	Hsync polarity
bIsVDenActiveLow	BOOL	VDEN polarity
bIsDClockRisingEdge	BOOL	Clock polarity

Table 20-5: S\_DRVPOST\_SYNCLCM\_POLARITY 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 20-6: 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 20-7: S\_DRVPOST\_MPULCM\_WINDOW structure

Field	Type	Description
blsSyncWithTV	BOOL	MPU timing sync with TV
blsVsyncSignalOut	BOOL	Specify MPU FrameMark pin as input or output pin
blsFrameMarkSignalIn	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_MPU_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

Table 20-8: S\_DRVPOST\_MPULCM\_TIMING structure

Field	Type	Description
u16VSize	UINT16	Specify OSD vertical size
u16HSize	UINT16	Specify OSD horizontal size

Table 20-9: S\_DRVPOST\_OSD\_SIZE structure

Field	Type	Description
u16VStart_1st	UINT16	Specify 1 <sup>st</sup> OSD bar vertical start position
u16VEnd_1st	UINT16	Specify 1 <sup>st</sup> OSD bar vertical end position
u16VOffset_2nd	UINT16	Specify 2 <sup>nd</sup> OSD bar vertical offset position (2nd_Start – 1st_End)
u16HStart_1st	UINT16	Specify 1 <sup>st</sup> OSD bar horizontal start position
u16HEnd_1st	UINT16	Specify 1 <sup>st</sup> OSD bar horizontal end position
u16HOffset_2nd	UINT16	Specify 2 <sup>nd</sup> OSD bar horizontal offset position (2nd_Start –

		1st_End)
--	--	----------

Table 20-10: S\_DRVVPPOST\_OSD\_POS structure

Field	Type	Description
blsOSDEnabled	BOOL	Specify OSD to be enabled or disabled
u32Address	UINT32	Specify the beginning address of OSD buffer
eType	E_DRVVPPOST_OSD_DATA_TYPE	Specify OSD data type
psSize	S_DRVVPPOST_OSD_SIZE*	Specify the data pointer of OSD size
psPos	S_DRVVPPOST_OSD_POS*	Specify the data pointer of OSD position

Table 20-11: S\_DRVVPPOST\_OSD\_CTRL structure

## 20.4 Functions

### vpostGetFrameBuffer

#### Synopsis

```
void *vpostGetFrameBuffer (void);
```

#### Description

Get the display frame buffer address

#### Parameter

None

#### Return Value

Display frame buffer address.

#### Example

### 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.

## vpostEnaBacklight

### Synopsis

```
void vpostEnaBacklight (void);
```

### Description

The function enables the backlight led.

### Parameter

None

### Return Value

None

### Example

None

## vpostSetOSD\_Enable

### Synopsis

```
void vpostSetOSD_Enable (void);
```

### Description

The function enables the OSD feature.

### Parameter

None

### Return Value

None

### Example

None

## **vpostSetOSD\_Disable**

### **Synopsis**

```
void vpostSetOSD_Disable (void);
```

### **Description**

The function disables the OSD feature.

### **Parameter**

None

### **Return Value**

None

### **Example**

None

## **vpostSetOSD\_Size**

### **Synopsis**

```
void vpostSetOSD_Size (
    S_DRVVPST_OSD_SIZE* psSize
);
```

### **Description**

The function sets the OSD size.

### **Parameter**

#### **psSize [in]**

Input the OSD size pointer.

### **Return Value**

None

### **Example**

None

## **vpostSetOSD\_Pos**

### **Synopsis**

```
void vpostSetOSD_Pos (
    S_DRVVPST_OSD_POS* psPos
);
```

### **Description**



The function sets the OSD position.

**Parameter**

**psPos [in]**

Input the OSD position pointer.

**Return Value**

None

**Example**

None

**vpostSetOSD\_DataType**

**Synopsis**

```
void vpostSetOSD_DataType (
    E_DRVVPOST_OSD_DATA_TYPE eType
);
```

**Description**

The function sets the OSD data type.

**Parameter**

**eType [in]**

Set the OSD data type.

**Return Value**

None

**Example**

None

**vpostSetOSD\_Transparent\_Enable**

**Synopsis**

```
void vpostSetOSD_Transparent _Enable (void);
```

**Description**

The function enables the OSD transparent feature.

**Parameter**

None

**Return Value**

None

**Example**

None

**vpostSetOSD\_Transparent\_Disable**

**Synopsis**

```
void vpostSetOSD_Transparent _Disable (void);
```

**Description**

The function disables the OSD transparent feature.

**Parameter**

None

**Return Value**

None

**Example**

None

**vpostSetOSD\_Transparent**

**Synopsis**

```
void vpostSetOSD_Transparent (
    E_DRVPOST_OSD_TRANSPARENT_DATA_TYPE eType
    UINT32 u32Pattern
);
```

**Description**

The function sets the OSD transparent data type and pattern.

**Parameter**

**eType [in]**

Set the OSD transparent data type.

**u32Pattern [in]**

Given the transparent pattern.

**Return Value**

None

**Example**

None

**vpostSetOSD\_BaseAddress**

### Synopsis

```
void vpostSetOSD_BaseAddress (
    UINT32 u32BaseAddress
);
```

### Description

The function sets the OSD buffer base address.

### Parameter

**u32BaseAddress [in]**

Given the OSD base address.

### Return Value

None

### Example

None

## 20.5 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

## 21 EDMA Library

### 21.1 Overview

This library is designed to make user application to set N9H20 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.

### 21.2 Programming Guide

#### 21.2.1 System Overview

The N9H20 contains an enhanced direct memory access (EDMA) controller that transfers data to and from memory or transfers data to and from APB. The EDMA controller has 5-channel DMA that include 1 channel VDMA (Video-DMA, Memory-to-Memory) and four channels PDMA (Peripheral-to-Memory or Memory-to-Peripheral). For channel 0 VDMA mode, it also supports color format transform and stride mode transfer. For PDMA channel (EDMA CH1~CH4), it can transfer data between the Peripherals APB IP (ex: UART, SPI, ADC....) and Memory. The N9H20 also supports hardware scatter-gather function, software can set CSRx [SG\_EN] to enable scatter-gather function.

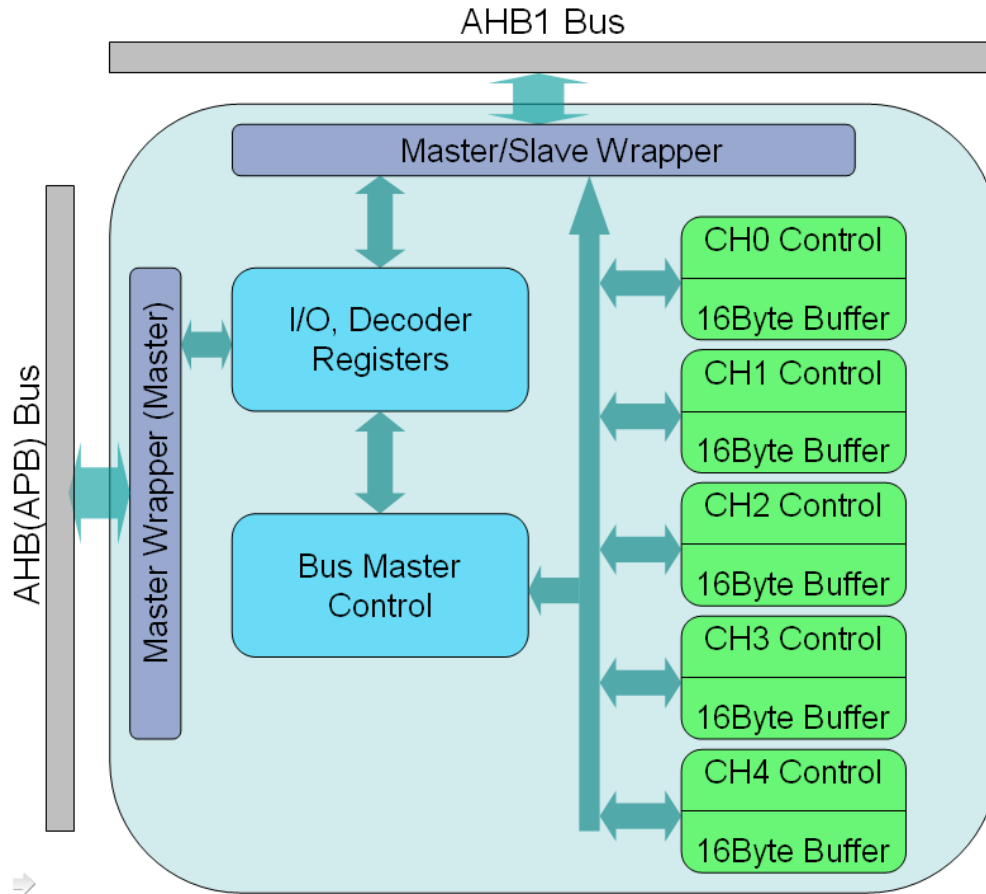
Software can stop the EDMA operation by disable DMA [DMACEN]. The CPU can recognize the completion of an EDMA operation by software polling or when it receives an internal EDMA interrupt. The N9H20 VDMA controller can increment source or destination address, decrement or fixed them as well, and the PDMA can increment source or destination, fixed or wrap around address.

#### 21.2.2 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.

#### 21.2.3 Block Diagram

The following figure describes the architecture of EDMA.



## 21.2.4 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\_CSRx [DMACEN] and then write a valid source address to the DMA\_SARx register, a destination address to the DMA\_DARx 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 [DMACEN] 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.

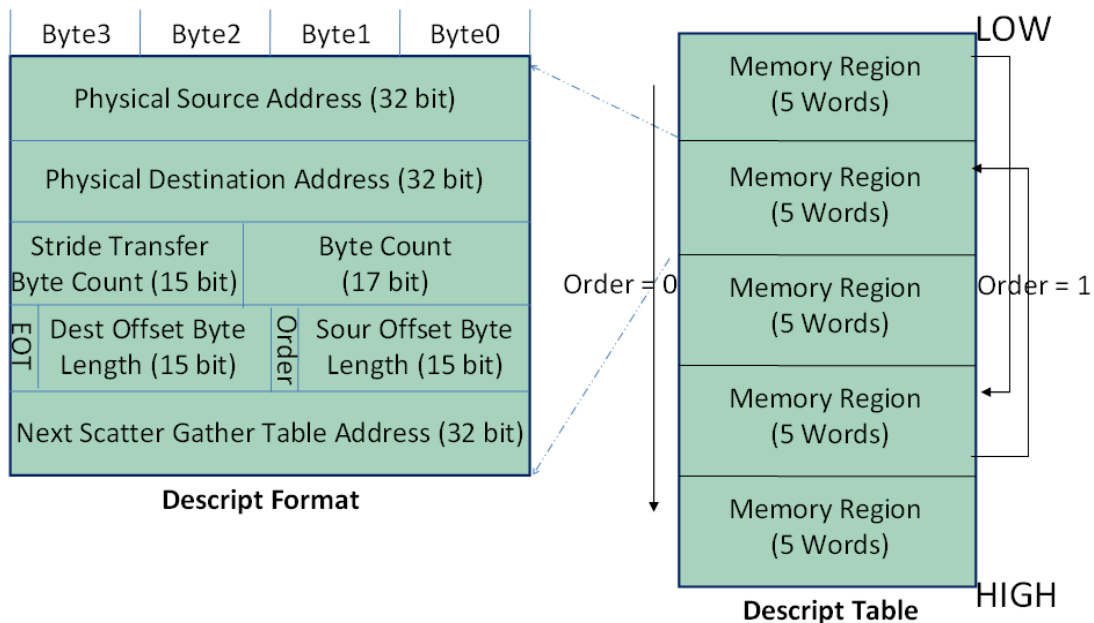
Below table lists the control register and control bit for it.

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 N9H20 also supports 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 be 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

## 21.3 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 ();
```

### EDMA\_Enable

#### Synopsis

```
void EDMA_Enable(int channel)
```

#### Description

This function is used to start EDMA channel operation.

**Parameter**

channel                      EDMA channel number

**Return Value**

None

**Example**

```
EDMA_Enable (0);
```

**EDMA\_Disable**

**Synopsis**

```
void EDMA_Disable(int channel)
```

**Description**

This function is used to stop EDMA channel operation.

**Parameter**

channel                      EDMA channel number

**Return Value**

None

**Example**

```
EDMA_Disable (0);
```

**EDMA\_Request**

**Synopsis**

```
int EDMA_Request(int channel)
```

**Description**

This function is used to allocate specified channel number.

**Parameter**

channel                      EDMA channel number

**Return Value**

SUCCESS                      Allocation channel is returned.

FAIL                              < 0 is returned.

EDMA\_ERR\_BUSY : specified channel is busy.

EDMA\_ERR\_INVALID : specified channel is greater than the maximum

number of channels.



### Example

```
EDMA_Request (0);
```

## 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 ();
```

## 21.3.1.1 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, eDRVEDMA_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_INVALID : 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\_INVALID : 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, eDRVEDMA_RGB565, eDRVEDMA_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.

**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\_TriggerDone****Synopsis**

void EDMA\_TriggerDone(int channel)

**Description**

This function is used to set the variable of EDMA channel transfer done.

**Parameter**

channel      EDMA channel number

**Return Value**

None

**Example**

```
EDMA_TriggerDone (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, eDRVEDMA_ADC, eDRVEDMA_READ_APB,
eDRVEDMA_WIDTH_32BITS);
```

### 21.3.1.2 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, eDRVEDMA_WRAPAROUND_EMPTY |
eDRVEDMA_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 , eDRVEDMA_DIRECTION_FIXED,
eDRVEDMA_DIRECTION_WRAPAROUND);
```

## 21.4 Error Code Table

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

## Revision History

Date	Revision	Description
2021.06.21	1.10	Revision
2018.05.02	1.00	Initially issued.



### Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

---

*Please note that all data and specifications are subject to change without notice.  
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.*