

N9H26 Non-OS Library Reference Guide

Document Information

Abstract	Introduce Non-OS Library for the N9H26 series microprocessor (MPU).
Apply to	N9H26 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

Table of Contents

1	INTRODUCTION	7
2	AAC LIBRARY	8
2.1	AAC Library Overview	8
2.2	AAC API	8
3	AES LIBRARY	11
3.1	AES Library Overview	11
3.2	Definition	11
3.3	API Function	11
3.4	AES Error Code Table	18
4	AUDIO ADC LIBRARY	19
4.1	Audio ADC Library Overview	19
4.2	Definition	19
4.3	API function	21
5	AVI LIBRARY	30
5.1	AVI Library Overview	30
5.2	Definition	31
5.3	API function	32
5.4	AVI Error Code Table	36
6	MP3 LIBRARY	38
6.1	MP3 Library Overview	38
6.2	Definition	38
6.3	API function	39
7	BLT LIBRARY	42
7.1	BLT Library Overview	42
7.2	Definition	50
7.3	API function	54
7.4	BLT Error Code Table	72
8	CRC LIBRARY	73

8.1	CRC Library Overview.....	73
8.2	Definition	73
8.3	API function	75
8.4	CRC Error Code Table.....	78
9	EDMA LIBRARY	79
9.1	EDMA Library Overview.....	79
9.2	API function	81
9.3	EDMA Error Code Table.....	90
10	FONT LIBRARY	91
10.1	Font Library Overview.....	91
10.2	Definition	91
10.3	API function	92
11	GNAND LIBRARY	99
11.1	GNAND Library Overview	99
11.2	Definition	101
11.3	API function	102
11.4	GNAND Error Code Table.....	109
12	GPIO LIBRARY	110
12.1	GPIO Library Overview	110
12.2	API function	110
13	H264 CODEC LIBRARY	122
13.1	H264 Codec Library Overview	122
13.2	Definition	123
13.3	API function	126
13.4	H264 Error Code Table.....	131
14	I²C LIBRARY	133
14.1	I ² C Library Overview.....	133
14.2	Definition	133
14.3	API function	133
14.4	I ² C Error Code Table.....	138

15	I²S LIBRARY	139
	15.1 API function	139
16	JPEG LIBRARY	144
	16.1 JPEG Library Overview	144
	16.2 Definition	150
	16.3 API function	156
17	NVTFAT LIBRARY	163
	17.1 NVTFAT Library Overview	163
	17.2 Definition	168
	17.3 API function	168
	17.4 NVTFAT Error Code Table	206
18	PWM LIBRARY	209
	18.1 PWM Library Overview	209
	18.2 Definition	209
	18.3 API function	211
19	RFC LIBRARY	225
	19.1 RFC Library Overview	225
	19.2 Definition	225
	19.3 API function	226
	19.4 RFC Error Code Table	229
20	ROTATION LIBRARY	230
	20.1 Rotation Library Overview	230
	20.2 Definition	230
	20.3 API function	231
	20.4 Rotation Error Code Table	234
21	RSC LIBRARY	235
	21.1 RSC Library Overview	235
	21.2 API function	235
	21.3 RSC Error Code Table	238

22	RTC LIBRARY	239
	22.1 EDMA Library Overview.....	239
	22.2 Definition	239
	22.3 API function	243
	22.4 RTC Error Code Table	249
23	SDIO LIBRARY	251
	23.1 SDIO Library Overview	251
	23.2 Definition	251
	23.3 API function	253
	23.4 SDIO Error Code Table	258
24	SIC LIBRARY	259
	24.1 SIC Library Overview	259
	24.2 Definition	260
	24.3 API function	261
	24.4 SIC Error Code Table	273
25	SPI LIBRARY	275
	25.1 SPI Library Overview.....	275
	25.2 API function	275
26	SPI SECUREIC LIBRARY	285
	26.1 SPI SecureIC Library Overview	285
	26.2 API function	285
27	SPI TO UART LIBRARY	292
	27.1 SPI to UART Library Overview	292
	27.2 API function	292
28	SPU LIBRARY	298
	28.1 SPU Library Overview	298
	28.2 Definition	298
	28.3 API function	298
29	SYSTEM LIBRARY	303

29.1 System Library Overview	303
29.2 Definition	303
29.3 API function	309
29.4 System Error Code Table	347
30 TOUCH ADC LIBRARY	349
30.1 System Library Overview	349
30.2 Definition	349
30.3 API function	349
30.4 Touch ADC Error Code Table	356
31 UDC LIBRARY	357
31.1 UDC Library Overview	357
31.2 Definition	358
31.3 API function	361
32 USB CORE LIBRARY	376
32.1 USB Core Library Overview	376
32.2 Definition	376
32.3 API function	376
32.4 USB 1.1 Host Like API function	376
32.5 USB 1.1 & 2.0 Host API function	377
33 VPE LIBRARY	387
33.1 VPE Library Overview	387
33.2 Definition	388
33.3 API function	389
33.4 VPE Error Code Table	394
34 VPOST LIBRARY	395
34.1 VPOST Library Overview	395
34.2 Definition	395
34.3 API function	400
34.4 Error Code Table	412
35 SUPPORTING RESOURCES	414

1 Introduction

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

2 AAC Library

2.1 AAC Library Overview

This library is designed to make user application to use N9H26 AAC IMDCT/MDCT more easily.

The AAC library has the following features:

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

N9h26 Non-OS BSP provide one library and one sample code to test AAC IMDCT/MDCT function. User could use them to verify hardware IP.

2.2 AAC API

DrvAAC_Open

Synopsis

```
ERRCODE  
DrvAAC_Open (void)
```

Description

This function enables the AAC engine clock.

Parameter

None

Return Value

E_OK

Example

```
DrvAAC_Open();
```

DrvAAC_Close

Synopsis

```
void DrvAAC_Close (void)
```

Description

This function disables the AAC engine clock.

Parameter

None

Return Value

None

Example

```
DrvAAC_Close();
```

DrvAAC_Decoder**Synopsis**

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

Description

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

Parameter

i32Size [in]
2048 or 256

pi32inbuf [in]
The input encoded data.

pi32outbuf [in]
The output data by running AAC IMDCT of decoder.

Return Value

The size of output buffer in byte (size x 4).

Example

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

DrvAAC_Encoder**Synopsis**

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

)

Description

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

Parameter

pi32inbuf [in]

The input encoded data.

pi32outbuf [out]

The output data by running AAC MDCT of encoder.

i32Size [in]

2048 or 256

Return Value

The size of output buffer in byte (size x 4).

Example

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

3 AES Library

3.1 AES Library Overview

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

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

3.2 Definition

3.2.1 Constant

KEYSIZE

Name	Value	Description
KEY_128	0	128-bit key size
KEY_192	1	192-bit key size
KEY_256	2	256-bit key size

Table 3-1 AES Key Size Definition

3.3 API Function

AES_Initial

Synopsis

void AES_Initial (void)

Description

Initialize AES engine and install interrupt service routine.

Parameter

None

Return Value

None

Example

None

AES_Final

Synopsis

void AES_Final (void)

Description

Tear down AES engine.

Parameter

None

Return Value

None

Example

None

AES_Encrypt

Synopsis

```
int
AES_Encrypt (
    UINT8 *cipher_buf,
    UINT8 *plain_buf,
    UINT32 data_len,
    UINT8 *iv,
    UINT8 *key,
    KEYSIZE key_size
)
```

Description

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

Parameter

plain_buf [in]

4-byte aligned address of input buffer

cipher_buf [out]

4-byte aligned address of output buffer. If NULL, cipher_buf = plain_buf

data_len [in]

Length of input buffer in bytes

iv [in]

16-byte initialization vector

key [in]

16-, 24-, or 32-byte key buffer

key_size [in]

key size as defined in Constant

KEYSIZE

Return Value

Success	Operation done
AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in Constant
KEYSIZE	
AES_ERR_IV	NULL initialization vector
AES_ERR_MODE	Wrong AES operation mode
AES_ERR_BUS_ERROR	Encounter bus error

Example

None

AES_Encrypt_Async

Synopsis

```
int
AES_Encrypt_Async (
    UINT8 *plain_buf,
    UINT8 *cipher_buf,
    UINT32 data_len,
    UINT8 *iv,
    UINT8 *key,
    KEYSIZE key_size
)
```

Description

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

Parameter

plain_buf [in]

4-byte aligned address of input buffer.

cipher_buf [out]

4-byte aligned address of output buffer. If NULL, cipher_buf = plain_buf.

data_len [in]

Length of input buffer in bytes.

iv [in]

16-byte initialization vector.

key [in]

16-, 24-, or 32-byte key buffer.

key_size [in]

key size as defined in Constant
KEYSIZE.

Return Value

Success	Operation done
AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY KEYSIZE	Key size not defined in Constant
AES_ERR_IV	NULL initialization vector
AES_ERR_RUNNING	Operation is on-going. Wait by AES_Flush or Poll by AES_Check_Status
AES_ERR_MODE	Wrong AES operation mode

Example

None

AES_Decrypt

Synopsis

```
int
AES_Decrypt (
    UINT8 *cipher_buf,
    UINT8 * plain_buf,
    UINT32 data_len,
    UINT8 *iv,
    UINT8 *key,
```

KEYSIZE key_size

)

Description

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

Parameter

cipher_buf [in]

4-byte aligned address of input buffer

plain_buf [out]

4-byte aligned address of output buffer. If NULL, plain_buf = cipher_buf

data_len [in]

Length of input buffer in bytes

iv [in]

16-byte initialization vector

key [in]

16-, 24-, or 32-byte key buffer

key_size [in]

key size as defined in Constant

KEYSIZE

Return Value

Success	Operation done
AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in Constant
KEYSIZE	
AES_ERR_IV	NULL initialization vector
AES_ERR_MODE	Wrong AES operation mode
AES_ERR_BUS_ERROR	Encounter bus error

Example

None

AES_Decrypt_Async

Synopsis

int

```
AES_Decrypt_Async (
    UINT8 *cipher_buf,
    UINT8 *plain_buf,
    UINT32 data_len,
    UINT8 *iv,
    UINT8 *key,
    KEYSIZE key_size
)
```

Description

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

Parameter

```
cipher_buf [in]
    4-byte aligned address of input buffer
plain_buf [out]
    4-byte aligned address of output buffer. If NULL, plain_buf = cipher_buf
data_len [in]
    Length of input buffer in bytes
iv [in]
    16-byte initialization vector
key [in]
    16-, 24-, or 32-byte key buffer
key_size [in]
    key size as defined in Constant
    KEYSIZE
```

Return Value

Success	Operation done
AES_ERR_DATA_LEN	Data length is not 16-byte aligned
AES_ERR_DATA_BUF	Address of input buffer is NULL
AES_ERR_CIPHER_KEY	Key size not defined in Constant
KEYSIZE	
AES_ERR_IV	NULL initialization vector
AES_ERR_RUNNING	Operation is on-going. Wait by AES_Flush or Poll by AES_Check_Status

AES_ERR_MODE

Wrong AES operation mode

Example

None

AES_Flush

Synopsis

int

AES_Flush (void)

Description

Wait for operation done

Parameter

None

Return Value

Success

Operation done

AES_ERR_BUS_ERROR

Encounter bus error

Example

None

AES_Check_Status

Synopsis

int

AES_Check_Status ()

Description

Check operation status.

Parameter

None

Return Value

Success

Operation done

AES_ERR_BUS_ERROR

Encounter bus error

AES_ERR_BUSY

Operation busy

Example

None

AES_Enable_Interrupt

Synopsis

void AES_Enable_Interrupt (void)

Description

Enable the only interrupt source.

Parameter

None

Return Value

None

AES_Disable_Interrupt

Synopsis

void AES_Disable_Interrupt (void)

Description

Disable the only interrupt source.

Parameter

None

Return Value

None

Example

None

3.4 AES Error Code Table

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

Table 3-2 AES Error Code Table

4 Audio ADC Library

4.1 Audio ADC Library Overview

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

4.2 Definition

4.2.1 Constant

Input Device

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

Table 4-1 Input Device Definition

Output Target Level

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

Table 4-2 Output Target Level Definition

Sampling Rate

Name	Value	Description
eAUR_SPS_48000	48000	48K sampling rate
eAUR_SPS_44100	44100	44.1K sampling rate
eAUR_SPS_32000	32000	32K sampling rate
eAUR_SPS_24000	24000	24K sampling rate
eAUR_SPS_22050	22050	22K sampling rate
eAUR_SPS_16000	16000	16K sampling rate
eAUR_SPS_12000	12000	12K sampling rate
eAUR_SPS_11025	11025	11.025K sampling rate
eAUR_SPS_8000	8000	8K sampling rate
eAUR_SPS_96000	96000	96K sampling rate
eAUR_SPS_192000	192000	192K sampling rate

Table 4-3 Sampling Rate Definition

Sigma-Delta Register Address

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

Table 4-4 Sigma-Delta Register Address Definition

Digital Gain

Name	Value	Description
eAUR_DIGI_MIC_GAIN_P0	0	+0db
eAUR_DIGI_MIC_GAIN_P1P6	1	+1.6db
eAUR_DIGI_MIC_GAIN_P3P2	2	+3.2db
eAUR_DIGI_MIC_GAIN_P4P8	3	+4.8db
eAUR_DIGI_MIC_GAIN_P6P4	4	+6.4db
eAUR_DIGI_MIC_GAIN_P8	5	+8db
eAUR_DIGI_MIC_GAIN_P9P6	6	+9.6db

Name	Value	Description
eAUR_DIGI_MIC_GAIN_P11P2	7	+11.2db
eAUR_DIGI_MIC_GAIN_P12P8	8	+12.8db
eAUR_DIGI_MIC_GAIN_P14P4	9	+14.4db
eAUR_DIGI_MIC_GAIN_P16	10	+16db
eAUR_DIGI_MIC_GAIN_P17P6	11	+17.6db
eAUR_DIGI_MIC_GAIN_P19P2	12	+19.2db
eAUR_DIGI_MIC_GAIN_P20P8	13	+20.8db
eAUR_DIGI_MIC_GAIN_P22P4	14	+22.4db
eAUR_DIGI_MIC_GAIN_P24	15	+24db

Table 4-5 Digital Gain Definition

Interface Between Audio ADC and EDMA

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

Table 4-6 Interface Between Audio ADC and EDMA Definition

PCM Data Format

Name	Value	Description
eAUR_ORDER_MONO_32BITS	0	Mono little endian 32 bits signed PCM
eAUR_ORDER_MONO_16BITS	1	Mono little endian 16 bits signed PCM
eAUR_ORDER_STEREO_16BITS	2	Stereo little endian 16 bits signed PCM
eAUR_ORDER_MONO_24BITS	3	(Non-standard 24 bits PCM)

Table 4-7 Supported PCM Data Format Definition

4.3 API function

DrvAUR_Open

Synopsis

INT32

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

Description

This function is used to open the Audio ADC library.

Parameter

eIntType [in]

Input device type. Please refer Table 4-1 Input Device.

blsCoworkEDMA [in]

Corporate with EDMA driver to receiver audio data.

TRUE: Enable corporation with EDMA.

FALSE: Disable corporation with EDMA.

The parameter should be always equal to TRUE.

Return Value

Successful

Example

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

DrvAUR_Close

Synopsis

INT32

DrvAUR_Close (void)

Description

Close the Audio ADC library.

Parameter

None

Return Value

Successful

Example

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

DrvAUR_InstallCallback

Synopsis

INT32

DrvAUR_InstallCallback (
 PFN_AUR_CALLBACK pfnCallback,

```
PFN_AUR_CALLBACK *pfnOldCallback
)
```

Description

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

Parameter

```
pfnCallback [in]
    The callback function wants to register
pfnOldCallback [out]
    old callback function
```

Return Value

Successful

Example

None

DrvAUR_EnableInt

Synopsis

```
void DrvAUR_EnableInt (void)
```

Description

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

Parameter

None

Return Value

None

Example

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

DrvAUR_DisableInt

Synopsis

```
void DrvAUR_DisableInt (void)
```

Description

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

Parameter

None

Return Value

None

Example

```
DrvAUR_DisableInt(); /* Disable interrupt if specified audio length done */
```

DrvAUR_AutoGainCtrl

Synopsis

```
INT32
DrvAUR_AutoGainCtrl (
    BOOL blsEnable,
    BOOL blsChangeStep,
    E_AUR_AGC_LEVEL eLevel
)
```

Description

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

Parameter

blsEnable [in]

Enable AGC or not.

blsChangeStep [in]

To trace the output target level, AGC algorithm change gain for each step.

eLevel [in]

Output target level. Please refer Table 4-2 Output Target Level.

Return Value

Successful

Example

None

DrvAUR_AutoClampingGain

Synopsis

```
INT32
DrvAUR_AutoClampingGain (
```

```

        UINT32 u32MaxGain,
        UINT32 u32MinGain
    )

```

Description

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

Parameter

u32MaxGain [in]

Maximum gain to clamp AGC. The value is from 0 ~15.

u32MinGain [in]

Minimum gain to clamp AGC. The value is from 0 ~15.

Return Value

Successful

Example

None

DrvAUR_SetSampleRate

Synopsis

```

    INT32
    DrvAUR_SetSampleRate (
        E_AUR_SPS eSampleRate
    )

```

Description

This function is used to set sampling rate.

Parameter

eSampleRate [in]

Sampling rate from 8K to 192K. Please refer Table 4-3 Sampling Rate.

Return Value

Successful

Example

None

DrvAUR_AudioI2cRead

Synopsis

```

INT32
DrvAUR_AudioI2cRead (
    UINT32 u32Addr,
    UINT8 *p8Data
)
    
```

Description

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

Parameter

u32Addr [in]
 Register address. Please refer Table 4-4 Sigma-Delta Register Address.

p8Data [out]
 Register content after read back.

Return Value

Successful

Example

None

DrvAUR_AudioI2cWrite

Synopsis

```

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

Description

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

Parameter

u32Addr [in]
 Register address. Please refer Table 4-4 Sigma-Delta Register Address.

u32Data [in]
 The content wants to program sigma-delta.

Return Value

Successful

Example

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

DrvAUR_SetDigiMicGain

Synopsis

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

Description

This function is used to set digital gain if input device is digital MIC. It is only for input device is Mono Digital MIC In or Stereo Digital MIC In. Please refer Table 4-1 Input Device.

Parameter

- blsEnable [in]
Enable digital gain for digital MIC.
- eDigiGain [in]
Digital gain. Please refer Table 4-5 Digital Gain.

Return Value

None

Example

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

DrvAUR_StartRecord

Synopsis

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

Description

Start-up sigma-delta ADC to converse audio data.

Parameter

- eMode [in]
Only eAUR_MODE_1 can be set if corporate with EDMA.

Please refer Table 4-6 Interface Between Audio ADC and EDMA.

Return Value

None

Example

```
DrvAUR_StartRecord(eAUR_MODE_1);
```

DrvAUR_StopRecord

Synopsis

```
VOID DrvAUR_StopRecord (void)
```

Description

Stop record

Parameter

None

Return Value

None

Example

```
DrvAUR_StopRecord();
```

DrvAUR_SetDataOrder

Synopsis

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

Description

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

Parameter

eOrder [in]

PCM data format. Please refer Table 4-7 Supported PCM Data Format.

Return Value

None

Example

```
DrvAUR_SetDataOrder(eAUR_ORDER_MONO_16BITS);
if(eMicType == eAUR_MONO_MIC_IN){
    DrvAUR_AudioI2cWrite(0x22, 0x1E); /* Adjust Pre-gain */
    DrvAUR_AudioI2cWrite(0x23, 0x0E); /* Adjust Post-gain*/
```

```

DrvAUR_DisableInt();
DrvAUR_SetSampleRate(aArraySampleRate[i32Idx]);
DrvAUR_AutoGainTiming(1,1,1);
DrvAUR_AutoGainCtrl(TRUE, TRUE, eAUR_OTL_N12P6);
}else if((eMicType == eAUR_MONO_DIGITAL_MIC_IN) ||
(eMicType == eAUR_STEREO_DIGITAL_MIC_IN)){
DrvAUR_SetDigiMicGain(TRUE, eAUR_DIGI_MIC_GAIN_P19P2);
DrvAUR_DisableInt();
DrvAUR_SetSampleRate(eAUR_DIGI_MIC_GAIN_P19P2);
}

```

DrvAUR_AutoGainTiming

Synopsis

```

INT32
DrvAUR_AutoGainTiming (
    UINT32 u32Attack,
    UINT32 u32Recovery,
    UINT32 u32Hold
)

```

Description

This function is used to set the timing for tracing the gain as enable AGC. Programmer can use the API to adjust the timing for tracing gain if AGC is enabled. The time unit is base on 64 audio sampling.

Parameter

u32Attack [in]
The time of one step to shrink the AGC gain.

u32Recovery [in]
The time of one step to enlarge the AGC gain.

u32Hold [in]
The time to hold the AGC gain.

Return Value

Successful

Example

None

5 AVI Library

5.1 AVI Library Overview

5.1.1 Video render

N9H26 can support JPEG decoder to output decoded packet data in DIRECT_RGB555, DIRECT_RGB565, DIRECT_RGB888 or DIRECT_YUV422 format. User application must initialize VPOST as corresponding format specified in AVI function call `aviPlayFile(...)`. AVI player library will configure JPEG output format as specified format and use DMA to copy the decoded data to VPOST frame buffer in Vsync period to avoid the tearing issue.

In this way, three frame buffers are required. One is allocated in VPOST initialized function and two buffers are allocated in AVI library.

5.1.2 How to use AVI player library

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

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

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

- Direct RGB555 – VPOST should select `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`

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

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

5.1.3 AVI player user callback

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

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

5.1.4 AVI playback information

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

5.2 Definition

5.2.1 Constant

Video

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

Table 5-1 Video Definition

Audio

Name	Value	Description
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

Table 5-2 Audio Definition

5.2.2 Structure

AVI_INFO_T Structure

Field name	Data 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 5-3 AVI_INFO_T Structure Definition

5.3 API function

aviStopPlayFile

Synopsis

void aviStopPlayFile (void)

Description

Stop current AVI file playback.

Parameter

None

Return Value

None

Example

None

aviPlayFile

Synopsis

int

```

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

Description

Play an AVI file.

Parameter

suFileName [in]

The full path file name of input AVI file.

x [in]

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

y [in]

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

mode [in]

Video render mode.

cb_func [in]

User application callback function.

Return Value

Success Successful

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 [out]
Return AVI parsing information.

Return Value

Success	Successful
ERRCODE	Error

Example

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

aviSetPlayVolume**Synopsis**

```
void aviSetPlayVolume (
    int vol
)
```

Description

Set the Left channel and Right channel playback audio volume.

Parameter

vol [in]
The audio volume

Return Value

None

Example

```
aviSetPlayVolume(0x1F);
```

aviSetRightChannelVolume

Synopsis

```
void aviSetRightChannelVolume (  
    int vol  
)
```

Description

Set the Right channel audio playback volume only.

Parameter

vol [in]
The audio volume

Return Value

None

Example

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

aviSetLeftChannelVolume

Synopsis

```
void aviSetLeftChannelVolume (  
    int vol  
)
```

Description

Set the Left channel audio playback volume only.

Parameter

vol [in]
The audio volume

Return Value

None

Example

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

5.4 AVI 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

Code Name	Value	Description
MFL_ERR_UNKNOWN_MEDIA	0xFFFF81E2	unknow media type

Table 5-4 AVI Error Code Table

6 MP3 Library

6.1 MP3 Library Overview

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

6.1.1 How to use MP3 player Library?

The MP3 player required user to follow the steps before playing an MP3 file:

- Initialize cache.
- Initialize UART.
- Initialize timer.
- Initialize file system.
- Initialize storage device.
- Initialize audio device.

6.1.2 MP3 player user callback

"ap_time", the member of structure MV_CFG_T can excute user defined API. Any time information or control can be handled in it.

6.1.3 MP3 player information

Structure MV_INFO_T will give you the time information, include current time and total time.

6.2 Definition

6.2.1 Constant

MP3 Definition

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

Table 6-1 MP3 Definition

6.2.2 Structure

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

Table 6-2 MV_CFG_T Structure Definition

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

Table 6-3 MV_INFO Structure Definition

6.3 API function

mflMediaPlayer

Synopsis

```

INT
mflMediaPlayer (
    MV_CFG_T *ptMvCfg
)
    
```

Description

Start play MP3 file.

Parameter

ptMvCfg [in]

The MV_CFG_T structure

Return Value

Success Successful

ERRCODE Error

Example

```
mflMediaPlayer(&_tMvCfg)
```

mflGetMovieInfo

Synopsis

INT

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

Description

Get MP3 time information.

Parameter

ptMvCfg [in]

The MV_CFG_T structure

ptMvInfo [out]

The MV_INFO_T structure

Return Value

Success Successful

Example

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

mflPlayControl

Synopsis

INT

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

)

Description

Control operation while playing MP3 file.

Parameter

ptMvCfg [in]

The MV_CFG_T structure.

ePlayCtrl [in]

The PLAY_CTRL_E enumeration.

nParam [in]

Reserved

Return Value

Success Successful

Example

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

7 BLT Library

7.1 BLT Library Overview

BLT supports the following features

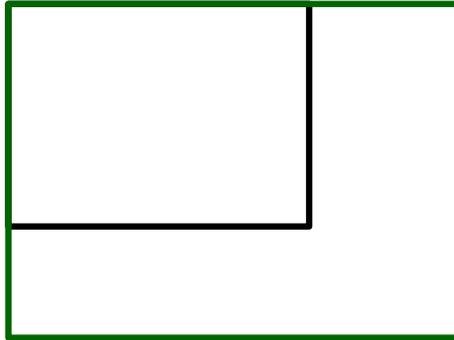
- Fill operation.
 - Fill color with alpha channel
- Blit operation
 - Transformation effects (Scaling, Rotation, Shearing, etc.) through 2x2 inverse transformation matrix.
 - Bitmap smoothing in bi-linear algorithm.
 - Tiling mode (for inversely mapped source pixels lying outside the boundaries of the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

7.1.1 Transformation Matrix

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

Scaling

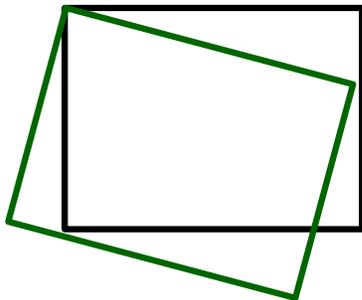
Resize the image by multiplying the location of each pixel by s_x on the x axis and s_y on the y axis.



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

Rotation

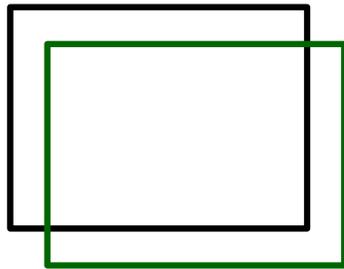
Rotate the image by an angle θ clockwise.



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

Translation

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



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

7.1.2 Amendment to User Transformation Matrix

On mapping back from destination CS¹ to source CS, a mapping point of destination pixel must be taken into consideration. Mapping point can be top-left (Top-left point as mapping point of destination pixel) or center (Center point as mapping point of destination pixel). In the blit implementation, top-left point is chosen and we may encounter an error due to the choice of mapping point. To help explain the issue, an example is given: blit with rotate 180°.

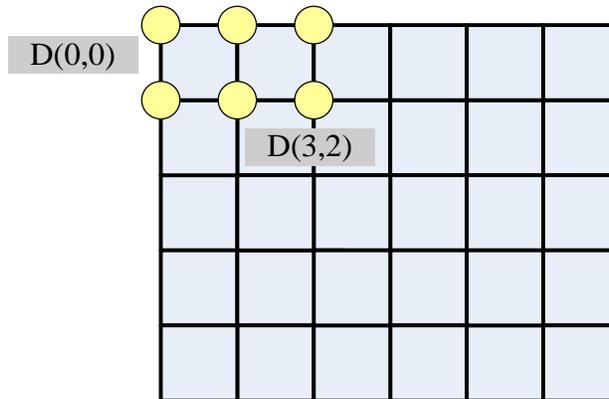


Figure 7-1 Top-left point as mapping point of destination pixel

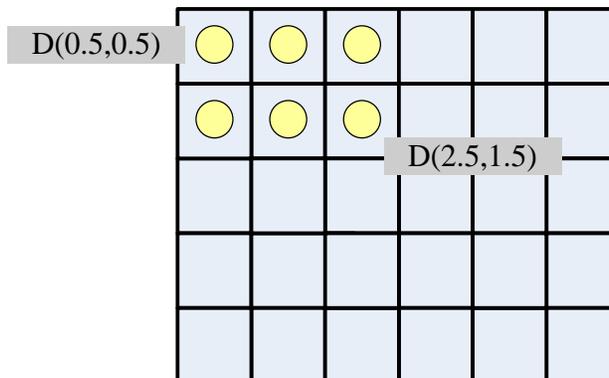


Figure 7-2 Center point as mapping point of destination pixel

1. User wants to blit **Error! Reference source not found.** and get **Error! Reference source not found.** result, This blit operation involves rotation and translation applied to the source image.

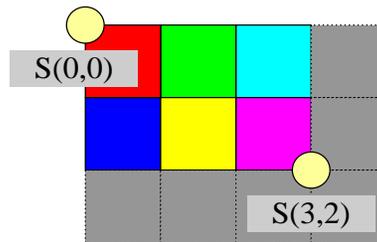


Figure 7-3 Source image

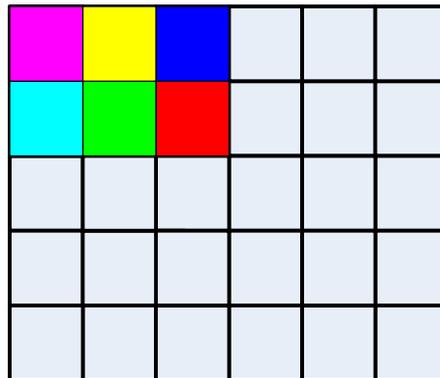


Figure 7-4 Final

2. First, just copy without any transformation effect and get **Error! Reference source not found.** result.

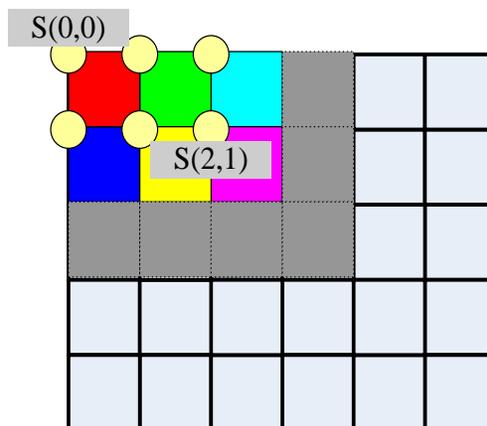


Figure 7-5 No transform

3. Rotate 180° clockwise and get **Error! Reference source not found.** result.

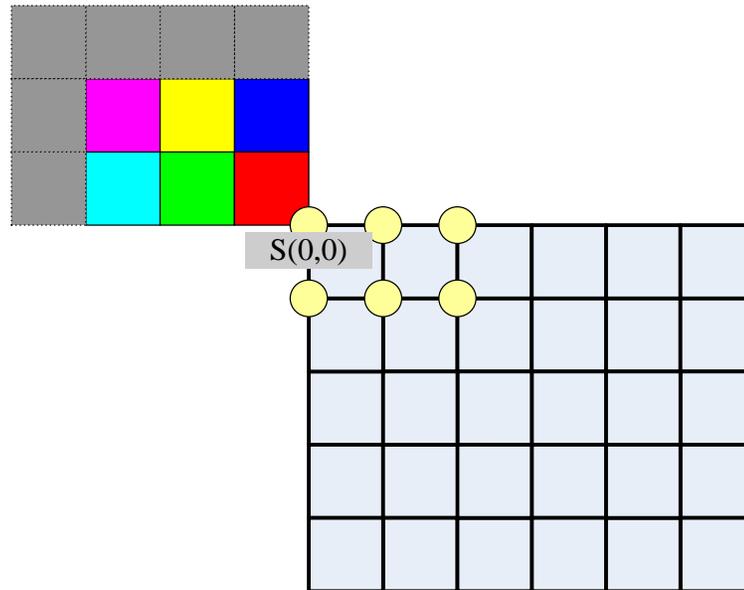


Figure 7-6 Rotate 180°

- Translate 3 along x-axis and 2 along y-axis and get **Error! Reference source not found.** 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.

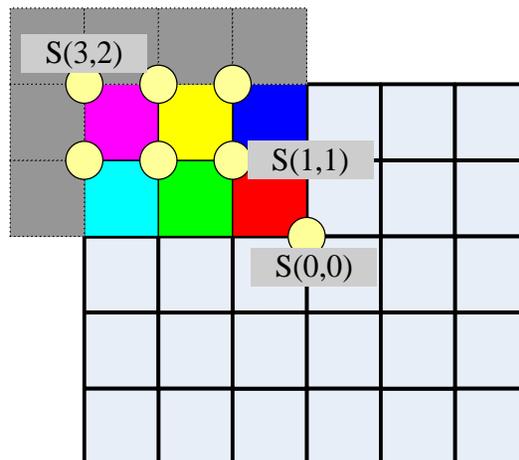


Figure 7-7 Rotate 180° + Translate (3, 2)

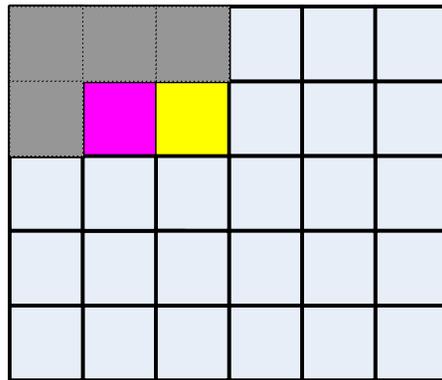


Figure 7-8 Rotate 180° + Translate (3, 2) (2)

- 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 Error! Reference source not found. 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.

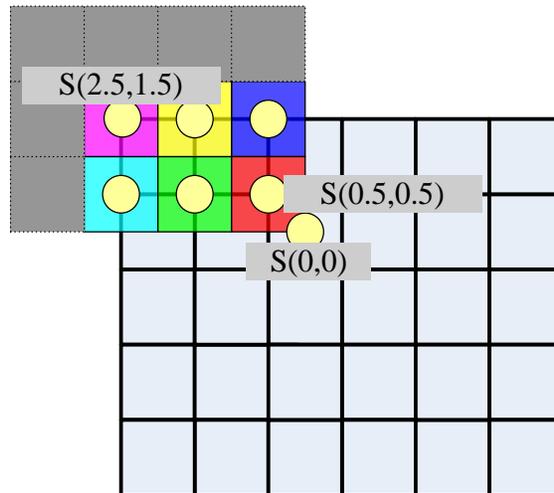


Figure 7-9 Rotate 180° + Translate (3, 2) + Translate (-0.5, -0.5)

7.1.3 Pixel Mapping

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

- Elements a, b, c, and d in S_DRVBLT_MATRIX.
- i32XOffset and i32YOffset in S_DRVBLT_SRC_IMAGE.

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

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

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

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

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

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

1. In M0, D(0, 0) (origin pixel of destination CS) is inversely mapped to S(1, 1), which needn't be the origin pixel of the source image. D(0, 0) is filled with **Red** color.
2. In M1, D(1, 1) is inversely mapped to S(2, 2), which lies inside the source image. D(1, 1) is filled with **Green** color.
3. In M2, D(4, 2) is inversely mapped to S(5, 3), which lies outside the source image. Dependent on tiling mode specified in E_DRVBLT_FILL_STYLE, there are 3 different rendering results:
 - ◆ No drawing: D(4, 2) is not drawn.
 - ◆ Clip to edge: D(4, 2) is inversely mapped to S(3, 3). D(4, 2) is filled with **Blue** color.
 - ◆ 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.

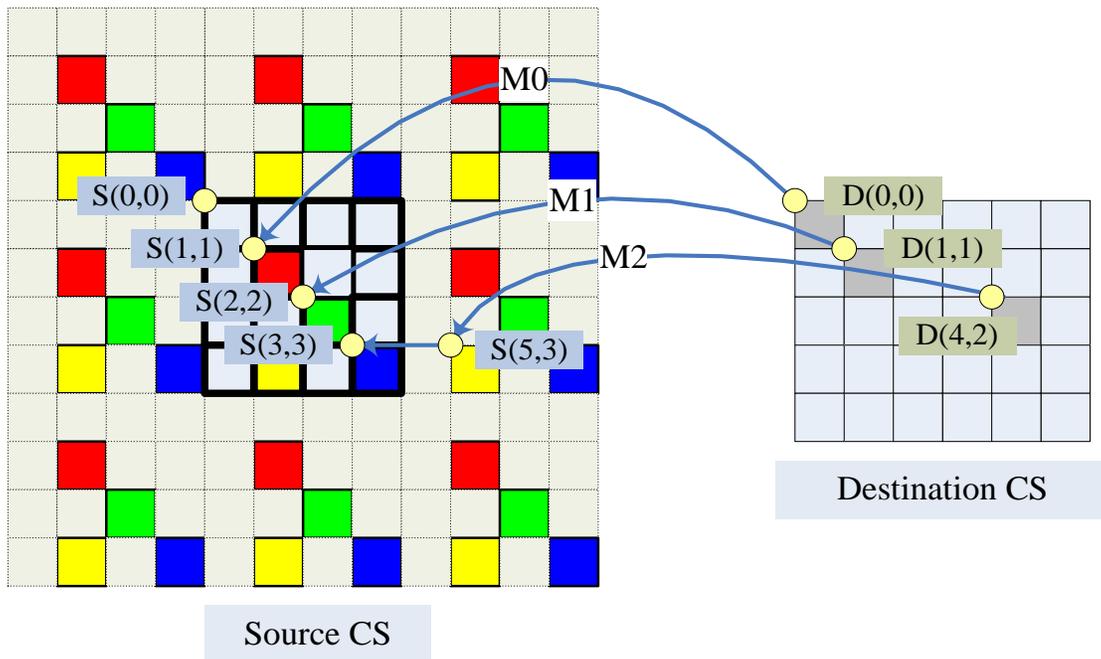


Figure 7-10 Mapping from destination CS to source CS

7.1.4 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.

$$\begin{aligned} \text{New alpha value} &= (\text{old alpha value} * \text{alphaMultiplier}) + \text{alphaOffset} \\ \text{New red value} &= (\text{old red value} * \text{redMultiplier}) + \text{redOffset} \\ \text{New green value} &= (\text{old green value} * \text{greenMultiplier}) + \text{greenOffset} \\ \text{New blue value} &= (\text{old blue value} * \text{blueMultiplier}) + \text{blueOffset} \end{aligned}$$

7.1.5 Palette

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

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

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

Palette entries

Palette ranges from BLT_BA+0x400. Its format is ARGB8888, premultiplied-alpha by default and user can change it by setting up the field SET2DA.S_ALPHA.

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

Source image in palette index format

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

For example,

One byte in image data=b7b6b5b4b3b2b1b0

Index size=2-bit

Index order=big-endian (SET2DA.L_ENDIAN=0) →

b7b6=1st pixel, b5b4=2nd pixel, b3b2=3rd pixel, b1b0=4th pixel

Index order=little-endian (SET2DA.L_ENDIAN=1) →

b7b6=4th pixel, b5b4=3rd pixel, b3b2=2nd pixel, b1b0=1st pixel

7.2 Definition

7.2.1 Constant

E_BLT_INT_TYPE

Name	Value	Description
BLT_INT_CMPLT	1	Fill/Blit operation completed
BLT_INT_PGFLT	2	BLT MMU Page Fault
BLT_INT_PGMS	3	BLT MMU Page Miss

Table 7-1 Interrupt type Definition

E_DRVBLT_FILLOP

Name	Value	Description
eDRVBLT_DISABLE	0	Blit operation
eDRVBLT_ENABLE	1	Fill operation

Table 7-2 Fill or Blit operation Definition

E_DRVBLT_REVEAL_ALPHA

Name	Value	Description
eDRVBLT_EFFECTIVE	0	Premultiplied alpha
eDRVBLT_NO_EFFECTIVE	1	Non-premultiplied alpha

Table 7-3 Premultiplied alpha or not for source format of ARGB8888 Definition

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_NONTRANSPARENCYE	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.

Table 7-4 Transform flags for Blit operation Definition

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

Table 7-5 Source format for Blit operation Definition

E_DRVBLT_DISPLAY_FORMAT

Name	Value	Description
eDRVBLT_DEST_ARGB8888	1	ARGB8888
eDRVBLT_DEST_RGB565	2	RGB565
eDRVBLT_DEST_RGB555	4	RGB555

Table 7-6 Destination format for Fill/Blit operation Definition

E_DRVBLT_FILL_STYLE

Other flags for Blit operation. eDRVBLT_CLIP_TO_EDGE / eDRVBLT_NONE_FIL specify how to behave when reverse mapping doesn't fall in the range of source bitmap.

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

Table 7-7 Other flags for Blit operation Definition

E_DRVBLT_PALETTE_ORDER

Other flags for Blit operation. eDRVBLT_CLIP_TO_EDGE / eDRVBLT_NONE_FIL specify how to behave when reverse mapping doesn't fall in the range of source bitmap.

Name	Value	Description
eDRVBLT_BIG_ENDIAN	0	Palette index in big endian
eDRVBLT_LITTLE_ENDIAN	1	Palette index in little endian

Table 7-8 Palette index in big-endian or little-endian Definition

7.2.2 Structure

S_DRVBLT_MATRIX

Other flags for Blit operation. eDRVBLT_CLIP_TO_EDGE / eDRVBLT_NONE_FIL specify how to behave when reverse mapping doesn't fall in the range of source bitmap.

Field name	Data Type	Description
a	INT32	Matrix a
b	INT32	Matrix b
c	INT32	Matrix c
d	INT32	Matrix d

Table 7-9 Palette index in big-endian or little-endian Structure

S_DRVBLT_ARGB16

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

Field name	Data Type	Description
i16Blue	INT16	Color multiplier/offset of blue channel
i16Green	INT16	Color multiplier/offset of green channel
i16Red	INT16	Color multiplier/offset of red channel
i16Alpha	INT16	Color multiplier/offset of alpha channel

Table 7-10 Transformation Structure

S_DRVBLT_ARGB8

Field name	Data Type	Description
u8Blue	UINT8	Value of blue channel
u8Green	UINT8	Value of green channel
u8Red	UINT8	Value of red channel
u8Alpha	UINT8	Value of alpha channel

Table 7-11 ARGB8888 color Structure

S_DRVBLT_SRC_IMAGE

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

Table 7-12 Source image Structure

S_DRVBLT_DEST_FB

Field name	Data Type	Description
u32FrameBufAddr	UINT32	Destination buffer address to start rendering to
i32XOffset	INT32	No use
i32YOffset	INT32	No use
i32Stride	INT32	Destination buffer's stride in bytes
i16Width	INT16	Destination buffer's width in pixels
i16Height	INT16	Destination buffer's height in pixels

Table 7-13 Destination buffer Structure

7.3 API function

bltOpen

Synopsis

ERRCODE
bltOpen (void)

Description

Initialize BLT and install interrupt service routine.

Parameter

None

Return Value

E_SUCCESS Success

Example

None

bltClose

Synopsis

void bltClose (void)

Description

Tear down BLT.

Parameter

None

Return Value

None

Example

None

bltSetTransformMatrix**Synopsis**

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

Description

Set up inverse transformation matrix.

Parameter

sMatrix [in]

Transformation matrix as defined in S_DRVBLT_MATRIX.

Return Value

None

Example

None

bltGetTransformMatrix**Synopsis**

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

Description

Retrieve inverse transformation matrix which has set up.

Parameter

psMatrix [out]

User-prepared buffer to save read-back transformation matrix as defined in S_DRVBLT_MATRIX.

Return Value

None

Example

None

bltSetSrcFormat

Synopsis

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

Description

Set up source format.

Parameter

eSrcFmt [in]
 Source format as defined in E_DRVBLT_BMPIXEL_FORMAT.

Return Value

E_SUCCESS	Success
ERR_BLT_INVALID_SRCFMT	Invalid source format

Example

None

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.

Example

None

bltSetDisplayFormat

Synopsis

```

ERRCODE
bltSetDisplayFormat (
    E_DRVBLT_DISPLAY_FORMAT eDisplayFmt
)
    
```

Description

Set up destination format.

Parameter

eDisplayFmt [in]
 Destination format defined in E_DRVBLT_DISPLAY_FORMAT.

Return Value

E_SUCCESS	Success
ERR_BLT_INVALID_DSTFMT	Invalid destination format

Example

None

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.

Example

None

bltEnableInt

Synopsis

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

Description

Enable specified interrupt type.

Parameter

eIntType [in]
Interrupt type as defined in Constant
E_BLT_INT_TYPE.

Return Value

None

Example

None

bltDisableInt**Synopsis**

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

Description

Disable specified interrupt type.

Parameter

eIntType [in]
Interrupt type as defined in Constant
E_BLT_INT_TYPE.

Return Value

None

Example

None

bltIsIntEnabled**Synopsis**

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

```

        E_BLT_INT_TYPE eIntType
    )

```

Description

Query if the specified interrupt type is enabled.

Parameter

eIntType [in]
 Interrupt type as defined in Constant
 E_BLT_INT_TYPE.

Return Value

TRUE	Specified interrupt enabled
FALSE	Specified interrupt disabled

Example

None

bltPollInt

Synopsis

```

    BOOL
    bltPollInt (
        E_BLT_INT_TYPE eIntType
    )

```

Description

Query interrupt status of the specified interrupt type.

Parameter

eIntType [in]
 Interrupt type as defined in Constant
 E_BLT_INT_TYPE.

Return Value

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

Example

None

bltInstallCallback

Synopsis

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

Description

Install callback function invocated on interrupt generated.

Parameter

eIntType [in]
 Interrupt type as defined in Constant E_BLT_INT_TYPE.

pfnCallback [in]
 New callback function to install. NULL to uninstall.

pfnOldCallback [out]
 User-prepared buffer to save previously installed callback function.

Return Value

None

Example

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 [in]
 Color multipliers of A, R, G, and B channels as defined in S_DRVBLT_ARGB16.

Return Value

None

Example

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 [out]

User-prepared buffer to save color multipliers of A, R, G, and B channels as defined in S_DRVBLT_ARGB16.

Return Value

None

Example

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 [in]

Color offsets of A, R, G, and B channels as defined in S_DRVBLT_ARGB16.

Return Value

None

Example

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 [out]

User-prepared buffer to save color offsets of A, R, G, and B channels as defined in S_DRVBLT_ARGB16.

Return Value

None

Example

None

bltSetSrcImage

Synopsis

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

Description

Set up source image.

Parameter

sSrcImage [in]

Source image as defined in S_DRVBLT_SRC_IMAGE.

Return Value

None

Example

None

bltSetDestFrameBuf

Synopsis

```
void bltSetDestFrameBuf (
```

```

        S_DRVBLT_DEST_FB sFrameBuf
    )

```

Description

Set up destination buffer.

Parameter

sFrameBuf [in]

Destination buffer as defined in S_DRVBLT_DEST_FB.

Return Value

None

Example

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 [in]

Fill color as defined in S_DRVBLT_ARGB8.

Return Value

None

Example

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 [out]

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

Return Value

None

Example

None

bltGetBusyStatus

Synopsis

BOOL

bltGetBusyStatus (void)

Description

Query if Fill/Blit operation is busy.

Parameter

None

Return Value

TRUE Busy

FALSE Free

Example

None

bltSetFillAlpha

Synopsis

void bltSetFillAlpha (

 BOOL bEnable

)

Description

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

Parameter

bEnable [in]

TRUE Fill color is ARGB8888
 FALSE Fill color is RGB888

Return Value

None

Example

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

Example

None

bltSetTransformFlag

Synopsis

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

Description

Set up transform flag.

Parameter

u32TransFlag [in]
 Transform flag as defined in E_DRVBLT_TRANSFORM_FLAG.

Return Value

None

Example

None

bltGetTransformFlag

Synopsis

UINT32

bltGetTransformFlag (void)

Description

Retrieve transform flag which has set up.

Parameter

None.

Return Value

Transform flag as defined in E_DRVBLT_TRANSFORM_FLAG.

Example

None

bltSetPaletteEndian

Synopsis

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

Description

Set up endianness of palette index.

Parameter

eEndian [in]

Endianness of palette index as defined in E_DRVBLT_PALETTE_ORDER.

Return Value

None

Example

None

bltGetPaletteEndian

Synopsis

```
E_DRVBLT_PALETTE_ORDER  
bltGetPaletteEndian (void)
```

Description

Retrieve endianness of palette index which has set up.

Parameter

None

Return Value

Endianness of palette index as defined in E_DRVBLT_PALETTE_ORDER.

Example

None

bltSetColorPalette**Synopsis**

```
Void bltSetColorPalette (  
    UINT32 u32PaletteIdx,  
    UINT32 u32Num,  
    S_DRVBLT_ARGB8 *psARGB  
)
```

Description

Set up palette's colors.

Parameter

u32PaletteIdx [in]
 Index of palette to start to set up
u32Num [in]
 Number of colors to set up
psARGB [in]
 ARGB8888 colors

Return Value

None

Example

None

bltSetFillOP**Synopsis**

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

Description

Set up operation to be Fill or Blit.

Parameter

eOP [in]

Operation as defined in E_DRVBLT_FILLOP.

Return Value

None

Example

None

bltGetFillOP

Synopsis

BOOL

bltGetFillOP (void)

Description

Retrieve operation which has set up.

Parameter

None

Return Value

TRUE Fill operation.

FALSE Blit operation.

Example

None

bltSetFillStyle

Synopsis

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

Description

Set up other flags for Blit operation.

Parameter

eStyle [in]

Other flags as defined in E_DRVBLT_FILL_STYLE.

Return Value

None

Example

None

bltGetFillStyle**Synopsis**

E_DRVBLT_FILL_STYLE

bltGetFillStyle (void)

Description

Retrieve other flags for Blit operation which has set up.

Parameter

None

Return Value

Other flags as defined in E_DRVBLT_FILL_STYLE.

Example

None

bltSetRevealAlpha**Synopsis**

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

Description

Set up premultiplied alpha or not for source format of ARGB8888

Parameter

eAlpha [in]

Premultiplied alpha or not as specified in E_DRVBLT_REVEAL_ALPHA.

Return Value

None

Example

None

bltGetRevealAlpha**Synopsis**

BOOL

bltGetRevealAlpha (void)

Description

Retrieve premultiplied alpha or not for source format of ARGB8888.

Parameter

None

Return Value

Premultiplied alpha or not as specified in E_DRVBLT_REVEAL_ALPHA.

Example

None

bltTrigger**Synopsis**

```
void bltTrigger (void)
```

Description

Start Fill/Blit operation.

Parameter

None

Return Value

None

Example

None

bltSetRGB565TransparentColor**Synopsis**

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

Description

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

Parameter

u16RGB565 [in]

RGB565 to be transparent color.

Return Value

None

Example

None

bltGetRGB565TransparentColor

Synopsis

UINT16

bltGetRGB565TransparentColor (void)

Description

Retrieve transparent color which has set up.

Parameter

None

Return Value

RGB565 to be transparent color

Example

None

bltSetRGB565TransparentCtl

Synopsis

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

Description

Enable color key or not.

Parameter

bEnable [in]

TRUE	Enable color key
FALSE	Disable color key

Return Value

None

Example

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

Example

None

bltFlush

Synopsis

void bltFlush (void)

Description

Wait for Fill/Blit operation to complete.

Parameter

None

Return Value

None

Example

None

7.4 BLT Error Code Table

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

Table 7-14 BLT Error Code Table

8 CRC Library

8.1 CRC Library Overview

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

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

8.2 Definition

8.2.1 Constant

E_CRC_CHANNEL_INDEX

Name	Value	Description
E_CHANNEL_0	0	CRC channel 0
E_CHANNEL_1	1	CRC channel 1

Table 8-1 CRC channel index Definition

E_CRC_OPERATION

Name	Value	Description
E_CH_DISABLE	0	CRC channel disable
E_CH_ENABLE	1	CRC channel enable

Table 8-2 CRC channel operation Definition

E_CRC_MODE

Name	Value	Description
E_CRCCCITT	0	CRC-CCITT polynomial
E_CRC8	1	CRC-8 polynomial
E_CRC16	2	CRC-16 polynomial
E_CRC32	3	CRC-32 polynomial

Table 8-3 CRC polynomials Definition

E_WRITE_LENGTH

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

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

Table 8-4 CRC data width Definition

E_DATA_1sCOM

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

Name	Value	Description
E_1sCOM_OFF	0	1's complement disable
E_1sCOM_ON	1	1's complement enable

Table 8-5 1's complement setting Definition

E_DATA_REVERSE

Order reverse setting for input data and CRC checksum.

Name	Value	Description
E_REVERSE_OFF	0	Order reverse disable
E_REVERSE_ON	1	Order reverse enable

Table 8-6 Order reverse setting Definition

E_TRANSFER_MODE

CRC CPU PIO or VDMA mode.

Name	Value	Description
E_CRC_CPU_PIO	0	CRC CPU PIO mode
E_CRC_VDMA	1	CRC VDMA mode

Table 8-7 Transfer Mode setting Definition

8.2.2 Structure

S_CRC_CHANNEL_INFO Structure

Field name	Data Type	Description
bInRequest	BOOL	CRC channel is in request or not
bInUse	BOOL	CRC channel is in use or not

Table 8-8 S_CRC_CHANNEL_INFO Structure Definition

S_CRC_DESCRIPTOR_SETTING Structure

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

Table 8-9 S_CRC_DESCRIPTOR_SETTING Structure Definition

8.3 API function

CRC_Init

Synopsis

INT32

CRC_Init (void)

Description

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

Parameter

None

Return Value

Success Always returns Successful

Example

None

CRC_Exit

Synopsis

void CRC_Exit (void)

Description

Clear CRC initial flag.

Parameter

None

Return Value

None

Example

None

CRC_Request

Synopsis

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

Description

Specify a channel for request.

Parameter

channel [in]
CRC channel number

Return Value

Success	Specified channel is requested
CRC_ERR_INVALID	Specified channel number is invalid
CRC_ERR_BUSY	Specified channel is busy
CRC_ERR_STATUS	Specified channel status is wrong

CRC_Free

Synopsis

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

Description

Release a previously acquired channel.

Parameter

channel [in]
CRC channel number

Return Value

None

Example

None

CRC_FindandRequest

Synopsis

INT32
CRC_FindandRequest (void)

Description

Try to find a free channel and request it.

Parameter

None

Return Value

Success	Allocated channel is returned
CRC_ERR_NODEV	No free channel is found

Example

None

CRC_Run

Synopsis

```

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

Description

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

Parameter

channel [in]

CRC channel number

pDataBuf [in]

Input buffer address

uDataLen [in]

Length of input buffer in bytes

psCRCDescript [out]

Pointer to the channel description of this CRC calculation

Return Value

- Success CRC checksum is returned
- CRC_ERR_STATUS Channel is not in request
- CRC_ERR_BUSY Channel is in use and cannot run a calculation

Example

None

8.4 CRC Error Code Table

Code Name	Value	Description
CRC_ERR_INVALID	CRC_ERR_ID 0x01	Channel number is invalid
CRC_ERR_NODEV	CRC_ERR_ID 0x02	No free channel is found
CRC_ERR_STATUS	CRC_ERR_ID 0x03	Channel status is wrong

Table 8-10 CRC Error Code Table

9 EDMA Library

9.1 EDMA Library Overview

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

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

9.1.1 System Overview

The N9H26 contains an enhanced direct memory access (EDMA) controller that transfers data to and from memory or transfer data to and from APB. The EDMA controller has 11-channel DMA that include 3 channel VDMA (Video-DMA, Memory-to-Memory) and 8 channels PDMA (Peripheral-to-Memory or Memory-to-Peripheral). For channel 0/5/8 VDMA mode, it also supports color format transform and stripe mode transfer. For PDMA channel (EDMA CH1~CH4, CH9~CH12), it can transfer data between the Peripherals APB IP (ex: UART, SPI, ADC....) and Memory. The N9H26 also support hardware scatter-gather function, software can set 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 N9H26 VDMA controller can increment source or destination address, decrement or fixed them as well, and the PDMA can increment source or destination, fixed or wrap around address.

9.1.2 EDMA Control

VDMA Transfer

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

Software must enable DMA channel DMA [DMACEN] and then write a valid source address to the DMA_SARx register, a destination address to the DMA_DSABx register, and a transfer count to the DMA_BCRx register. Next, trigger the DMA_CSRx [Trig_EN]. If the source address and destination are not in wrap around mode, the transfer will start transfer until DMA_CBCRx reaches zero (in wrap around mode, when DMA_CBCRx equal zero, the DMA will reload DMA_CBCRx and work around until software disable DMA_CSRx [DMACEN]). If an error occurs during the EDMA operation, the channel stops unless software clears the error condition, sets the DMA_CSRx [SW_RST] to reset the EDMA channel and set EDMA_CSRx [EDMACEN] and [Trig_EN] bits field to start again.

PDMA Transfer

The PDMA is used to transfer data between SDRAM and APB device. Currently, the APB device only supports UART 0/1, SPIMS 0/1 and ADC audio recording. The data direction can

be from APB device or to APB device dependent on the setting of PDMA_CSRx[MODE_SEL]. Hardware IP will do the necessary handshaking signal between PDMA and APB device.

In the PDMA transfer, the APB device data port should be set as the source or destination address dependent on the setting of PDMA_CSRx[MODE_SEL], and the address direction must be set as fixed for APB address. Besides this, the APB device has corresponding register setting to enable PDMA transfer.

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

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

Table 9-1 Control Register

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

Scatter Gather Transfer

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

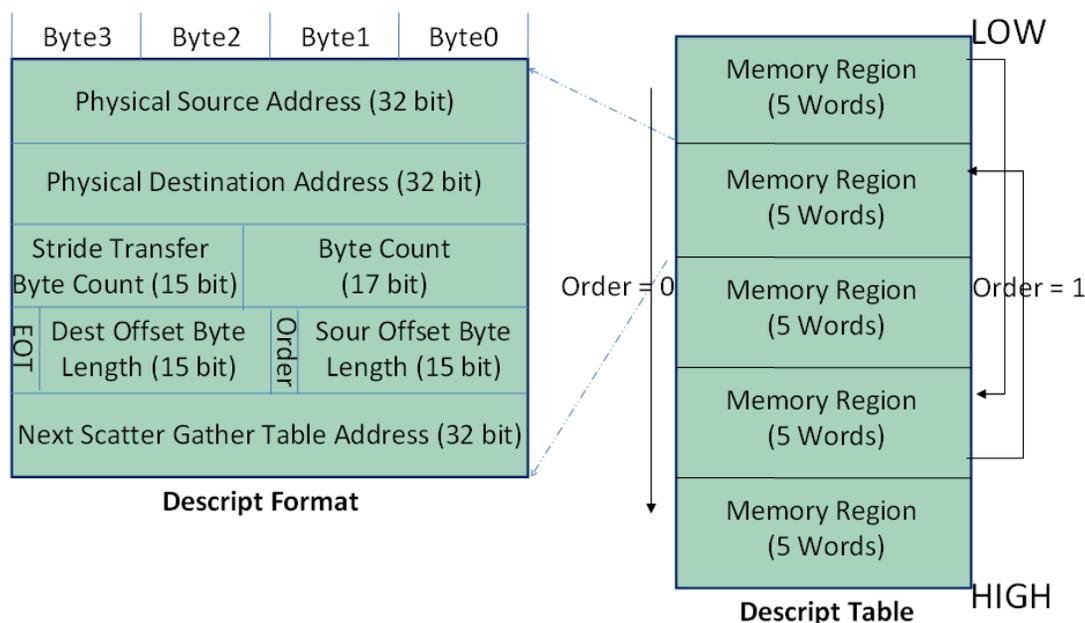


Figure 9-1 Descriptor table format

The N9H26 also support hardware scatter-gather function, software can set DMA_CSRx[SG_EN] to enable scatter-gather function. When in scatter-gather function mode, some

register will be automatically updated by descriptor table

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

9.2 API function

EDMA_Init

Synopsis

```
int
EDMA_Init (void)
```

Description

This function initializes the software resource.

Parameter

None

Return Value

successes

Example

```
EDMA_Init ();
```

EDMA_Exit

Synopsis

```
void EDMA_Exit (void)
```

Description

Disable EDMA engine clock.

Parameter

None

Return Value

None

Example

```
EDMA_Exit ();
```

VDMA_FindandRequest

Synopsis

int

VDMA_FindandRequest (void)

Description

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

Parameter

None

Return Value

Allocated channel SUCCESS

EDMA_ERR_NODEV FAIL

Example

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

PDMA_FindandRequest

Synopsis

int

PDMA_FindandRequest (void)

Description

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

Parameter

None

Return Value

Allocated channel SUCCESS

EDMA_ERR_NODEV FAIL

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 [in]
EDMA channel number

interrupt [in]
EDMA interrupt enable

irq_handler [in]
The callback function pointer for specified EDMA channel.

data [out]
User specified value to be passed to the handlers.

Return Value

SUCCESS	Success
EDMA_ERR_NODEV	FAIL

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

```

Return Value

SUCCESS	Success
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 [in]
    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 [in]
 EDMA channel number

src_addr [in]
 Source address

dest_addr [in]
 Destination address

dma_length [in]
 Total length of the transfer request in bytes

Return Value

SUCCESS	Success.
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 [in]
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 [in]
EDMA channel number
eSrcFormat [in]
The source color format
eDestFormat [in]
The destination color format

Return Value

SUCCESS	Success
EDMA_ERR_BUSY	Bus is busy.

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 [in]
EDMA channel number

Return Value

SUCCESS Success

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 [in]
EDMA channel number

Return Value

None.

Example

```
/* Trigger EDMA channel transfer */
EDMA_Trigger (g_VdmaCh);
```

EDMA_IsBusy

Synopsis

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

Description

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

Parameter

channel [in]
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 [in]
EDMA channel number
eDevice [in]
Specify the APB device which will use the EDMA channel

eRWAPB [in]

Indicate that read or write APB device

eTransferWidth [in]

Set the transfer width for specified channel

Return Value

0 SUCCESS

EDMA_ERR_BUSY FAIL

Example

```
/* Setup ADC use EDMA channel*/
EDMA_SetAPB (g_PdmaCh, eDRVEDMA_ADC, eDRVEDMA_READ_APB,
eDRVEDMA_WIDTH_32BITS);
```

EDMA_SetWrapINTType

Synopsis

```
int
DMA_SetWrapINTType (
    int channel,
    int type
)
```

Description

Set the EDMA wrap around interrupt select for specified channel.

Parameter

channel [in]

EDMA channel number

type [in]

Set the wrap around mode for specified channel

Return Value

SUCCESS Success

EDMA_ERR_BUSY Bus is busy.

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 [in]
    EDMA channel number
src_dir [in]
    The source transfer direction
dest_dir [in]
    The destination transfer direction
```

Return Value

```
SUCCESS          Success
EDMA_ERR_BUSY     Bus is busy.
```

Example

```
/* Set source transfer direction fixed and destination wraparound*/
EDMA_SetDirection (g_PdmaCh , eDRVEDMA_DIRECTION_FIXED,
eDRVEDMA_DIRECTION_WRAPAROUND);
```

9.3 EDMA 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

Table 9-2 EDMA Error Code Table

10 Font Library

10.1 Font Library Overview

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

10.2 Definition

10.2.1 Structure

Font Information Structure

Field Name	Data Type	Description
u32FontRectWidth	UINT32	Font width. Now fixed in 16
u32FontRectHeight	UINT32	Font height. Now fixed in 22
u32FontOffset	UINT32	Font Offset. Now fixed in 11
u32FontStep	UINT32	Font Step. Now fixed in 10
u32FontOutputStride	UINT32	Output Stride. It should same as the panel width
u32FontInitedDone	UINT32	1 = Font library initialized done. 0 = Font library not yet initialized done or de-initialized.
u32FontFileSize	UINT32	Useless.
pu32FontFileTmp	UINT32	Useless
pu32FontFile	UINT32	Pointer of font file
au16FontColor[3]	UINT16	RGB565 color au16FontColor[0]: Font background color au16FontColor[1]: Font color au16FontColor[2]: Border color

Table 10-1 Font Information Structure Definition

Rectangle Information Structure

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 10-2 Rectangle Information Structure Definition

10.3 API function

InitFont

Synopsis

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

Description

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

Parameter

ptFont [out]
 Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

u32FrameBufAddr [in]
 Frame buffer base address.

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 [in]

Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

u32x [in]

start x position.

u32y [in]

start y position.

pszString [in]

The specified string for writing to frame buffer.

Return Value

None

Example

```

/* Draw a string to the position (0, 0) of frame buffer */
__align(32) static S_DEMO_FONT s_sDemo_Font
char szString[64];
sprintf(szString, "N9H26 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 [out]

Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

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 [in]

Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

ptRect [in]

Solid Rectangle pointer Reference Table 10-2 Rectangle Information Structure Definition.

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 [in]

Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

ptRect [in]

Solid Rectangle pointer. Reference Table 10-2 Rectangle Information Structure Definition.

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 [in]

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 [in]
Font library information pointer. Reference Table 10-1 Font Information Structure Definition.
- ptRect [in]
Solid rectangle pointer. Reference Table 10-2 Rectangle Information Structure Definition.
- u32Width [in]
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 [in]
Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

u16RGB565 [in]
RGB565n format

Return Value

None

Example

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

DemoFont_GetFontColor

Synopsis

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

Description

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

Parameter

ptFont [in]

Font library information pointer. Reference Table 10-1 Font Information Structure Definition.

Return Value

RGB565 format

Example

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

11 GNAND Library

11.1 GNAND Library Overview

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

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

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

11.1.1 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 upper layer FAT file system library or USB mass storage device driver.

The relationship between these component libraries was shown in the following picture:

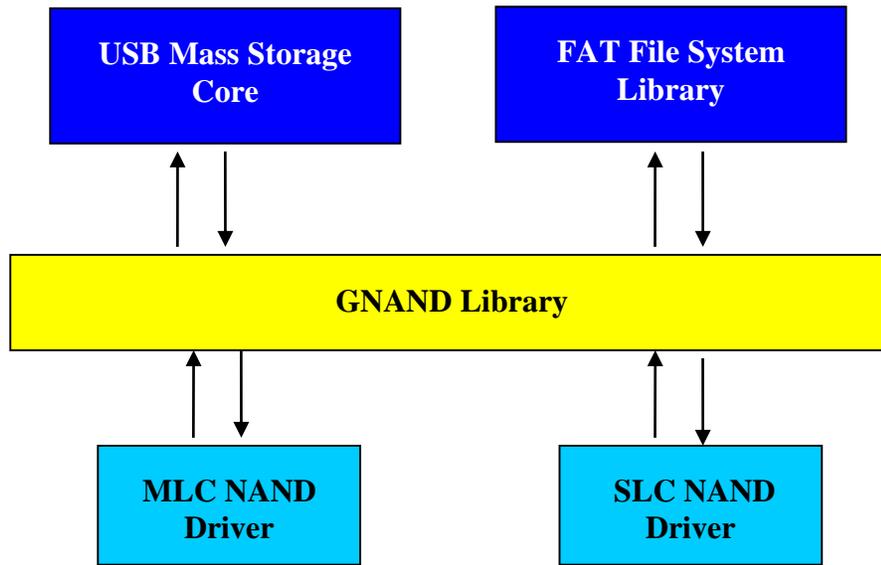


Figure 11-1 GNAND System Overview

Initialize GNAND Library

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

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

GNAND work with Nuvoton FAT Library

If `GNAND_InitNAND()` returns `GNAND_OK`, application can invoke `GNAND_MountNandDisk()` to mount NAND disk to NVTFAT file system.

NAND driver function set

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

```

#define NDRV_T struct ndrvt_t
struct ndrvt_t
{
    INT (*init)(NDISK_T *NDInfo);
    INT (*pread)(INT nPBlockAddr, INT nPageNo, UINT8 *buff);
}
  
```

```

INT (*pwrite)(INT nPBlockAddr, INT nPageNo, UINT8 *buff);
INT (*is_page_dirty)(INT nPBlockAddr, INT nPageNo);
INT (*is_valid_block)(INT nPBlockAddr);
INT (*ioctl)(INT param1, INT param2, INT param3, INT param4);
INT (*block_erase)(INT nPBlockAddr);
INT (*chip_erase)(VOID);
VOID *next;
};
    
```

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

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.

11.2 Definition

11.2.1 Structure

NDISKT_T Structure

Field Name	Data Type	Return by init()	Description
vendor_ID	INT	Optional	
device_ID	INT	Optional	
NAND_type	INT	Must	NAND_TYPE_SLC or NAND_TYPE_MLC

Field Name	Data Type	Return by init()	Description
nZone	INT	Must	Number of zones
nBlockPerZone	INT	Must	Maximum number of physical blocks per zone
nPagePerBlock	INT	Must	Number of pages per block
nLBPPerZone	INT	Must	Maximum number of allowed logical blocks per zone
nPageSize	INT	Must	Page size in bytes
nStartBlock	INT	Must	Reserved number of leading blocks
nBadBlockCount	INT	Optional	Bad block count for all zones
driver	NDRV_T *	Must	NAND driver function set pointer
nNandNo	INT	Optional	
pDisk	VOID *	Optional	
write_page_in_seq	INT	Must	Program pages within the block in sequence or not
reserved[59]	INT	Ignore	
need2L2PN	INT	Optional	Need second P2LN block or not
p2ln_block1	INT	Optional	Physical block address for second P2LN block
p2lm	P2LM_T *	Ignore	GNAND internal used
l2pm	L2PM_T *	Ignore	GNAND internal used
dp_tbl	UINT8 *	Ignore	GNAND internal used
db_idx[16]	UINT16	Ignore	GNAND internal used
p2ln_block	UINT16	Ignore	GNAND internal used
op_block	UINT16	Ignore	GNAND internal used
op_offset	INT	Ignore	GNAND internal used
last_op[32]	UINT8	Ignore	GNAND internal used
err_sts	INT	Ignore	GNAND internal used
next	struct NDRV_T *	Ignore	GNAND internal used

Table 11-1 NDISKT_T Structure Definition

11.3 API function

GNAND_InitNAND

Synopsis

INT

```
GNAND_InitNAND (
    NDRV_T *ndriver,
    NDISKT_T *ptNDisk,
    BOOL bEraseIfNotGnandFormat
```

)

Description

Initialize a NAND disk.

Parameter

ndriver [in]

NAND driver function set to hook NAND driver on GNAND library.

ptNDisk [out]

NAND disk information that GNAND initiated. You need this pointer to call other GNAND APIs.

bEraseIfNotGnandFormat [in]

If NAND disk was GNAND format, ignore this argument.

If NAND disk was not GNAND format, format it if this argument is 1, otherwise, return an GNERR_GNAND_FORMAT error.

Return Value

0 Success

Otherwise Error code defined in Table 11-2 GNAND Error Code Table.

Example

```
NDRV_T _nandDiskDriver0 =
{
nandInit0,
nandpread0,
nandpwrite0,
nand_is_page_dirty0,
nand_is_valid_block0,
nand_ioctl,
nand_block_erase0,
nand_chip_erase0,
0
};
NDISK_T *ptNDisk;
int status;

fsInitFileSystem();

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

```

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

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

status = GNAND_MountNandDisk(ptNDisk);
if (status < 0)
{
    printf("Mount NAND disk failed, status = %x\n", status);
    return status;
}
    
```

GNAND_MountNandDisk

Synopsis

```

INT
GNAND_MountNandDisk (
    NDISK_T *ptNDisk
)
    
```

Description

Mount NAND disk to NVTFAT file system.

Parameter

ptNDisk [in]

The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().

Return Value

0	Success
Otherwise	Error code defined in Table 11-2 GNAND 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 [in]

The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().
- nSectorNo [in]

Read start sector number.
- nSectorCnt [in]

Number of sectors to be read.
- buff [out]

Memory buffer to receive data, which is 32 bytes aligned non-cacheable buffer.

Return Value

- 0 Success
- Otherwise Error code defined in Table 11-2 GNAND Error Code Table.

Example

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

GNAND_write

Synopsis

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

Description

Write logical sectors to NAND disk

Parameter

- ptNDisk [in]

The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().
- nSectorNo [in]

Write start sector number.
- nSectorCnt [in]

Number of sectors to be written.
- buff [in]

Memory buffer to write data, which is 32 bytes aligned non-cacheable buffer

Return Value

- 0 Success
- Otherwise Error code defined in Table 11-2 GNAND Error Code Table.

Example

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

GNAND_block_erase

Synopsis

```
INT
```

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

Description

Erase a physical bock.

Parameter

ptNDisk [in]

The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().

pba [in]

NAND physical block address.

Return Value

0 Success

Otherwise Error code defined in Table 11-2 GNAND Error Code Table.

Example

```
NDISK_T *ptNDisk;
int status;

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

GNAND_chip_erase

Synopsis

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

Description

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

Parameter

ptNDisk [in]

The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().

Return Value

0 Success

Otherwise Error code defined in Table 11-2 GNAND Error Code Table.

Example

```

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

GNAND_UnMountNandDisk

Synopsis

```

VOID GNAND_UnMountNandDisk (
    NDISK_T *ptNDisk
)
    
```

Description

Unmount NAND disk from NVT FAT file system.

Parameter

ptNDisk [in]

The pointer refer to the NAND disk information that initiated by GNAND_InitNAND().

Return Value

0 Success

Otherwise Error code defined in Table 11-2 GNAND Error Code Table.

Example

```

NDISK_T *ptNDisk;
int status;
status = GNAND_UnMountNandDisk(ptNDisk);
if (status != 0)
    
```

```
{
  /* handle error status */
}
```

11.4 GNAND Error Code Table

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

Table 11-2 GNAND Error Code Table

12 GPIO Library

12.1 GPIO Library Overview

This library is designed to control GPIO.

12.2 API function

gpio_open

Synopsis

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

Description

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

Parameter

```
port [in]
    GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD, GP
    O_PORTE, GPIO_PORTG, and GPIO_PORTH
```

Return Value

```
0      Success
```

Example

```
None
```

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 [in]

GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD,
GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

num [in]

pin number

Return Value

0 Success

-1 Unknown port number

Example

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

gpio_readport

Synopsis

int

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

Description

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

Parameter

port [in]

GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD, GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

val [out]

Return port value

Return Value

0 Success,

-1 Unknown port number

Example

```
/* Read PORTC value*/
unsigned short val;
```

```
gpio_readport(GPIO_PORTC, &val);
```

gpio_setportdir

Synopsis

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

Description

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

Parameter

```
port [in]
    GPIO_PORTA,    GPIO_PORTB,    GPIO_PORTC,    GPIO_PORTD,
    GPIO_PORTE,   GPIO_PORTG, and GPIO_PORTH
mask [in]
    pin mask, each bit stands for one pin
dir [in]
    Direction, each bit configures one pin, 0 means input, 1 means output
```

Return Value

```
0      Success,
-1     Unknown port number
```

Example

```
/* Set PORTC pin1 to output mode, and pin0 to input mode */
gpio_setportdir (GPIO_PORTC, 0x3, 0x2);
```

gpio_setportval

Synopsis

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

)

Description

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

Parameter

port [in]

GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD,
GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

mask [in]

pin mask, each bit stands for one pin

val [in]

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

Return Value

- 0 Success,
- 1 Unknown port number

Example

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

gpio_setportpull

Synopsis

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

Description

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

Parameter

port [in]

GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD,
GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

mask [in]

pin mask, each bit stands for one pin

pull [in]

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

Return Value

- 0 Success,
- 1 Unknown port number

Example

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

gpio_setdebounce

Synopsis

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

Description

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

Parameter

- clk [in]
 - 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 [in]
 - Debounce sampling interrupt source. Valid values are between 0~15. Each bit represents one interrupt source

Return Value

- 0 Success
- 1 Parameter error

Example

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

gpio_getdebounce

Synopsis

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

Description

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

Parameter

```
clk [out]
    Debounce sampling clock

src [out]
    Debounce sampling interrupt source
```

Return Value

None

Example

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

gpio_setsrcgrp

Synopsis

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

Description

This function is used to set external interrupt source group.

Parameter

```
port [in]
    GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD,
    GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH
```

mask [in]
 pin mask, each bit stands for one pin

irq [in]
 external irq number. Could be 0~3

Return Value

0 Success
 -1 Parameter error

Example

```
/* Set GPIO port C pin0 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 [in]
 GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD,
 GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

val [out]
 Current source setting. Every two bits stands for the interrupt source each pin triggers

Return Value

0 Success
 -1 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 [in]
    GPIO_PORTA,    GPIO_PORTB,    GPIO_PORTC,    GPIO_PORTD,
    GPIO_PORTE,   GPIO_PORTG, and GPIO_PORTH
mask [in]
    Pin mask, each bit stands for one pin
falling [in]
    Triggers on falling edge, each bit stands for one pin
rising [in]
    Triggers on rising edge, each bit stands for one pin
```

Return Value

```
0      Success
-1     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 pin]

```
    GPIO_PORTA,    GPIO_PORTB,    GPIO_PORTC,    GPIO_PORTD,
    GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH
```

falling [out]

Triggers on falling edge, each bit stands for one pin

rising [out]

Triggers on rising edge, each bit stands for one pin

Return Value

```
    0      Success
    -1     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 [in]

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

0 Success
 -1 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 [out]
 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 [in]
 GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTD,

GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

val [out]

Variable to store latch value

Return Value

- 0 Success
- 1 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 [in]
 GPIO_PORTA, GPIO_PORTB, GPIO_PORTC, GPIO_PORTH,
 GPIO_PORTE, GPIO_PORTG, and GPIO_PORTH

src [out]
 Variable to store trigger source

Return Value

- 0 Success
- 1 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 [in]
    GPIO_PORTA,    GPIO_PORTB,    GPIO_PORTC,    GPIO_PORTD,
    GPIO_PORTE,   GPIO_PORTG, and GPIO_PORTH
```

Return Value

```
0      Success
-1     Parameter error
```

Example

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

13 H264 Codec Library

13.1 H264 Codec Library Overview

Encoder Features

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

Decoder Features.

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

13.1.1 Rate Control for Encoder

The encoded bitstream rate control is implemented in application level. At the beginning, the application needs to call `H264RateControlInit(...)` function to initialize the variable in `H264RateControl` structure. After that, `H264RateControlUpdate(...)` function is used to calculate the next Quant value for next encode frame which is stored in the `rtn_quant` field of `H264RateControl` structure.

Specify this `rtn_quant` in `u32Quant` field of `FAVC_ENC_PARAM` structure for next encode frame. Then, the rate control will be completed in such kind of loop.

13.2 Definition

13.2.1 Constant

IOCTL COMMAND

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

Table 13-1 IOCTL COMMAND Definition

13.2.2 Structure

AVC_DEC_RESULT

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

Table 13-2 H264 AVC_DEC_RESULT Definition

FAVC_DEC_PARAM

Field name	Data Type	Description
u32API_version	UINT32	API version
u32MaxWidth	UINT32	Not used now
u32MaxHeight	UINT32	Decoded bitstream height
u32FrameBufferWidth	UINT32	if (u32FrameBufferWidth != -1), decoded image width is cropped with u32FrameBufferWidth if (u32FrameBufferWidth == -1), decoded image width is continued on memory
u32FrameBufferHeight	UINT32	if (u32FrameBufferHeight != -1), decoded

Field name	Data Type	Description
		image height is cropped with u32FrameBufferHeight if (u32FrameBufferHeight == -1), decoded image height is continued on memory
u32Pkt_size	UINT32	Current decoding bitstream length (the exact bitstream length for one frame)
pu8Pkt_buf	UINT8*	Current decoding bitstream buffer address (application ready bitstream here)
pu8Display_addr[3]	UINT32	Buffer address for decoded data
got_picture	UINT32	0 -> Decoding has something error. 1 -> decoding is OK in current bitstream
pu8BitStream_phy	UINT8*	physical address. buffer for bitstream (allocated and used by library only)
u32OutputFmt	UINT32	Decoded output format, 0-> Planar YUV420 format, 1-> Packet YUV422 format
crop_x	UINT32	pixel unit: crop x start point at decoded-frame (not supported now)
crop_y	UINT32	pixel unit: crop y start point at decoded-frame (not supported now)
tResult	FAVC_DEC_RESULT	Return decoding result by library

Table 13-3 FAVC_DEC_PARAM Definition

FAVC_ENC_PARAM

Field name	Data Type	Description
u32API_version	UINT32	API version
u32BitRate	UINT32	The encoded bitrate in bps.
u32FrameWidth	UINT32	The width of encoded frame in pels.
u32FrameHeight	UINT32	The height of encoded frame in pels
fFrameRate	UINT32	The base frame rate per second
u32IPInterval	UINT32	The frame interval between I-frames.
u32MaxQuant	UINT32	The maximum quantization value. (max = 51)
u32MinQuant	UINT32	The minimum quantization value. (min=0)
u32Quant	UINT32	The frame quantization value for initialization
ssp_output	INT32	This variable tells the H.264 must be encoded out sps + pps before slice data. --> 1 : force the encoder to output sps+pps -> 0 : force the encoder to output sps+pps on any Slice I frame

Field name	Data Type	Description
		-> -1: (default) only output SPS+PPS on first IDR frame.
intra	INT32	This variable tells the H.264 must be encoded out an I-Slice type frame. -> 1 : forces the encoder to create a keyframe. -> 0 : forces the encoder not to create a keyframe. -> -1: (default) let the encoder decide (based on contents and u32IPInterval)
bROIEnable	INT32	To enable the function of encoding rectangular region of interest(ROI) within captured frame
u32ROIX	UINT32	The upper-left corner x coordinate of rectangular region of interest
u32ROIY	UINT32	The upper-left corner coordinate y of region of interest
u32ROIWidth	UINT32	The width of user-defined rectangular region of interest
u32ROIHeight	UINT32	The height of user-defined rectangular region of interest
pu8YFrameBaseAddr	UINT8*	The base address for input Y frame buffer
pu8UVFrameBaseAddr	UINT8*	The base address for input UV frame buffer in H.264 2D mode
pu8UFrameBaseAddr	UINT8*	The base address for input U frame buffer
pu8VFrameBaseAddr	UINT8*	The base address for input V frame buffer
bitstream	void*	Bitstream Buffer address for driver to write bitstream
pu8BitstreamAddr	UINT8*	The bitstream buffer address while encoding one single frame allocated by librar
bitstream_size	UINT32	Bitstream length for current frame
keyframe	INT32	This parameter is indicated the Slice type of frame
frame_cost	INT32	frame_cout is updated by driver
no_frames	UINT32	The number of frames to be encoded
threshold_disable	UINT32	The transform coefficients threshold
chroma_threshold	UINT32	The chroma coefficients threshold (0 ~ 7)
luma_threshold	UINT32	The luma coefficients threshold (0 ~ 7)
beta_offset	UINT32	The beta offset for in-loop filter.
alpha_offset	UINT32	The alpha offset for in-loop filter.
chroma_qp_offset	UINT32	The chroma qp offset (-12 to 12 inclusively)
disable_ilf	UINT32	To disable in-loop filter or not
watermark_enable	UINT32	To enable watermark function or not (Don't enable it now)

Field name	Data Type	Description
watermark_interval	UINT32	To specify the watermark interval if watermark function is enabled
watermark_init_pattern	UINT32	To specify the initial watermark pattern if watermark function is enabled
pu8ReConstructFrame	UINT8*	The address of reconstruct frame buffer.
pu8ReferenceFrame	UINT8*	The address of reference frame buffer
pu8SysInfoBuffer	UINT8*	The address of system info buffer
pu8DMABuffer_phy	UINT8*	The physical address of DMA buffer
nvop_ioctl	INT32	This parameter is valid only on FAVC_IOCTL_ENCODE_NVOP
multi_slice	UINT32	Multi-slice mode
pic_height	UINT32	This parameter is used to keep the frame height for sps and pps on Multi Slice mode
pic_width	UINT32	This parameter is used to keep the frame width for sps and pps on Multi Slice mode
img_fmt	UINT32	0: 2D format, CbCr interleave, named H264_2D (VideoIn supported only)
control	UINT32	0 : Do NOT force one frame as one slice(default), 1 : Force one frame as one slice

Table 13-4 FAVC_ENC_PARAM Definition

13.3 API function

H264Dec_Open

Synopsis

```
int
H264Dec_open(void)
```

Description

Initialize H264 Decoder and install interrupt service routine.

Parameter

None

Return Value

```
>1    Success
-1    Fail
```

Example

None

H264Enc_Open

Synopsis

int

H264Enc_Open (void)

Description

Initialize H264 Encoder and install interrupt service routine.

Parameter

None

Return Value

>1 Success

-1 Fail

Example

None

H264_ioctl

Synopsis

Int

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

Description

Perform the H264 encoder/decoder related operation

Parameter

cmd [in]

Specify the operation for Encoder or Decoder which is defined in ioctl command set.

param [in/out]

The pointer to FAVC_ENC_PARAM or FAVC_DEC_PARAM dependent on cmd

Return Value

0 Success

-1 Fail

Example

None

H264Enc_Close**Synopsis**

```
void H264Enc_Close (void)
```

Description

Close H264 Encoder and free related buffer allocation.

Parameter

None

Return Value

None

Example

None

H264Dec_Close**Synopsis**

```
void H264Dec_Close (void)
```

Description

Close H264 Decoder and free related buffer allocation.

Parameter

None

Return Value

None

Example

None

nv_malloc**Synopsis**

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

Description

Allocate memory in size which is alignment.

Parameter

size [in]

Specify the allocated memory size

alignment [in]

specify the allocated memory alignment

Return Value

Pointer to the allocated memory

Example

None

nv_free**Synopsis**

Int

```
nv_free (  
    void* ptr  
)
```

Description

Free the memory specified by ptr.

Parameter

ptr [in]

pointer to memory which is to free.

Return Value

0 Success

Example

None

H264_ioctl_ex**Synopsis**

int

```
H264_ioctl_ex (  
    int handle,  
    int cmd,  
    void *param  
)
```

Description

Perform the H264 encoder/decoder related operation specified by handle instance.

Parameter

handle [in]

Specify instance handle for this operation

cmd [in]

Specify the operation for Encoder or Decoder which is defined in ioctl command set.

param [in/out]

The pointer to FAVC_ENC_PARAM or FAVC_DEC_PARAM dependent on cmd

Return Value

0 Success

-1 Fail

Example

None

H264Enc_Close_ex

Synopsis

```
void H264Enc_Close_ex (
    int handle
)
```

Description

Close H264 Encoder and free related buffer allocation specified by handle instance.

Parameter

handle [in]

Specify instance handle to close H264 Encoder.

Return Value

None

Example

None

H264Dec_Close_ex

Synopsis

```
void H264Dec_Close_ex (
    int handle
)
```

Description

Close H264 Decoder and free related buffer allocation specified by handle instance

Parameter

handle [in]
Specify instance handle to close H264 Decoder.

Return Value

None

Example

None

13.4 H264 Error Code Table

Code Name	Value	Description
RETCODE_OK	0	
RETCODE_ERR_MEMORY	1	
RETCODE_ERR_API	2	
RETCODE_ERR_HEADER	3	
RETCODE_ERR_FILL_BUFFER	4	
RETCODE_ERR_FILE_OPEN	5	
RETCODE_HEADER_READY	6	
RETCODE_BS_EMPTY	7	
RETCODE_WAITING	8	
RETCODE_DEC_OVERFLOW	9	
RETCODE_HEADER_FINISH	10	
RETCODE_DEC_TIMEOUT	11	
RETCODE_PARSING_TIMEOUT	12	
RETCODE_ERR_GENERAL	13	
RETCODE_NOT_SUPPORT	14	Frame cropping Not support
RETCODE_FAILURE	15	

Code Name	Value	Description
RETCODE_FRAME_NOT_COMPLETE	16	

Table 13-5 H264 Return Code Table

14 I²C Library

14.1 I²C Library Overview

This library provides APIs for programmers to access I²C slaves connecting with N9H26 I²C interfaces. The default clock frequency is configured at 100 kHz after i2cOpen() is called, programmers could use i2cloctl() 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.

14.2 Definition

14.2.1 Constant

IOCTL COMMAND

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

Table 14-1 I²C IOCTL Command Definition

14.3 API function

i2cInit

Synopsis

INT32

i2cInit (void)

Description

This function configures GPIO to I²C mode.

Parameter

None

Return Value

0 Always successes

Example

```
i2cInIt();
```

i2cOpen

Synopsis

INT32

i2cOpen(void)

Description

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

Parameter

None

Return Value

0 Successful
 I2C_ERR_BUSY Interface already opened

Example

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

i2cClose

Synopsis

INT32

i2cClose (void)

Description

This function disables I²C engine clock.

Parameter

None

Return Value

0 Successful

Example

```
i2cClose();
```

i2cRead

Synopsis

```
INT32
i2cRead (
    PUINT8 buf,
    UINT32 len
)
```

Description

This function reads data from I²C slave.

Parameter

buf [in]
Receive buffer pointer

len [in]
Receive buffer length

Return Value

> 0	Return read length on success
I2C_ERR_NOERROR	No error
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

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.

Parameter

- cmd [in]
Command
- arg0 [in]
First argument of the command
- arg1 [in]
Second argument of the command

Return Value

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

Example

```
/* Set clock frequency to 400 kHz */
i2cloctl(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();
```

14.4 I²C Error Code Table

Code Name	Value	Description
I2C_ERR_ID	0xFFFF1100	Device error
I2C_ERR_NOERROR	0x00	No error
I2C_ERR_LOSTARBITRATION	0xFFFF1101	Arbitration lost during transmission
I2C_ERR_BUSBUSY	0xFFFF1102	Interface 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

Table 14-2 I²C Error Code Table

15 I²S Library

15.1 API function

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();
```

DrvI2S_EnableInt

Synopsis

```
VOID DrvI2S_EnableInt (
    UINT32 u32InterruptFlag,
    PFN_DRVI2S_CB_FUNC *pfnCallBack
)
```

Description

Enable I²S selected interrupt source and setup its corresponding interrupt service routine.

Parameter

u32InterruptFlag [in]

Select enabled interrupt source.

pfnCallBack [in]

Setup corresponding interrupt service routine.

Return Value

None

Example

```
DrvI2S_EnableInt(DRVI2S_IRQ_PLAYBACK, (PFN_DRVI2S_CB_FUNC*)
&pfnPlaybackCallBack);
```

DrvI2S_DisableInt

Synopsis

```
VOID DrvI2S_DisableInt (
    UINT32 u32InterruptFlag
)
```

Description

Disable I²S selected interrupt source.

Parameter

u32InterruptFlag [in]

Select disabled interrupt source.

Return Value

None

Example

```
DrvI2S_DisableInt(DRVI2S_IRQ_RECORD);
```

DrvI2S_ClearInt

Synopsis

```
VOID DrvI2S_ClearInt (
    UINT32 u32InterruptFlag
)
```

Description

Clear Interrupt I²S selected interrupt flag.

Parameter

u32InterruptFlag [in]

Select cleared interrupt flag.

Return Value

None

Example

```
DrvI2S_ClearInt(DRVI2S_IRQ_RECORD);
```

DrvI2S_PollInt**Synopsis**

UINT32

DrvI2S_PollInt (

UINT32 u32InterruptFlag

)

Description

Check if I²S selected interrupt flag to be set.

Parameter

u32InterruptFlag [in]

Select polling interrupt flag.

Return Value

Interrupt Flag Being set interrupt flag.

Example

```
DrvI2S_PollInt(DRVI2S_IRQ_PLAYBACK);
```

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 [in]

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 [in]

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

INT

```
DrvI2S_SetSampleRate (
    E_DRVI2S_SAMPLING eSampleRate
)
```

Description

Set Play/Record sampling rate.

Parameter

eSampleRate [in]

Given sampling rate.

Return Value

0 Success

-1 Fail

Example

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

16 JPEG Library

16.1 JPEG Library Overview

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

The JPEG library has the following features:

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

16.1.1 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.

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

JPEG Operation Control

- Memory access

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

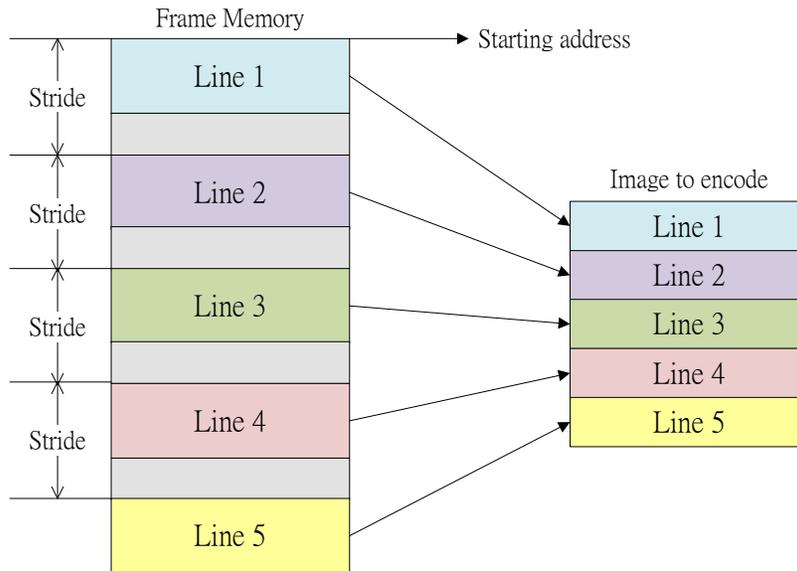


Figure 16-1 Stride function for Encode

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

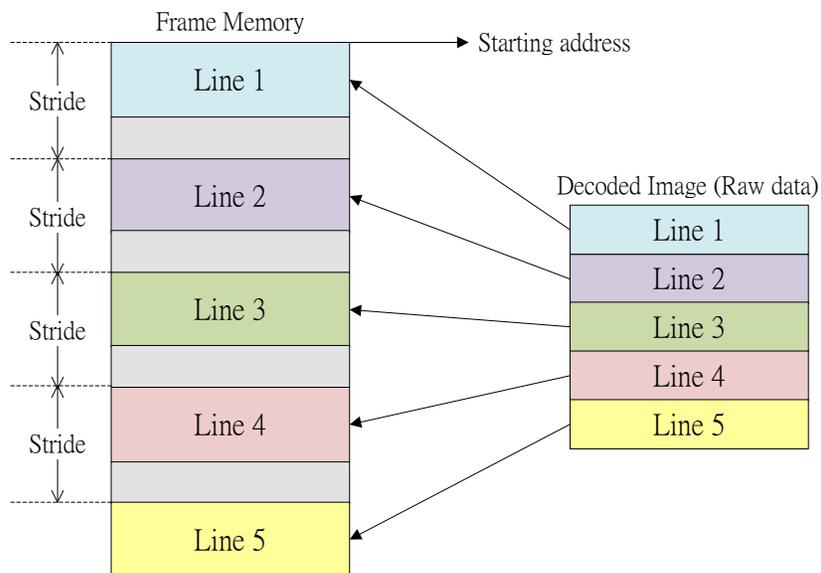


Figure 16-2 Stride function for Decode

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.

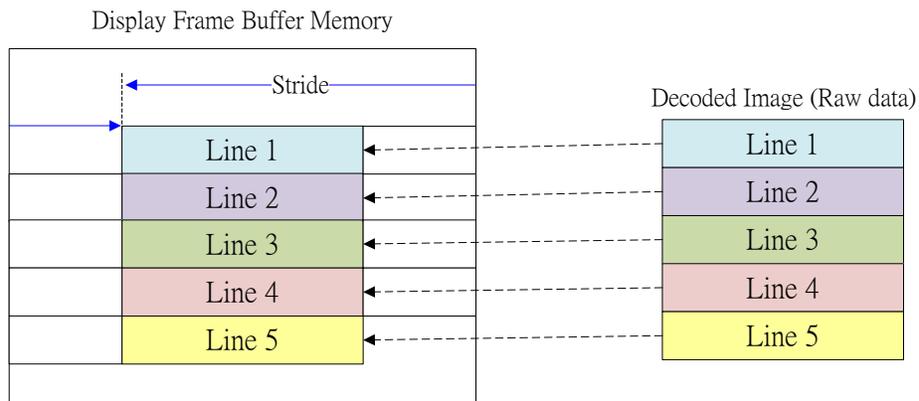


Figure 16-3 Stride function for Display

● Encode operation flow

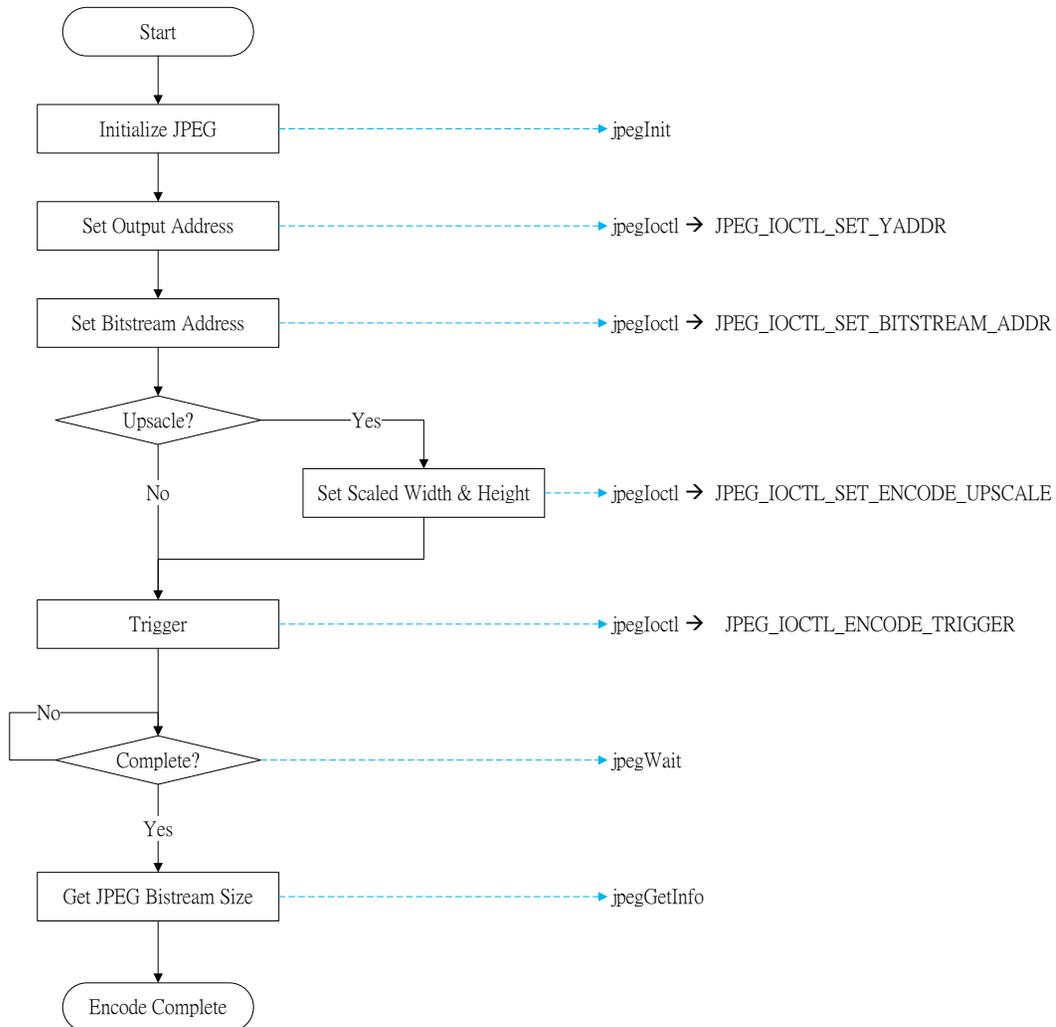


Figure 16-4 Encode operation flow

● Decode operation flow

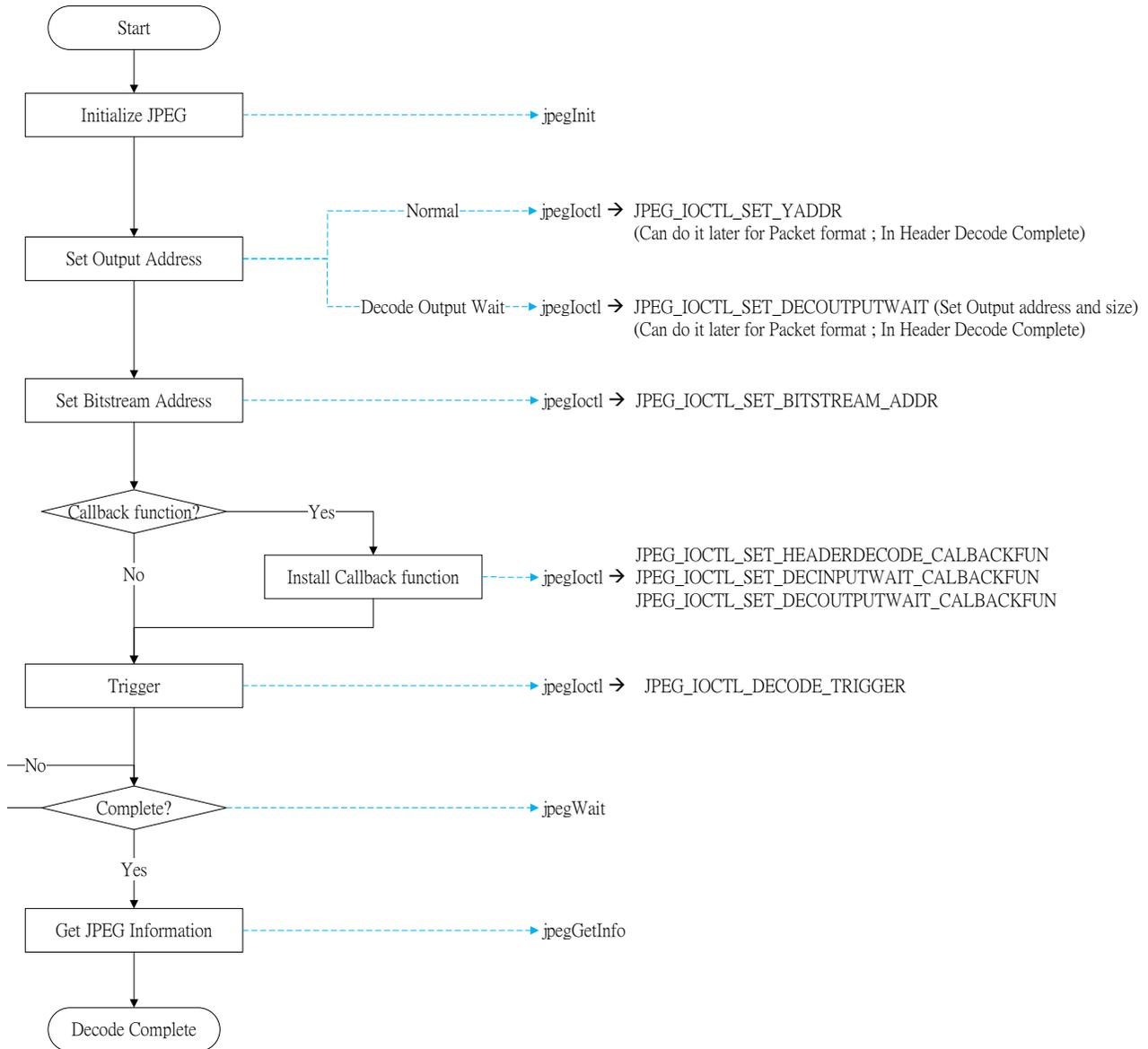


Figure 16-5 Decode operation flow

● Decode stride

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

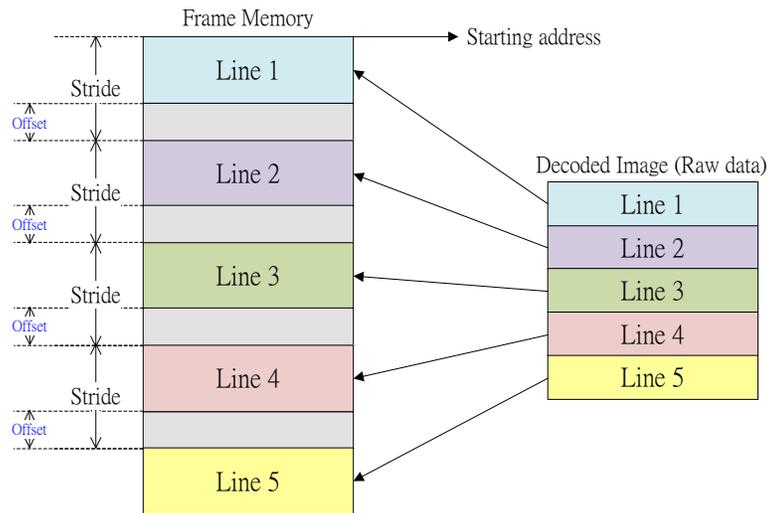


Figure 16-6 Offset for Decode

● 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.

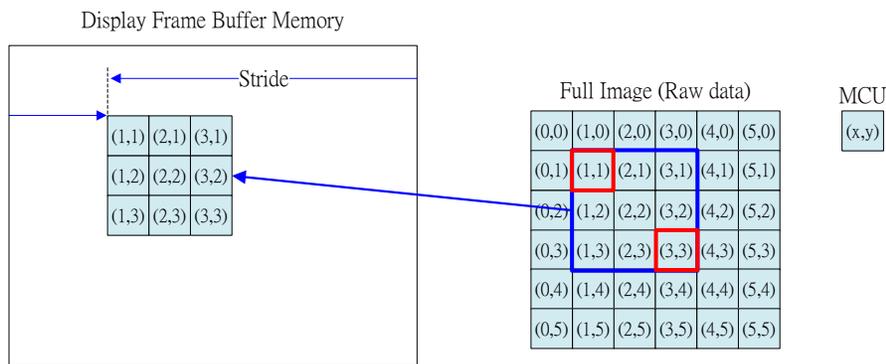


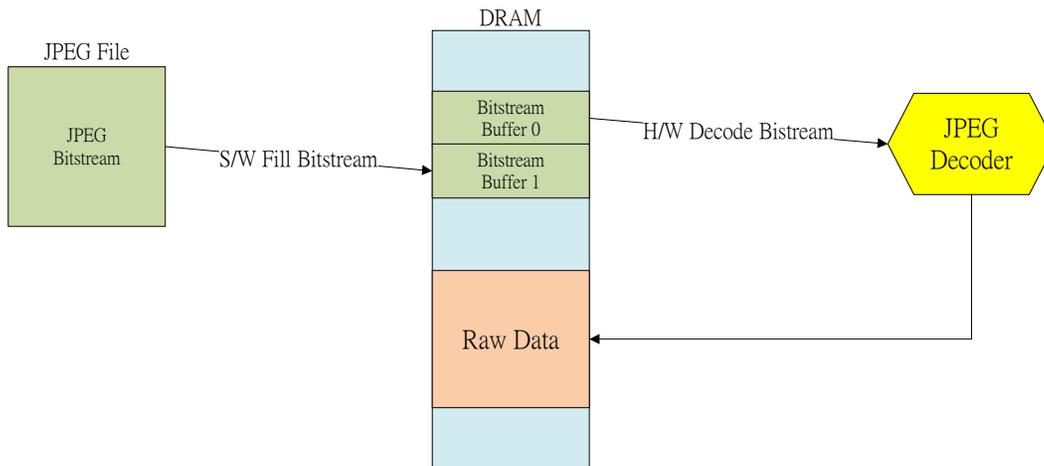
Figure 16-7 Window Decode

● Decode Input Wait

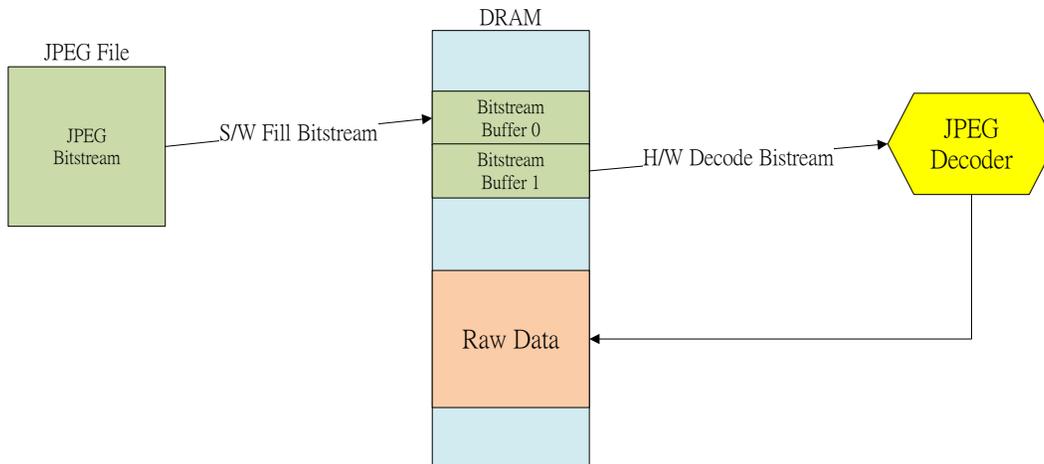
When the JPEG is in decoding mode, the input source is the JPEG bit-stream written by software. The bit-stream buffer size is in 2 K unit dual-buffer manner. If the buffer-size is 2 KB, user needs to fill 1 KB bit-stream into one of the half buffer region before resuming JPEG operation when an input-wait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

When JPEG engine decodes one of the half buffers, the Decode Input Wait call back function will be called. The Only thing user needs to do is to fill bit stream to the other buffer like the following Step 1 & Step 2 into the buffer.

1. JPEG engine decodes the data in Buffer 0 and S/W fills the data into Buffer 1.



2. JPEG engine decodes the data in Buffer1 and S/W fills the data into Buffer 0.



- Decode Input Wait

When there is not enough continuous space to store the decode output raw data, JPEG engine support a function to output data partially. User can get the whole data by assigning several output data address and size settings (JPEG_IOCTL_SET_DECOUPTWAIT). Using this function, user can get the JPEG decoded image that larger than the available continuous memory space.

The decode output wait call back function will be called when the Output Data size is equal to the Size assigned by IOCTL - JPEG_IOCTL_SET_DECOUPTWAIT. In the call back function, user should move the data to the destination address and call the IOCTL again to set next address and data size.

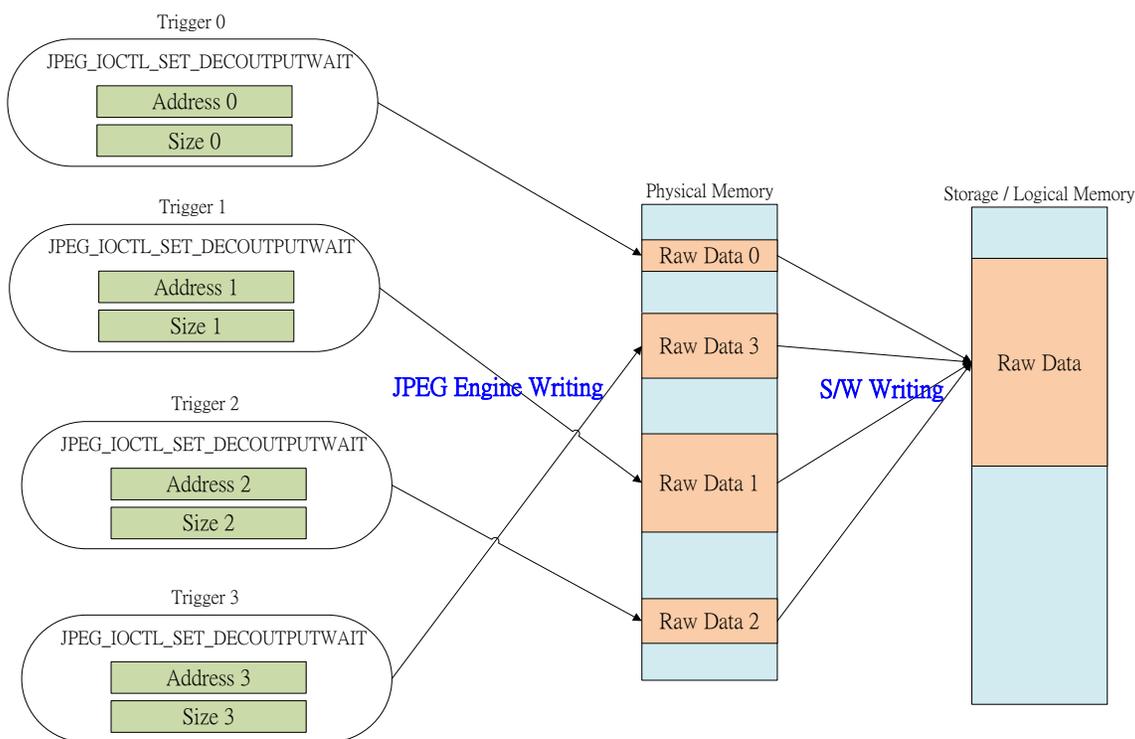


Figure 16-8 Decode Input Wait

The data size of the final ICOTL must equal to the extract output size. Otherwise, user will get the Decode complete interrupt instead of Decode output wait interrupt.

- Header Decode Complete

In the callback function, user can get JPEG image width and height by calling jpegGetInfo(). After getting the information, user can use jpegIoctl to

- Allocate and set output buffer JPEG_IOCTL_SET_YADDR for Packet Format Only
- Change output buffer address JPEG_IOCTL_SET_YADDR for Packet Format Only
- Set Downscale JPEG_IOCTL_SET_DECODE_DOWNSCALE
- Set Decode Output Stride JPEG_IOCTL_SET_DECODE_STRIDE
- Set windows decode JPEG_IOCTL_SET_WINDOW_DECODE

16.2 Definition

16.2.1 Constant

Encode operation

Name	Value	Description
Encode format		
JPEG_ENC_PRIMARY	0	Encode operation : Primary JPEG

Name	Value	Description
JPEG_ENC_THUMBNAIL	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_THUMBNAIL_YUV420	0x90	Thumbnail Encode image format : YUV 4:2:0
JPEG_ENC_THUMBNAIL_YUV422	0x98	Thumbnail Encode image format : YUV 4:2:2
JPEG_ENC_THUMBNAIL_GRAY	0x91	Thumbnail Encode image format : GRAY
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_THUMBNAIL_DRI	0x1	Restart Interval in Thumbnail JPEG Header
JPEG_ENC_THUMBNAIL_QTAB	0x2	Quantization-Table in Thumbnail JPEG Header
JPEG_ENC_THUMBNAIL_HTAB	0x4	Huffman-Table in Thumbnail JPEG Header
JPEG_ENC_THUMBNAIL_JFIF	0x8	JFIF Header in Thumbnail JPEG Header
Interrupt		
JPEG_EER_INTS	0x01	Encode Error Interrupt Status
JPEG_DER_INTS	0x02	Decode Error Interrupt Status
JPEG_DEC_INTS	0x04	Decode Complete Interrupt Status
JPEG_ENC_INTS	0x08	Encode Complete Interrupt Status
JPEG_IPW_INTS	0x20	Input Wait Interrupt Status
JPEG_DHE_INTS	0x40	JPEG Header Decode End Wait Interrupt Status

Table 16-1 Encode Operation Definition

Decode Operation

Name	Value	Description
Decode output format		
JPEG_DEC_PRIMARY_PLANAR_YUV	0x8021	Primary Decode output format : planar format
JPEG_DEC_PRIMARY_PACKET_YUV422	0x0021	Primary Decode output format : planar YUV422
JPEG_DEC_PRIMARY_PACKET_RGB555	0x004021	Primary Decode output format : packet RGB555
JPEG_DEC_PRIMARY_PACKET_RGB555R1	0x404021	Primary Decode output format : packet RGB555R1

Name	Value	Description
JPEG_DEC_PRIMARY_PACKET_RGB555R2	0x804021	Primary Decode output format : packet RGB555R2
JPEG_DEC_PRIMARY_PACKET_RGB565	0x006021	Primary Decode output format : packet RGB565
JPEG_DEC_PRIMARY_PACKET_RGB565R1	0x406021	Primary Decode output format : packet RGB565R1
JPEG_DEC_PRIMARY_PACKET_RGB565R2	0x806021	Primary Decode output format : packet RGB565R2
JPEG_DEC_PRIMARY_PACKET_RGB888	0x14021	Primary Decode output format : packet RGB888
JPEG_DEC_THUMBNAI_PLANAR_YUV	0x8031	Thumbnail Decode output format : planar YUV
JPEG_DEC_THUMBNAI_PACKET_YUV422	0x0031	Thumbnail Decode output format : packet RGB555
JPEG_DEC_THUMBNAI_PACKET_RGB555	0x4031	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

Table 16-2 Encode Operation Definition

IOCTL

Command	Argument 0	Argument 1	comment
JPEG_IOCTL_SET_YADDR	JPEG Y component frame buffer address		Specify the JPEG Y component frame buffer address.
JPEG_IOCTL_SET_YSTRIDE	JPEG Y component frame buffer stride		Specify the JPEG Y component frame buffer stride
JPEG_IOCTL_SET_USTRIDE	JPEG U component frame buffer stride		Specify the JPEG U component frame buffer stride
JPEG_IOCTL_SET_VSTRIDE	JPEG V component frame buffer stride		Specify the JPEG V component frame buffer stride

Command	Argument 0	Argument 1	comment
JPEG_IOCTL_SET_BITSTREAM_ADDR	JPEG bit stream buffer starting address		Specify the bit stream frame buffer starting address
JPEG_IOCTL_SET_SOURCE_IMAGE_HEIGHT	The encode source image height in pixel		Specify the encode source image height in pixel
JPEG_IOCTL_ENC_SET_HEADER_CONTROL	JPEG_ENC_PRIMARY_DRI JPEG_ENC_PRIMARY_QTAB JPEG_ENC_PRIMARY_HEADER 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_PACKET_YUV JPEG_DEC_PRIMARY_PACKET_YUV422 JPEG_DEC_PRIMARY_PACKET_RGB555 JPEG_DEC_PRIMARY_PACKET_RGB555R1 JPEG_DEC_PRIMARY_PACKET_RGB555R2 JPEG_DEC_PRIMARY_PACKET_RGB565 JPEG_DEC_PRIMARY_PACKET_RGB565R1 JPEG_DEC_PRIMARY_PACKET_RGB565R2 JPEG_DEC_PRIMARY_PACKET_RGB888		Specify the decoded image output format
JPEG_IOCTL_SET_ENCODE_MODE	JPEG_ENC_SOURCE_PACKET JPEG_ENC_SOURCE_PACKET	JPEG_ENC_PRIMARY_YUV420 JPEG_ENC_PRIMARY_YUV422	Specify the encode source format and encoding image format

Command	Argument 0	Argument 1	comment
JPEG_IOCTL_SET_DIMENSION	Image height	Image width	Set the encode image dimension or decode output image dimension
JPEG_IOCTL_ENCODE_TRIGGER			Trigger the JPEG operation for encoding
JPEG_IOCTL_DECODE_TRIGGER			Trigger the JPEG operation for decoding
JPEG_IOCTL_WINDOW_DECODE	JPEG_WINDOW_DECODE_T		Enable window decode mode and set the decode window region
JPEG_IOCTL_SET_DECODE_STRIDE	Decode Output Stride (in pixel)		Specify the decode output stride
JPEG_IOCTL_SET_DECODE_DOWNSCALE	Scaled Height	Scaled Width	Set Decode downscale function
JPEG_IOCTL_SET_ENCODE_UPSCALE	Scaled Height	Scaled Width	Set Encode Upscale function
JPEG_IOCTL_SET_HEADER_DECODE_CALLBACKFUN	Header Decode Complete Call Back function pointer		Set Header Decode Complete Call Back function pointer
JPEG_IOCTL_SET_DECODEINPUT_WAIT_CALLBACKFUN	Decode Input Wait Call Back function pointer		Set Decode Input Wait Call Back function pointer
JPEG_IOCTL_ADJUST_QTABLE	JPEG_ENC_PRIMARY JPEG_ENC_THUMBNAIL	Quantization-Table Adjustment and control values[0]	Set Quantization-Table Adjustment and control values
JPEG_IOCTL_ENC_RESERVED_FOR_SOFTWARE	Reserved size		Reserve memory space for user application
JPEG_IOCTL_SET_UADDR	Address for U Component		Set address for U Component
JPEG_IOCTL_SET_VADDR	Address for V Component		Set address for V Component
JPEG_IOCTL_SET_ENCODE_PRIMARY_RESTART_INTERVAL	Primary Restart interval		Set Primary Restart interval size

Command	Argument 0	Argument 1	comment
JPEG_IOCTL_SET_ENCODE_THUMBNAIL_RESTART_INTERVAL	Thumbnail Restart interval		Set Thumbnail Restart interval size
JPEG_IOCTL_GET_ENCODE_PRIMARY_RESTART_INTERVAL	The pointer to store Primary Restart interval size		Get Primary Restart interval size
JPEG_IOCTL_GET_ENCODE_THUMBNAIL_RESTART_INTERVAL	The pointer to store Thumbnail Restart interval size		Get Thumbnail Restart interval size
JPEG_IOCTL_SET_THUMBNAIL_DIMENSION	Thumbnail Height	Thumbnail Width	Set Thumbnail Dimension
JPEG_IOCTL_SET_ENCODE_SW_OFFSET	Offset		Set Software Encode Offset
JPEG_IOCTL_GET_THUMBNAIL_DIMENSION	The pointer to store Thumbnail 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_THUMBNAIL_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)
JPEG_IOCTL_SET_DECODE_OUTPUT_WAIT_CALLBACKFUN	Decode Output Wait call back function pointer		Set Decode Output Wait call back function (Packetformat Only)

Command	Argument 0	Argument 1	comment
JPEG_IOCTL_SET_DECOU PUTWAIT	Data Output Address	Data Output Size	Set Decode Output Wait address and size
JPEG_IOCTL_GET_DECOU PUTWAIT_ADDR	The pointer to store Decode Output Wait Address		Get Decode Output Wait Address
JPEG_IOCTL_GET_DECOU PUTWAIT_SIZE	The pointer to store Decode Output Wait Size		Get Decode Output Wait Size
JPEG_IOCTL_SET_DECODE _COMPLETE_CALLBACKFUN	Decode Complete Call Back function pointer		Set Decode Complete Call Back function pointer
JPEG_IOCTL_SET_ENCODE _COMPLETE_CALLBACKFUN	Encode Complete Call Back function pointer		Set Encode Complete Call Back function pointer
JPEG_IOCTL_SET_DECODE _ERROR_CALLBACKFUN	Decode Error Call Back function pointer		Set Decode Error Call Back function pointer

Table 16-3 JPEG IOCTL Definition

16.2.2 Structure

JPEG_INFO_T Structure

Field Name	Type	Value	Description
yuvfor mat	UINT32	JPEG_DEC_YUV420 JPEG_DEC_YUV422 JPEG_DEC_YUV444 JPEG_DEC_YUV411 JPEG_DEC_GRAY JPEG_DEC_YUV422T	JPEG format (Decode only)
width	UINT32	< 8192	Decode Output width (Decode only)
height	UINT32	< 8192	Decode Output height (Decode only)
jpeg_width	UINT32	< 65535	JPEG width (Decode only)
jpeg_height	UINT32	< 65535	JPEG height (Decode only)
stride	UINT32	< 8192	Decode output Stride (Decode only)
bufferend	UINT32	Reserved	Reserved
image_size[2]	UINT32	< 2 ²⁴ -1	Encode Bitstream Size (Encode Only)

Table 16-4 JPEG_INFO_T Structure Definition

16.3 API function

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_JPEG_SUCCESS Sccesses

Example

```
jpegOpen();
```

jpegClose

Synopsis

VOID jpegClose (VOID)

Description

Disable clock of JPEG engine and disable its interrupt

Parameter

None

Return Value

None

Example

```
jpegClose();
```

jpegInit

Synopsis

VOID jpegInit (VOID)

Description

Reset JPEG engine and set default value to its registers

Parameter

None

Return Value

None

Example

```
jpegInit();
```

jpegGetInfo

Synopsis

```
VOID jpegGetInfo (
    JPEG_INFO_T *info
)
```

Description

This function can get JPEG width and height after header decode complete and get JPEG bit stream size after encode complete.

Parameter

info [out]
 JPEG Data type pointer stores the returned JPEG header information

Return Value

None

Example

```
JPEG_INFO_T jpegInfo;
/* Get JPEG Header information */
jpegGetInfo(&jpegInfo);
```

jpegWait

Synopsis

```
INT
jpegWait (VOID)
```

Description

After triggers JPEG engine, application need to wait the completion flag while JPEG engine completes it job.

Parameter

None

Return Value

E_JPEG_FAIL	Error happen
E_JPEG_SUCCESS	Action is done

Example

```
/* Trigger JPEG encoder */
jpegIoctl(JPEG_IOCTL_ENCODE_TRIGGER, 0, 0);
/* Wait for complete */
```

```
if(jpegWait())
{
    jpegGetInfo(&jpegInfo);
    sysprintf("\tJPEG Encode Complete!!\n");
    sysprintf("\tJpeg Image Size = %d\n",jpegInfo.image_size[0]);
    len = jpegInfo.image_size[0];
}
else
{
    sysprintf("\tJPEG Encode Error!!\n");
    len = 0;
    return;
}
```

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

```
/* Trigger JPEG encoder */
jpegIoctl(JPEG_IOCTL_ENCODE_TRIGGER, 0, 0);

while(!jpegIsReady()); /* Wait for complete */
jpegGetInfo(&jpegInfo);
sysprintf("\tJPEG Encode Complete!!\n");
sysprintf("\tJpeg Image Size = %d\n",jpegInfo.image_size[0]);
len = jpegInfo.image_size[0];
```

jpegSetQTAB

Synopsis

INT

```
jpegSetQTAB (
    PUINT8 puQTable0,
    PUINT8 puQTable1,
    PUINT8 puQTable2,
    UUINT8 u8num
);
```

Description

The function can specify the Quantization table

Parameter

puQTable0 [in]

Specify the address of Quantization table 0

puQTable1 [in]

Specify the address of Quantization table 1

puQTable2 [in]

Specify the address of Quantization table 2

u8num [in]

Specify the number of Quantization table

Return Value

E_JPEG_SUCCESS	Success
E_JPEG_TIMEOUT	Set Quantization table timeout

Example

```
jpegSetQTAB(g_au8QTable0,g_au8QTable1, 0, 2);
```

jpegIoctl

Synopsis

```
VOID jpegIoctl (
    UUINT32 cmd,
    UUINT32 arg0,
    UUINT32 arg1
);
```

Description

This function allows programmers configure JPEG engine, the supported command and arguments listed in the Table 14-1 I2C IOCTL Command Definition.

Parameter

cmd [in]

Command

arg0 [in/out]

First argument of the command

arg1 [in/out]

Second argument of the command

Return Value

None

Example

```
/* Set Downscale to QVGA */
jpegIoctl(JPEG_IOCTL_SET_DECODE_DOWNSCALE, 240, 320);

/* Set Decode Stride to Panel width (480 pixel)*/
jpegIoctl(JPEG_IOCTL_SET_DECODE_STRIDE, 480, 0);

/* Set Decoded Image Address */
jpegIoctl(JPEG_IOCTL_SET_YADDR, u32FrameBuffer, 0);

/* Set Bit stream Address */
jpegIoctl(JPEG_IOCTL_SET_BITSTREAM_ADDR, u32BitStream, 0);

/* Set Decode Input Wait mode (Input wait buffer is 8192) */
jpegIoctl(JPEG_IOCTL_SET_DECINPUTWAIT_CALLBACKFUN, (UINT32) JpegDecInputWait,
8192);

/* Decode mode */
jpegIoctl(JPEG_IOCTL_SET_DECODE_MODE, JPEG_DEC_PRIMARY_PACKET_YUV422, 0);

/* Set JPEG Header Decode End Call Back Function */
jpegIoctl(JPEG_IOCTL_SET_HEADERDECODE_CALLBACKFUN, (UINT32)
JpegDecHeaderComplete, 0);

/* Trigger JPEG decoder */
jpegIoctl(JPEG_IOCTL_DECODE_TRIGGER, 0, 0);

/* Set Source Y/U/V Stride */
jpegIoctl(JPEG_IOCTL_SET_YSTRIDE, u16Width, 0);
jpegIoctl(JPEG_IOCTL_SET_USTRIDE, u16Width/2, 0);
```

```

jpegIoctl(JPEG_IOCTL_SET_VSTRIDE, u16Width/2, 0);

/* Primary Encode Image Width / Height */
jpegIoctl(JPEG_IOCTL_SET_DIMENSION, u16Height, u16Width);

/* Encode upscale 2x */
jpegIoctl(JPEG_IOCTL_SET_ENCODE_UPSCALE, u16Height * 2, u16Width * 2);

/* Set Encode Source Image Height */
jpegIoctl(JPEG_IOCTL_SET_SOURCE_IMAGE_HEIGHT, u16Height, 0);

/* Include Quantization-Table and Huffman-Table */
jpegIoctl(JPEG_IOCTL_ENC_SET_HEADER_CONTROL, JPEG_ENC_PRIMARY_QTAB |
JPEG_ENC_PRIMARY_HTAB, 0);

/* Use the default Quantization-table 0, Quantization-table 1 */
jpegIoctl(JPEG_IOCTL_SET_DEFAULT_QTAB, 0, 0);

```

jpegPollInt

Synopsis

```

BOOL
jpegPollInt (
    UINT32 u32Intflag
)

```

Description

The function can get the JPEG engine status.

Parameter

None

Return Value

TRUE	Engine is ready
FALSE	Engine is busy

Example

```

If(jpegPollInt(JPEG_ENC_INTS))
    sysprintf("Get Encode Complete Interrupt\n");

```

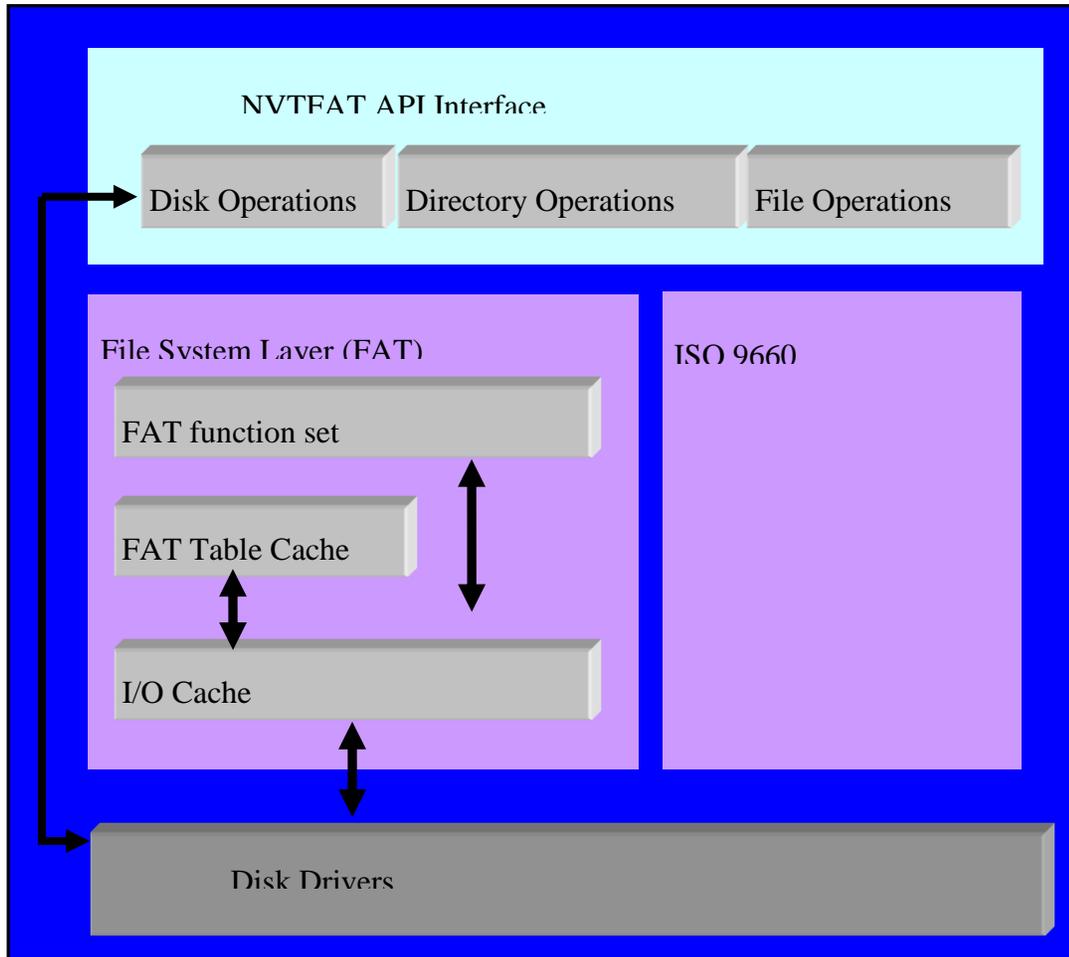
17 NVTFAT Library

17.1 NVTFAT Library Overview

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

17.1.1 System Overview



17.1.2 Initialize File System

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

17.1.3 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.

17.1.4 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()`.

17.1.5 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 17-1 Disk Format.

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 17-1 Disk Format

17.1.6 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.

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 ≥ 3000 . Otherwise, the call will receive a negative value, which represented an error code (refer to Error Code Table).

Function fsOpenFile() receives third parameters. The first parameter is the Unicode full path file name of the file to be opened. The full path file name must also include disk number. For example, "C:\\OpenATestFile.txt". The second parameter could be Null or ASCII code name. The third parameter is combination of control flags. It uses bit-OR to represent various control flags. The control flags and their effectives are listed in Table 17-2 File open control flags.

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.

Flag	Description
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 17-2 File open control flags

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.

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()

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()

Directory Operations

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

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.

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.

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.

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.

17.2 Definition

17.2.1 Constant

Seek Position Base

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

Table 17-3 Seek Position Base Definition

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

Table 17-4 Open File capability Definition

17.3 API function

17.3.1 Disk Operations

fsPhysicalDiskConnected

Synopsis

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

Description

Register and parsing a newly detected disk.

Parameter

ptPDisk [in]
The pointer refers to the physical disk descriptor

Return Value

0	Success
Otherwise	Error code defined in Table 17-5 NVTFAT 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 [in]
 The pointer refers to the physical disk descriptor

Return Value

0 Success
 Otherwise Error code defined in Table 17-5 NVTFAT 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 [in]
 The pointer refers to the physical disk descriptor

Return Value

0	Success
Otherwise	Error code defined in Table 17-5 NVT FAT 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. 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 [in]

The disk drive number

puBlockSize [out]

The size of stroage block. For FAT, it's cluster size

puFreeSize [out]

The size in Kbytes of disk free space.

puDiskSize [out]

The size in Kbytes of disk

Return Value

0 Success

Otherwise Error code defined in Table 17-5 NVTFAT 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 [in]

The pointer refers to the physical disk descriptor.

Return Value

0 Success

Otherwise Error code defined in Table 17-5 NVT FAT 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 */
fsFormatFlashMemoryCard (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, NVT FAT will automatically shrink the size of the second partition to fit disk size.

Parameter

ptPDisk [in]

The pointer refers to the physical disk descriptor.

firstPartSize [in]

The size (in KBs) of the first partition

secondPartSize [in]

The size (in KBs) of the second partition.

Return Value

0	Success
Otherwise	Error code defined in Table 17-5 NVTFAT 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 */
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 [in]
The drive number. Valid number is 'A' ~ 'Z'.
- disk_type [in]
Disk type defines in *nvtfat.h*. Prefixed with "DISK_TYPE_". For example,

NAND disk type is DISK_TYPE_SMART_MEDIA

instance [in]

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 [in]

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 Table 17-5 NVT FAT 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 [in]
 The drive number. Valid number is 'A' ~ 'Z'.

Return Value

0 Success
 Otherwise Error code defined in Table 17-5 NVT FAT 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');
```

fsGetFullDiskInformation

Synopsis

PDISK_T *
 fsGetFullDiskInformation (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 Table 17-5 NVT FAT Error Code Table

Example

```
PDISK_T *pDiskList, *ptPDiskPtr;
PARTITION_T *ptPartition;
INT nDiskIdx = 0;
INT nPartIdx;
ptPDiskPtr = pDiskList = fsGetFullDiskInformation();
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;
    }
    ptPDiskPtr = ptPDiskPtr->ptPDiskAllLink;
}
fsReleaseDiskInformation(pDiskList);S

```

fsReleaseDiskInformation

Synopsis

```

VOID fsReleaseDiskInformation (
    PDISK_T *ptPDiskList
)

```

Description

Release the memory allocated by fsGetFullDiskInformation().

Parameter

ptPDiskList [in]

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

Return Value

0	Success
Otherwise	Error code defined in Table 17-5 NVTFAT Error Code Table

Example

See example code of fsGetFullDiskInfomation()

fsInitFileSystem

Synopsis

INT
fsInitFileSystem (VOID)

Description

Initialize file system.

Parameter

None

Return Value

0 Success

Example

```
sysEnableCache(CACHE_WRITE_THROUGH);
fsInitFileSystem();
fmIInitDevice();
fmIInitSDDevice();
```

fsSetReservedArea

Synopsis

INT
fsSetReservedArea (
 UINT32 u32StartSector
)

Description

Specify the start sector in file system.

Parameter

u32StartSector [in]

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

0 Success

Example

```
#define RESERVED_SIZE (1024*1024)
```

```
fsSetReservedArea(REERVED_SIZE/512);/* Start sector from 2048 sector */
pDiskList = fsGetFullDiskInformation();
fsFormatFlashMemoryCard(pDiskList);
```

17.3.2 File/Directory Operations

fsCloseFile

Synopsis

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

Description

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

Parameter

hFile [in]
The file handle of the file to be closed.

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVTFAT Error Code Table

Example

Refer to the example of fsOpenFile ().

fsDeleteFile

Synopsis

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

Description

Delete a file.

Parameter

suFileName [in]

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 [in]

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>

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVTFAT 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 [in]
 The file handle of the file to be closed.

n64Offset [in]

Byte offset from the position indicated by <usWhence>

usWhence [in]

Seek position base. See Table 17-3 Seek Position Base Definition.

Return Value

0	Success
Otherwise	Error code defined in Table 17-5 NVT FAT Error Code Table

Example

```

INT  hFile, nReadLen;
CHAR suFileName[] = { 'C', 0, ':', 0, '\\', 0, 'I', 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 [in]
 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[in]
 The file-search object obtained by previous fsFindFirst() call.

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVTFAT 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 [in]
 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 [in]
 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 [out]

caller prepares file/directory entry container.

Return Value

0	Success
Otherwise	Error code defined in Table 17-5 NVT FAT 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 [out]
 The file-search object used in the previous fsFindFirst () call.

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVT FAT 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 [in]

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

puPos [out]

The current read/write position.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT Error Code Table

Example

```
INT hFile, nStatus;
UINT32 uFilePos;
CHAR szPath1[128], suFileName[512];
/* Open a read-only file */
memset(szPath1, 0, sizeof(szPath1));
sprintf(szPath1, "C:\\Test.jpg");
fsAsciiToUnicode(szPath1, suFullName1, TRUE);
hFile = fsOpenFile(suFileName, szPath1, O_RDONLY);
fsFileSeek(hFile, 1000, SEEK_SET);
fsGetFilePosition(hFile, &uFilePos);
printf("Current file position is: %d\n", uFilePos);
fsCloseFile(hFile);
```

fsGetFileSize

Synopsis

INT64

```
fsGetFileSize (
    INT hFile
)
```

Description

Get the current size of an opened file.

Parameter

hFile [in]

The file handle of the opened file to get size.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVTFAT Error Code Table

Example

```

INT hFile, nStatus;
CHAR  szPath1[128], suFileName[512];
/* Open a read-only file */
memset(szPath1, 0, sizeof(szPath1));
sprintf(szPath1, "C:\\Test.jpg");
fsAsciiToUnicode(szPath1, suFullName1, TRUE);
hFile = fsOpenFile(suFileName, szPath1, O_RDONLY);
sysprintf("The size of %s is %d\n", file, fsGetFileSize(hFile));
fcloseFile(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 [in]

The file handle of the opened file.

suFileName [in]

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

szAsciiName [in]

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

ptFileStat [in]

Caller prepares the container to receive status of this file.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT Error Code Table

Example

```
CHAR suFileName[] = { 'C', 0, ':', 0, '\\', 0, 'I', 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 [in]

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

szAsciiName [in]

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

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT 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 [in]

The Unicode full path name of the file/directory to be moved.
- szOldAsciiName [in]

The ASCII version name of < suOldName > excluding the path part.
- suNewName [in]

The Unicode full path name of the old file/directory to be moved to.
- szNewAsciiName [in]

The ASCII version name of < suNewName > excluding the path part.
- blsDirectory [in]

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

Return Value

- FS_OK Success
- Otherwise Error code defined in Table 17-5 NVT FAT 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 [in]

The Unicode full path name of the file to be copied.
- szSrcAsciiName [in]

The ASCII version name of < suSrcName > excluding the path part.
- suDstName [in]

The Unicode full path name of the file/directory to be generated.
- szDstAsciiName [in]

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

Return Value

- FS_OK Success
- Otherwise Error code defined in Table 17-5 NVT FAT 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 [in]
 The file handle of the file to be closed.

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVTFAT Error Code Table

Example

None

fsOpenFile

Synopsis

```

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

Description

Open/Create a file.

Parameter

suFileName [in]
 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 [in]
 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>.

uFlag [in]

See Table 17-4 Open File capability Definition.

Return Value

< 0	Error code defined in Table 17-5 NVT FAT Error Code Table
Otherwise	file handle

Example

```

INT hFile;
CHAR suFileName[] = { 'C', 0, ':', 0, '\\', 0, 'l', 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 *pucPtr,
    INT nBytes,
    INT *pnReadCnt
)
    
```

Description

Read <nBytes> of octets from an opened file

Parameter

hFile [in]
 The file handle of an opened file.

pucPtr [in]

Refer to the buffer to receive data read from the specified file

nBytes [in]

Number of bytes to read

pnReadCnt [out]

Number of bytes actually read.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT Error Code Table

Example

```

UINT8 pucBuff[4096];
INT hFileSrc, hFileOut;
INT nReadLen, nWriteLen, nStatus;
CHAR szPath1[128], suFileName[512];

memset(szPath1, 0, sizeof(szPath1));
sprintf(szPath1, "C:\\log.txt");
fsAsciiToUnicode(szPath1, suFullName1, TRUE);
hFileSrc = fsOpenFile(suFileName, szPath1, O_RDONLY);
if ((hFileSrc = fsOpenFile(suFileName, szPath1, O_RDONLY) < 0)
return hFileSrc;

memset(szPath1, 0, sizeof(szPath1));
sprintf(szPath1, "C:\\logcopy.txt");
fsAsciiToUnicode(szPath1, suFullName1, TRUE);
if ((hFileOut = fsOpenFile(suFileName, szPath1, 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 [in]
 The Unicode full path name of the directory to be removed.

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

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVT FAT 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 [in]

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

szOldAsciiName [in]

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

suNewName [in]

Rename into the Unicode full path name of the file/directory.

szNewAsciiName [in]

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

blsDirectory [in]

TRUE: is renaming a directory.

FALSE: is renaming a file.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT 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);
```

fsSetFileAttribute

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 [in]

The file handle of the opened file to be set attribute,

suFileName [in]

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

szAsciiName [in]

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

ptFileStat [in]

The specified file attribute.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT 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 [in]

The file handle of the opened file.

suFileName [in]

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

szAsciiName [in]

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

newSize [in]

Set the file size to be extended to or truncated.

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT 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 [in]

The file handle of the opened file.

suFileName [in]

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

szAsciiName [in]

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

ucYear [in]

Years from 1980. For example, for 2003, <year> is equal to 23.

ucMonth [in]

1 <= month <= 12

ucHour [in]

0 <= hour <= 23

ucMin [in]

0 <= min <= 59

unSec [in]

0 <= sec <= 59

Return Value

0 Success

Otherwise Error code defined in Table 17-5 NVTFAT 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 [in]
 The file handle of an opened file.

pucBuff [in]
 The buffer contains the data to be written

nBytes [in]
 Number of bytes to written

pnWriteCnt [out]
 Number of bytes actually written

Return Value

FS_OK	Success
Otherwise	Error code defined in Table 17-5 NVTFAT 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;
    }
}
    
```

17.3.3 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 [in]

The Unicode string to be translated. It must be ended with two 0x0 characters.

pvASCII [out]

Caller prepares the container to accommodate the translation result.

blsNullTerm [in]

Add a NULL character (0x0) to the end of pvASCII

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT 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.szLongName, 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 [out]

The ASCII string to be translated. It must be NULL-terminated.

pvUniStr [in]

Caller prepares the container to accommodate the translation result.

blsNullTerm [in]

Add two 0x0 characters to the end of pvUniStr

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT Error Code Table

Example

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

fsAsciiNonCaseCompare

Synopsis

```
INT
fsUnicodeNonCaseCompare (
```

```

        VOID * pvASCII1,
        VOID * pvASCII2
    )

```

Description

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

Parameter

pvASCII1 [in]

The source (0x0,0x0)-ended ASCII string to compared.

pvASCII2 [in]

The target (0x0,0x0)-ended ASCII string to compared.

Return Value

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

Example

```

CHAR szName1[] = "log.txt"
CHAR szName2[] = "Log.TXT";
CHAR suName1[32], suName2[32];
if (fsAsciiNonCaseCompare(szName1, szName2) == 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 [in]

The source (0x0,0x0)-ended Unicode string to compared.
 pvUnicode2 [in]

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 [out]
 The Unicode string to be copied to.
 pvStr2 [in]
 The source Unicode string. It must be (0x0,0x0)-ended.

Return Value

FS_OK Success
 Otherwise Error code defined in Table 17-5 NVT FAT 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 [out]

The Unicode string to be concatenated to.

pvUniStr2 [in]

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

Return Value

FS_OK Success

Otherwise Error code defined in Table 17-5 NVT FAT Error Code Table

Example

Refer to the example of fsUnicodeCopyStr();

fsToUpperCase

Synopsis

```
INT
fsToUpperCase (
    CHAR chr
)
```

Description

Translate a character to uppercase.

Parameter

```
chr [in]
    A character will be translated to uppercase.
```

Return Value

An uppercase character

Example

None

fsAsciiToUpperCase**Synopsis**

```
INT
fsAsciiToUpperCase (
    VOID *pvASCII
)
```

Description

Translate a NULL-terminated ASCII string to uppercase.

Parameter

```
pvASCII [in]
    The NULL-terminated ASCII string.
```

Return Value

None

Example

None

fsUnicodeToUpperCase**Synopsis**

```
INT
fsUnicodeToUpperCase (
```

```
        VOID * pvUnicode
    )
```

Description

Translate a (0x00, 0x00) double NULL-terminated unicode string to uppercase.

Parameter

pvUnicode [in]
 The double NULL-terminated Unicode string.

Return Value

None

Example

None

fsUnicodeToLowerCase

Synopsis

```
INT
fsUnicodeToLowerCase (
    VOID * pvUnicode
)
```

Description

Translate a (0x00, 0x00) double NULL-terminated unicode string to lowercase.

Parameter

pvUnicode [in]
 The double NULL-terminated Unicode string.

Return Value

None

Example

None

fsUnicodeToLowerCase

Synopsis

```
INT
fsUnicodeToLowerCase (
    VOID * pvUnicode
)
```

Description

Translate a (0x00, 0x00) double NULL-terminated unicode string to lowercase.

Parameter

pvUnicode [in]

The double NULL-terminated Unicode string.

Return Value

None

Example

None

fsUnicodeStrLen

Synopsis

INT

```
fsUnicodeStrLen (
    VOID * pvUnicode
)
```

Description

Calculate the string length for a (0x00, 0x00) double NULL-terminated unicode string.

Parameter

pvUnicode [in]

The double NULL-terminated Unicode string.

Return Value

String length

Example

None

17.4 NVTFAT 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

Code Name	Value	Description
ERR_INTERNAL_ERROR	0xFFFF820C	File system internal error
ERR_SYSTEM_LOCK	0xFFFF820E	File system locked by ScanDisk or Defragment
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_DIR_MOVE_DISK	0xFFFF829E	Cannot move the whole directory from disk to another disk

Code Name	Value	Description
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_DEL_OPENED	0xFFFF82CC	Try to delete a file, which has been opened
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
ERR_CHECKDISK_FILE_OPENED	0xFFFF83A4	There's file opened, cannot do scandisk operation
ERR_CHECKDISK_LOCK	0xFFFF83A6	Service locked by check disk operation

Table 17-5 NVTFAT Error Code Table

18 PWM Library

18.1 PWM Library Overview

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

- Support hardware Scatter-Gather function. PWM signal frequency and duty setting
- PWM Capture function

18.2 Definition

18.2.1 Constant

PWM Definition

Name	Value	Description
PWM_TIMER0	0x0	PWM Timer 0
PWM_TIMER1	0x1	PWM Timer 1
PWM_TIMER2	0x2	PWM Timer 2
PWM_TIMER3	0x3	PWM Timer 3
PWM_CAP0	0x0	PWM Capture 0
PWM_CAP1	0x1	PWM Capture 1
PWM_CAP2	0x2	PWM Capture 2
PWM_CAP3	0x3	PWM Capture 3
PWM_CAP_ALL_INT	0	All PWM Capture Interrupt
PWM_CAP_RISING_INT	1	PWM Capture Rising Interrupt
PWM_CAP_FALLING_INT	2	PWM Capture Falling Interrupt
PWM_CAP_RISING_FLAG	6	Capture rising interrupt flag
PWM_CAP_FALLING_FLAG	7	Capture falling interrupt flag
PWM_CLOCK_DIV_1	4	Input clock divided by 1
PWM_CLOCK_DIV_2	0	Input clock divided by 2
PWM_CLOCK_DIV_4	1	Input clock divided by 4
PWM_CLOCK_DIV_8	2	Input clock divided by 8
PWM_CLOCK_DIV_16	3	Input clock divided by 16
PWM_TOGGLE_MODE	TRUE	PWM Timer Toggle mode
PWM_ONE_SHOT_MODE	FALSE	PWM Timer One-shot mode
PWM_TIMER0	0x0	PWM Timer 0
PWM_TIMER1	0x1	PWM Timer 1
PWM_TIMER2	0x2	PWM Timer 2

Name	Value	Description
PWM_TIMER3	0x3	PWM Timer 3
PWM_CAP0	0x0	PWM Capture 0
PWM_CAP1	0x1	PWM Capture 1
PWM_CAP2	0x2	PWM Capture 2
PWM_CAP3	0x3	PWM Capture 3
PWM_CAP_ALL_INT	0	All PWM Capture Interrupt
PWM_CAP_RISING_INT	1	PWM Capture Rising Interrupt
PWM_CAP_FALLING_INT	2	PWM Capture Falling Interrupt
PWM_CAP_RISING_FLAG	6	Capture rising interrupt flag
PWM_CAP_FALLING_FLAG	7	Capture falling interrupt flag
PWM_CLOCK_DIV_1	4	Input clock divided by 1
PWM_CLOCK_DIV_2	0	Input clock divided by 2
PWM_CLOCK_DIV_4	1	Input clock divided by 4
PWM_CLOCK_DIV_8	2	Input clock divided by 8
PWM_CLOCK_DIV_16	3	Input clock divided by 16
PWM_TOGGLE_MODE	TRUE	PWM Timer Toggle mode
PWM_ONE_SHOT_MODE	FALSE	PWM Timer One-shot mode

Table 18-1 PWM Definition

18.2.2 Structure

PWM_TIME_DATA_T Structure

Field name	Data Type	Value	Description
u8Mode	UINT8	PWM_ONE_SHOT_MODE / PWM_TOGGLE_MODE	PWM Timer Trigger mode
fFrequency	FLOAT	>= 0	The timer/capture frequency ^[Note 0]
u8HighPulseRatio	UINT8	1~100	High pulse ratio
bInverter	BOOL	TRUE / FALSE	Inverter Enable / Inverter Disable
u8ClockSelector	UINT8	PWM_CLOCK_DIV_1/ PWM_CLOCK_DIV_2/ PWM_CLOCK_DIV_4/ PWM_CLOCK_DIV_8/ PWM_CLOCK_DIV_16	Clock Selector ^[Note 1]
u16PreScale	UINT16	2 ~ 256	Clock Prescale ^[Note 1]
u32Duty	UINT32	0~65535	Pulse duty ^[Note 2]

Table 18-2 PWM_TIME_DATA_T Structure Definition

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

- Frequency-setting mode (u8Frequency > 0)
User doesn't need to set u8ClockSelector / u16PreScale / u32Duty fields. PWM library will set the proper values according to current APB clock automatically.
- Property-setting mode (u8Frequency = 0)
- User must set u8ClockSelector / u16PreScale / u32Duty fields by himself. Please refer to the previous section "Prescaler and clock selector."

[Note 1] The value take effect only when Property-setting mode.

[Note 2] The value takes effect when Property-setting mode or the Capture functions. It is the capture monitor period.

18.3 API function

PWM_Open

Synopsis

VOID PWM_Open (void)

Description

Enable PWM engine clock and reset PWM.

Parameter

None

Return Value

None

Example

```
/* Enable PWM clock */
PWM_Open();
```

PWM_Close

Synopsis

VOID PWM_Close (void)

Description

Disable PWM engine clock and the I/O enable.

Parameter

None

Return Value

None

Example

```
/* Disable PWM clock */
PWM_Close();
```

PWM_SetClockSetting

Synopsis

```
BOOL
PWM_SetClockSetting (
    E_SYS_SRC_CLK eSrcClk,
    UINT32 u32PIIDiver,
    UINT32 u32EngineDiver
)
```

Description

This function is used to set PWM engine clock source and divider.

Parameter

- eSrcClk [in]
PWM clock source.
It could be eSYS_EXT = 0, eSYS_APLL = 2 and eSYS_UPLL = 3.
- u32PIIDiver [in]
PWM PLL Divider Selection (1~8)
- u32EngineDiver [in]
Engine Clock divider (1~256)

Return Value

- TRUE Success.
- FALSE Setting Fail.

Note

Parameter “u32PIIDiver” is only be valid when eSrcClk is eSYS_APLL or eSYS_UPLL

Example

```
/* PWM Engine clock is UPLL / 4, and Engine Clock divider is 2 */
PWM_SetClockSetting(eSYS_UPLL, 4, 2);
```

PWM_GetEngineClock

Synopsis

```

UINT32
PWM_GetEngineClock (
    E_SYS_SRC_CLK *peSrcClk
)
    
```

Description

This function is used to get Current PWM engine clock

Parameter

peSrcClk [out]
 Sytem clock source.
 It could be eSYS_EXT = 0, eSYS_APLL = 2 and eSYS_UPLL = 3.

Return Value

PWM Engine Clock (Hz)

Example

```

u32PWMClock = PWM_GetEngineClock(&eSrcClk);
sysprintf("PWM Clock Source is ");
switch(eSrcClk)
{
    case eSYS_EXT:
        sysprintf("External Crystal\n");
        break;
    case eSYS_APLL:
        sysprintf("APLL\n");
        break;
    case eSYS_UPLL:
        sysprintf("UPLL\n");
        break;
}
sysprintf("PWM Clock is %dHz\n",u32PWMClock);
    
```

PWM_SetTimerClk

Synopsis

```

FLOAT
PWM_SetTimerClk (
    UINT8 u8Timer,
    PWM_TIME_DATA_T *sPt
)
    
```

Description

This function is used to configure the frequency/pulse/mode/inverter function

Parameter

u8Timer [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

sPt [in]

PWM property information

Return Value

= 0 Setting Fail.

> 0 Success. The actual frequency by PWM timer.

Note

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

The function can set the proper frequency property (Clock selector/Prescale) for capture function and user needs to set the proper pulse duty by himself.

Example

```
sPt.u8Mode = PWM_TOGGLE_MODE;
sPt.fFrequency = g_u16Frequency;
sPt.u8HighPulseRatio = 1; /* High Pulse period, Total Pulse peroid = 1 : 100 */
sPt.blInverter = FALSE;

/* 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 [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

bEnable [in]

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 [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Return Value

None

Example

```
/* Enable the PWM Timer0 */
PWM_Enable(PWM_TIMER0,TRUE);
```

PWM_IsTimerEnabled

Synopsis

```

BOOL
PWM_IsTimerEnabled (
    UINT8 u8Timer
)
    
```

Description

This function is used to get PWM specified timer enable/disable state.

Parameter

u8Timer [in]
 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 [in]
 The function to be set
 PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

u16Counter [in]
 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 [in]
 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 [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

u8Length [in]

Dead Zone Length : 0~255

bEnableDeadZone [in]

Enable DeadZone (TRUE) / Disable DeadZone (FALSE)

Return Value

None

Note

If Deadzone for PWM_TIMER0 or PWM_TIMER1 is enabled, the output of PWM_TIMER1 is inverse waveform of PWM_TIMER0.

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 [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

u8Int [in]

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

PWM_CAP_RISING_INT: The capture rising interrupt.

PWM_CAP_FALLING_INT: The capture falling interrupt.

PWM_CAP_ALL_INT: All capture interrupt.

Return Value

None

Example

```
/* Enable Interrupt Sources of PWM Timer 0 */
PWM_EnableInt(PWM_TIMER0, 0);
/* Enable Interrupt Sources of PWM Capture3 */
PWM_EnableInt(PWM_CAP3, PWM_CAP_FALLING_INT);
```

PWM_DisableInt

Synopsis

```
VOID PWM_DisableInt (
    UINT8 u8Timer,
    UINT8 u8Int
)
```

Description

This function is used to disable the PWM timer/capture interrupt.

Parameter

u8Timer [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

u8Int [in]

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 [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

Pfncallback [in]

The callback function pointer for specified timer / capture.

pfnOldcallback [out]

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 [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

Return Value

None

Example

```
/* Clear the PWM Capture 3 Interrupt */
PWM_ClearInt(PWM_CAP3);
```

PWM_GetIntFlag

Synopsis

```
BOOL
PWM_GetIntFlag (
    UINT8 u8Timer
)
```

Description

This function is used to get the PWM timer/capture interrupt flag.

Parameter

u8Timer [in]

The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

Return Value

- TRUE The specified interrupt occurs.
- FLASE The specified interrupt doesn't occur.

Example

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

PWM_GetCaptureIntStatus

Synopsis

```
VOID PWM_GetCaptureIntStatus (
```

```

        UINT8 u8Capture,
        UINT8 u8IntType
    )

```

Description

Check if there's a rising / falling transition.

Parameter

u8Capture [in]

The function to be set

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

u8IntType [in]

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

PWM_CAP_RISING_INT: The capture rising interrupt.

PWM_CAP_FALLING_INT: The capture falling interrupt.

Return Value

TRUE The specified interrupt occurs.

FLASE The specified interrupt doesn't occur.

Example

```

/* Wait for Interrupt Flag (Falling) */
while(PWM_GetCaptureIntStatus(PWM_CAP0, PWM_CAP_FALLING_FLAG)!=TRUE);

```

PWM_ClearCaptureIntStatus

Synopsis

```

VOID PWM_ClearCaptureIntStatus (
    UINT8 u8Capture,
    UINT8 u8IntType
)

```

Description

Clear the rising / falling transition interrupt flag.

Parameter

u8Capture [in]

The function to be set

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

u8IntType [in]

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

PWM_CAP_RISING_INT: The capture rising interrupt.

PWM_CAP_FALLING_INT: The capture falling interrupt.

Return Value

None

Example

```
/* Clear the Capture Interrupt Flag */
PWM_ClearCaptureIntStatus(PWM_CAP0, PWM_CAP_FALLING_FLAG);
```

PWM_GetRisingCounter

Synopsis

```
UINT16
PWM_GetRisingCounter (
    UINT8 u8Capture
)
```

Description

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

Parameter

u8Capture [in]
 The function to be set
 PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

Return Value

This function is used to get value which latches the counter when there's a rising transition.

Example

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

PWM_GetFallingCounter

Synopsis

```
UINT16
PWM_GetFallingCounter (
    UINT8 u8Capture
)
```

Description

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

Parameter

u8Capture [in]

The function to be set

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

Return Value

This function is used to get value which latches the counter when there's a falling transition

Example

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

19 RFC Library

19.1 RFC Library Overview

The RF-CODEC includes the Convolution encode, Viterbi decode, Inner Interleave, and Inner De-Interleave. These are a forward error correction code (FEC) for wireless transceiver. The convolution encode includes a puncture function to change the coding rate from 1/2 to 2/3, 3/4, 5/6, or 7/8. If selecting 7/8 coding rate, the transfer data rate is maximum; otherwise, if selecting 1/2 coding rate, it gains the maximum BER performance. The Viterbi Decode is hard decision and the trace-back length is 32. The interleave function is used to disperse the transfer data. Because the performance of the Viterbi decode will be worst by burst error. The RF-CODEC block diagram is in Figure1. One thing is important that the RF-CODEC only supports PDMA function to handle the data from or to memory.

- Supports Convolution encode and Viterbi decode
 - Coding rate supports 1/2, 2/3, 3/4, 5/6 and 7/8
- Supports Inner Interleave and Inner De-Interleave
 - Supports PDMA function to handle the data from or to memory

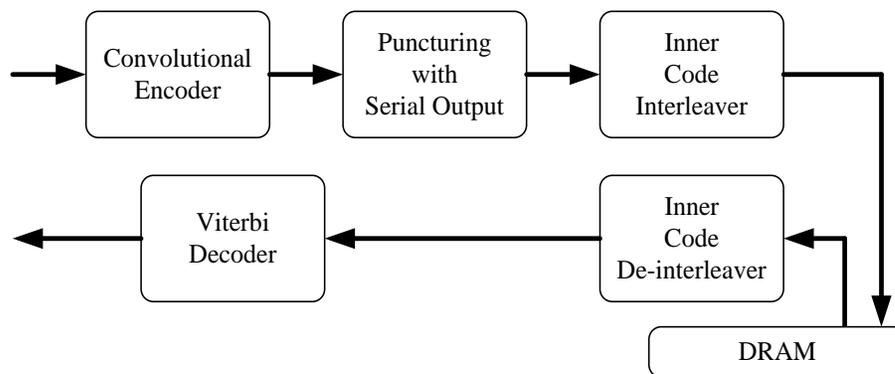


Figure 19-1 RFC Diagram

19.2 Definition

19.2.1 Constant

E_RF_PNCTR_MODE

Name	Value	Description
<i>E_PNCTR_1_2</i>	0	Coding rate is 1/2
<i>E_PNCTR_2_3</i>	1	Coding rate is 2/3
<i>E_PNCTR_3_4</i>	2	Coding rate is 3/4
<i>E_PNCTR_5_6</i>	3	Coding rate is 5/6
<i>E_PNCTR_7_8</i>	4	Coding rate is 7/8

Table 19-1 E_RF_PNCTR_MODE Definition

19.3 API function

RF_Open

Synopsis

INT32

RF_Open (void)

Description

Initialize RFC engine, install interrupt service routine, and call EDMA_Init to initialize PDMA engine.

Parameter

None

Return Value

Successful Always returns Successful

Example

None

RF_Close

Synopsis

void RF_Close (void)

Description

Tear down RFC engine.

Parameter

None

Return Value

None

Example

None

RF_Enable_Int

Synopsis

void RF_Enable_Int (void)

Description

Enable RFC interrupt source.

Parameter

None

Return Value

None

Example

None

RF_Disable_Int

Synopsis

```
void RF_Disable_Int (void)
```

Description

Disable RFC interrupt source.

Parameter

None

Return Value

None

Example

None

RF_Set_Puncture

Synopsis

```
INT32
RF_Set_Puncture (
    E_RF_PNCTR_MODE ePnctrMod
)
```

Description

Set the coding rate of the puncture function.

Parameter

ePnctrMod [in]
The coding rate of the puncture function

Return Value

Successful	Set puncture is successful
RFC_ERR_PNCTR_MODE	Invalid puncture coding rate

Example

None

RF_Get_Puncture

Synopsis

```
E_RF_PNCTR_MODE
RF_Get_Puncture (void)
```

Description

Get the coding rate of the puncture function.

Parameter

None

Return Value

Successful The puncture coding rate is returned

Example

None

RF_Encrypt

Synopsis

```
INT32
RF_Encrypt (
    UINT8 *plainBuf,
    UINT8 *cipherBuf,
    INT32 plainDataLen
)
```

Description

Start to run a RFC encryption calculation and wait for its finish.

Parameter

plainBuf [in]
 Pointer to input plain text buffer

cipherBuf [out]
 Pointer to output cipher text buffer

plainDataLen [in]
 Length of plain buffer in bytes

Return Value

(Value > 0) Length of output buffer in bytes

RFC_ERR_DATA_BUF RFC input buffer address is wrong

Example

None

RF_Decrypt

Synopsis

```
INT32
RF_Decrypt (
    UINT8 *cipherBuf,
    UINT8 *plainBuf,
    UINT32 plainDataLen
)
```

Description

Start to run a RFC decryption calculation and wait for its finish.

Parameter

cipherBuf [in]
 Pointer to input cipher text buffer

plainBuf [out]
 Pointer to output plain text buffer

plainDataLen [in]
 Length of plain buffer in bytes

Return Value

(Value > 0) Length of output buffer in bytes

RFC_ERR_DATA_BUF RFC input buffer address is wrong

Example

None

19.4 RFC Error Code Table

Code Name	Value	Description
RFC_ERR_FAIL	RFC_ERR_ID 0x01	Internal error
RFC_ERR_PNCTR_MODE	RFC_ERR_ID 0x02	Invalid puncture coding rate
RFC_ERR_DATA_BUF	RFC_ERR_ID 0x03	NULL buffer address

Table 19-2 RFC Error Code Table

20 Rotation Library

20.1 Rotation Library Overview

The N9H26 Rotation library provides a set of APIs to rotate image in SDRAM. It uses SRAM as temporary buffer. With these APIs, user can rotate image quickly.

The Rotation engine supports rotation left 90-degree and right 90-degree. It doesn't support downscale and format conversion. It only supports to rotate packet RGB565, packet XRGB888 and packet YUV422. It supports source line offset and destination line offset.

20.2 Definition

20.2.1 Constant

Interrupt Type

Name	Value	Description
E_ROT_COMP_INT	0	Rotation done interrupt
E_ROT_ABORT_INT	1	Memory abort interrupt
E_ROT_OVERFLOW_INT	2	Buffer overrun interrupt

Table 20-1 Interrupt Type Definition

Rotation Format

Name	Value	Description
E_ROT_PACKET_RGB565	0	Packet RGB565. The data width is 2
E_ROT_PACKET_RGB888	1	Packet RGB888. The data width is 4
E_ROT_PACKET_YUV422	2	Packet YUV422. The data width is 2

Table 20-2 Rotation Format Definition

20.2.2 Structure

T_ROT_CONF Structure

Field name	Data Type	Value	Description
eRotFormat	E_ROTENG_FMT	E_ROT_PACKET_RGB565 = 0 E_ROT_PACKET_RGB888 = 1 E_ROT_PACKET_YUV422 = 2	Pixel Format
eBufSize	E_ROTENG_BUFSIZE	E_LBUF_4 = 0 E_LBUF_8 = 2 E_LBUF_16 = 16	Use SRAM line.
eRotDir	UINT32	E_ROT_ROT_R90 = 0 E_ROT_ROT_L90 = 1	Right or left rotation

Field name	Data Type	Value	Description
u32RotDimHW	UINT32	[31:16] : Rotate image height [15:0] : Rotate image width	Rotate image dimension
u32SrcLineOffset	UINT32		Source line offset
u32DstLineOffset	UINT32		Source line offset
u32SrcAddr	UINT32		Source Buffer Address
u32DstAddr	UINT32		Destination Buffer Address

Table 20-3 T_ROT_CONF T Structure Definition

20.3 API function

rotOpen

Synopsis

void rotOpen (void)

Description

This function is used to open the rotation library.

Parameter

None

Return Value

None

Example

```
/* Open Rotation engine clock */
rotOpen();
```

rotClose

Synopsis

void rotClose (void)

Description

Close the rot library.

Parameter

None

Return Value

Noe

Example

```
/* Close Rotation library*/
rotClose();
```

rotInstallISR

Synopsis

```
void rotInstallCallback (
    UINT32 u32IntNum,
    PVOID pvIsr
)
```

Description

This function is used to install callback function that is used to notice the upper layer for specified rotation image is done, buffer overrun or memory abort.

Parameter

u32IntNum [in]
Rotation interrupt type. Please refer Table 20-1 Interrupt Type Definition

pvIsr [in]
Callback function

Return Value

None.

Example

```
rotOpen();
rotInstallISR(E_ROT_COMP_INT, (PVOID)rotDoneHandler); /* Rotation done */
rotInstallISR(E_ROT_ABORT_INT, (PVOID)rotAbortHandler); /* Memory Abort */
```

rotImageConfig

Synopsis

```
INT32
rotImageConfig (
    T_ROT_CONF *ptRotConf
)
```

Description

This function is used to configure Rotation engine.

Parameter

ptRotConf [out]
The structure to configure Rotation engine. Please refer Table 20-3

T_ROT_CONF T Structure Definition.

Return Value

Successful

Example

```
T_ROT_CONF tRotConf;
rotOpen();
rotInstallISR(E_ROT_COMP_INT, (PVOID)rotDoneHandler); /* Rotation done */
rotInstallISR(E_ROT_ABORT_INT, (PVOID)rotAbortHandler); /* Memory Abort */
tRotConf.eBufSize = E_LBUF_4;
tRotConf.eRotDir = E_ROT_ROT_L90;
tRotConf.eRotFormat = E_ROT_PACKET_RGB565;
tRotConf.u32RotDimHW = 0x01E00280;
tRotConf.u32SrcLineOffset = 0;
tRotConf.u32DstLineOffset = 0;
tRotConf.u32SrcAddr = ADDR_ROT_SRC_ADDR;
tRotConf.u32DstAddr = ADDR_ROT_DST_ADDR;
/* Set parameter then trigger ROT */
rotImageConfig(&tRotConf);
```

rotGetPacketPixelWidth

Synopsis

```
INT32
rotGetPacketPixelWidth (
    E_ROTENG_FMT ePacFormat
)
```

Description

The function is used to get data width for the specified format.

Parameter

ePacFormat [in]
 Rotation format. Please refer Table 20-2 Rotation Format Definition

Return Value

The data width of rotation image. Byte unit.

Example

```
UINT8 u8PixelWidth;
u8PixelWidth=rotGetPacketPixelWidth(ptRotConf->eRotFormat);
```

rotTrigger

Synopsis

INT32

rotTrigger (void)

Description

The function is used to get data width for the specified format

Parameter

None

Return Value

1: Meaning Rotation engine busy

0: Successful.

Example

```
/* Set parameter then trtrigger ROT */
rotImageConfig(&tRotConf);
rotClearDoneFlag();
rotTrigger();
```

20.4 Rotation Error Code Table

Code Name	Value	Description
ERR_ROT_BUSY	0xFFFF2001	Rotation engine is busy

Table 20-4 Rotation Error Code Table

21 RSC Library

21.1 RSC Library Overview

The RS_CODEC controller performs two main functions - Reed-Solomon Encoder / Decoder and Convolutional Interleaver / Deinterleaver. When in encode mode, data from system bus can be encoded by Reed-Solomon Encoder and interleaved by convolutional interleaver. When in decode mode, data from system bus can be de-interleaved and decoded by Reed-Solomon Decoder.

- Supports Reed-Solomon Encoder / Decoder
 - (N=204, K=188, t=8) with the Field Generator Polynomial: $p(x)=x^8+x^4+x^3+x^2+1$
 - Can correct 8 bytes error in 188 bytes block transmission
- Supports Convolutional Interleaver / Deinterleaver
 - Convolutional byte-wise interleaving with depth l=12 and 17 bytes FIFO
- Support PDMA to access RSC read / write buffers

21.2 API function

RS_Open

Synopsis

INT32

RS_Open (void)

Description

Initialize RSC engine, install interrupt service routine, and call EDMA_Init to initialize PDMA engine.

Parameter

None

Return Value

Successful Always returns Successful

Example

None

RS_Close

Synopsis

void RS_Close (void)

Description

Tear down RSC engine.

Parameter

None

Return Value

None

Example

None

RS_Enable_Int

Synopsis

void RS_Enable_Int (void)

Description

Enable RSC interrupt source.

Parameter

None

Return Value

None

Example

None

RS_Disable_Int

Synopsis

void RS_Disable_Int (void)

Description

Disable RSC interrupt source.

Parameter

None

Return Value

None

Example

None

RS_Encrypt

Synopsis

INT32

RS_Encrypt (

```

    UINT8 *plainBuf,
    UINT8 *cipherBuf,
    INT32 dataLen,
    UINT8 isInterleave
)

```

Description

Start to run a RSC encryption calculation and wait for its finish.

Parameter

```

plainBuf [in]
    Pointer to input plain text buffer
cipherBuf [out]
    Pointer to output cipher text buffer
dataLen [in]
    Length of input buffer in bytes
isInterleave [in]
    RSC runs in interleave or deinterleave mode

```

Return Value

(Value > 0)	Length of output buffer in bytes
RSC_ERR_DATA_LEN	RSC input data length is wrong
RSC_ERR_DATA_BUF	RSC input buffer address is wrong

Example

None

RS_Decrypt

Synopsis

```

INT32
RS_Decrypt (
    UINT8 *cipherBuf,
    UINT8 *plainBuf,
    UINT32 dataLen,
    UINT8 isInterleave
)

```

Description

Start to run a RSC decryption calculation and wait for its finish.

Parameter

cipherBuf [in]

Pointer to input cipher text buffer

plainBuf [out]

Pointer to output plain text buffer

dataLen [in]

Length of input buffer in bytes

isInterleave [in]

RSC runs in interleave or deinterleave mode

Return Value

(Value > 0)

Length of output buffer in bytes

RSC_ERR_DATA_LEN

RSC input data length is wrong

RSC_ERR_DATA_BUF

RSC input buffer address is wrong

RSC_ERR_DEC_ERROR

RSC decode error and cannot fix

Example

None

21.3 RSC Error Code Table

Code Name	Value	Description
RSC_ERR_FAIL	RSC_ERR_ID 0x01	Internal error
RSC_ERR_DATA_LEN	RSC_ERR_ID 0x02	Data length is not block alignment
RSC_ERR_DATA_BUF	RSC_ERR_ID 0x03	NULL buffer address
RSC_ERR_DEC_ERROR	RSC_ERR_ID 0x04	RSC decode error

Table 21-1 RSC Error Code Table

22 RTC Library

22.1 EDMA Library Overview

This library is designed to make user application access N9H26 RTC more easily.

The RTC library has the following features:

- There is a time counter (second, minute, hour) and calendar counter (day, month, year) for user to check the time.
- Absolute Alarm register (second, minute, hour, day, month, year).
- Relative Alarm
- Alarm Mask for Minutely / Hourly / Daily / Weekly / Monthly/Yearly Alarm
- 12-hour or 24-hour mode is selectable.
- Recognize leap year automatically.
- The day of week counter.
- Frequency compensate register (FCR).
- Beside FCR, all clock and alarm data expressed in BCD code.
- Support time tick interrupt.
- Support wake up function.
- System Power off Control function

22.2 Definition

22.2.1 Constant

RTC

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

Name	Value	Description
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	Aboslute Alarm Interrupt
RTC_TICK_INT	0x02	Tick Interrupt
RTC_PSWI_INT	0x04	Power Switch Interrupt
RTC_RELATIVE_ALARM_INT	0x08	Relative Alarm Interrupt
RTC_ALL_INT	0x0F	All Interrupt
RTC_IOC_IDENTIFY_LEAP_YEAR	0	Identify the leap year command
RTC_IOC_SET_TICK_MODE	1	Set tick mode command
RTC_IOC_GET_TICK	2	Get tick command
RTC_IOC_RESTORE_TICK	3	Restore tick command
RTC_IOC_ENABLE_INT	4	Enable interrupt command
RTC_IOC_DISABLE_INT	5	Disable interrupt command
RTC_IOC_SET_CURRENT_TIME	6	Set Current time command
RTC_IOC_SET_ALAMRM_TIME	7	Set Alarm time command
RTC_IOC_SET_FREQUENCY	8	Set Frequency command
RTC_IOC_SET_POWER_ON	9	Set Power On (Set PWR_ON to 1)
RTC_IOC_SET_POWER_OFF	10	Set Power Off (Set PWR_ON to 0)
RTC_IOC_SET_POWER_OFF_PERIOD	11	Set Power Off Period (PCLR_TIME)
RTC_IOC_ENABLE_HW_POWEROFF	12	Enable H/W Power Off
RTC_IOC_DISABLE_HW_POWEROFF	13	Disable H/W Power Off
RTC_IOC_GET_POWERKEY_STATUS	14	Get Power Key Status
RTC_IOC_SET_PSWI_CALLBACK	15	Set Power Switch Interrupt Callback function
RTC_IOC_GET_SW_STATUS	16	Get SW Status data (8 bits)
RTC_IOC_SET_SW_STATUS	17	Set SW Status data (8 bits)
RTC_IOC_SET_RELEATIVE_ALARM	18	Set relative alarm and install call back function
RTC_IOC_SET_POWER_KEY_DELAY	19	Minimum duration that power key must be pressed to turn on core power

Name	Value	Description
RTC_IOC_SET_CLOCK_SOURCE	20	Set RTC clock source
RTC_IOC_GET_CLOCK_SOURCE	21	Get RTC clock source
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

Table 22-1 RTC Definition

Command	Argument 0	Argument 1	comment
RTC_IOC_IDENTIFY_LEAP_YEAR	Unsigned integer pointer to store the return leap year value	None	Get the leap year
RTC_IOC_SET_TICK_MODE	Unsigned integer stores the tick mode data	None	Set Tick mode
RTC_IOC_GET_TICK	Unsigned integer pointer to store the return tick number	None	Get the tick counter
RTC_IOC_RESTORE_TICK	None	None	Restore the tick counter
RTC_IOC_ENABLE_INTERRUPT	interrupt type	None	Enable interrupt
RTC_IOC_DISABLE_INTERRUPT	interrupt type	None	Disable interrupt
RTC_IOC_SET_CURRENT_TIME	None	None	Set current time
RTC_IOC_SET_ALARM_TIME	None	None	Set alarm time
RTC_IOC_SET_FREQUENCY	Unsigned integer stores the Frequency Compensation value	None	Set Frequency Compensation Data
RTC_IOC_SET_POWER_ON	None	None	Set Power on
RTC_IOC_SET_POWER_OFF	None	None	Set Power off
RTC_IOC_SET_POWER_OFF_PERIOD	Unsigned integer stores the power off period value : 0~15	None	Set Power Off Period
RTC_IOC_ENABLE_HW_POWEROFF	None	None	Enable H/W Power Off

Command	Argument 0	Argument 1	comment
RTC_IOC_DISABLE_HW_POWEROFF	None	None	Disable H/W Power Off
RTC_IOC_GET_POWERKEY_STATUS	Unsigned integer pointer to store the return Power Key status	None	Get Power Key Status
RTC_IOC_SET_POWER_SWITCH_CALLBACK	The call back function pointer for Power Switch Interrupts	None	Set Power Switch Interrupt Callback function
RTC_IOC_GET_SW_STATUS	Unsigned integer pointer to store the return SW Status (8 Bits)	None	Get SW Status data (8 bits)
RTC_IOC_SET_SW_STATUS	Unsigned integer stores the SW Status data (8 Bits)	None	Set SW Status data (8 bits)
RTC_IOC_SET_RELATIVE_ALARM	The call back function pointer for Relative Alarm Interrupts	Alarm time (0~4095)	Set relative alarm and install call backfunction
RTC_IOC_SET_POWER_KEY_DELAY	power key duration	None	Minimum duration that power key must be pressed to turn on core power
RTC_IOC_SET_CLOCK_SOURCE	None	None	Set RTC clock source
RTC_IOC_GET_CLOCK_SOURCE	None	None	Get RTC clock source

Table 22-2 RTC IOCTL Definition

22.2.2 Structure

Time and Date Structure

Field name	Data Type	Value	Description
u8cClockDisplay	UINT8	RTC_CLOCK_12 / RTC_CLOCK_24	12 Hour Clock / 24 Hour Clock
u8cAmPm	UINT8	RTC_AM / RTC_PM	the AM hours / the PM hours
u32cSecond	UINT32	0~59	Second value
u32cMinute	UINT32	0~59	Minute value
u32cHour	UINT32	1~11 / 0~23	Hour value
u32cDayOfWeek	UINT32	RTC_SUNDAY~ RTC_SATURDAY	Day of week

Field name	Data Type	Value	Description
u32cDay	UINT32	1~31	Day value
u32cMonth	UINT32	1~12	Month value
u32Year	UINT32	0~99	Year value
u32AlarmMaskDayOfWeek	UINT32	0/1 (Disable/Enable)	Dya of Week Alarm Mask Enable
u32AlarmMaskSecond	UINT32	0/1 (Disable/Enable)	Second Alarm Mask Enable
u32AlarmMaskMinute	UINT32	0/1 (Disable/Enable)	Minute Alarm Mask Enable
u32AlarmMaskHour	UINT32	0/1 (Disable/Enable)	Hour Alarm Mask Enable
u32AlarmMaskDay	UINT32	0/1 (Disable/Enable)	Day Alarm Mask Enable
u32AlarmMaskMonth	UINT32	0/1 (Disable/Enable)	Month Alarm Mask Enable
u32AlarmMaskYear	UINT32	0/1 (Disable/Enable)	Year Alarm Mask Enable
pfnAlarmCallBack	PFN_RTC_CALLBACK *	Callback function pointer	Alarm interrupt Callback function

Table 22-3 Time and Date Structure Definition

22.3 API function

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 [in]
 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

```
/* Time Setting */
sInitTime.u32Year = 2010;
sInitTime.u32cMonth = 11;
sInitTime.u32cDay = 25;
sInitTime.u32cHour = 13;
sInitTime.u32cMinute = 20;
sInitTime.u32cSecond = 0;
sInitTime.u32cDayOfWeek = RTC_FRIDAY;
sInitTime.u8cClockDisplay = RTC_CLOCK_24;

/* Initialization the RTC timer */
if(RTC_Open(&sInitTime) !=E_RTC_SUCCESS)
    sysprintf("Open Fail!!\n");
```

RTC_Close

Synopsis

UINT32
 RTC_Close (VOID)

Description

Disable AIC channel of RTC and both tick and alarm interrupt

Parameter

None

Return Value

E_SUCCESS Success

Example

```
/* Disable RTC */
RTC_Close();
```

RTC_Read

Synopsis

UINT32
 RTC_Read (
 E_RTC_TIME_SELECT eTime,
 RTC_TIME_DATA_T *sPt
)

Description

Read current date/time or alarm date/time from RTC

Parameter

eTime [in]
 The current/alarm time to be read
 RTC_CURRENT_TIME - Current time
 RTC_ALARM_TIME - Alarm time
 sPt [out]
 RTC time property and 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 [in]
 The current/alarm time to be read
 RTC_CURRENT_TIME - Current time
 RTC_ALARM_TIME - Alarm time

sPt [in]
 RTC time property and 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

```
/* Time Setting */
sCurTime.u32Year = 2010;
sCurTime.u32cMonth = 1;
sCurTime.u32cDay = 8;
sCurTime.u32cHour = 10;
sCurTime.u32cMinute = 13;
sCurTime.u32cSecond = 0;
sCurTime.u32cDayOfWeek = RTC_FRIDAY;
sCurTime.u8cClockDisplay = RTC_CLOCK_24;
```

```
RTC_Write(RTC_CURRENT_TIME,&sCurTime);
```

RTC_EnableClock

Synopsis

```
VOID RTC_EnableClock (
    BOOL bEnable
)
```

Description

Enable / Disable RTC Clock

Parameter

bEnable [in]
TRUE/FALSE

Return Value

None

Example

```
/* Enable RTC Clock */
RTC_EnableClock(TRUE);
```

RTC_WriteEnable

Synopsis

```
UITN32
RTC_WriteEnable (
    BOOL bEnable
)
```

Description

Enable / Disable RTC register access

Parameter

bEnable [in]
TRUE/FALSE

Return Value

E_SUCCESS	Success
E_RTC_ERR_EIO	Access RTC Failed.

Example

```
/* Enable RTC Access */
RTC_WriteEnable(TRUE);
```

```
/* Disable RTC Access */
RTC_WriteEnable(FALSE);
```

RTC_DoFrequencyCompensation

Synopsis

```
UINT32
RTC_DoFrequencyCompensation (void)
```

Description

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

Parameter

None

Return Value

E_SUCCESS	Success
E_RTC_ERR_FCR_VALUE	Can't do compensation.

Example

```
RTC_DoFrequencyCompensation ()
```

RTC_ioctl

Synopsis

```
UINT32
RTC_ioctl (
    INT32 i32Num,
    E_RTC_CMD eCmd,
    UINT32 u32Arg0,
    UINT32 u32Arg1
)
```

Description

This function allows user to set some commands for application.

Parameter

- i32Num [in]
 - Please refer Table 22-2 RTC IOCTL Definition.
- u32Arg0 [in/out]
 - Please refer Table 22-2 RTC IOCTL Definition.
- u32Arg1 [in]

Please refer Table 22-2 RTC IOCTL Definition.

Return Value

None

Example

```

/* Set Tick setting */
RTC_ioctl(0,RTC_IOC_SET_TICK_MODE, (UINT32)&sTick,0);

/* Enable RTC Tick Interrupt and install tick call back function */
RTC_ioctl(0,RTC_IOC_ENABLE_INT, (UINT32)RTC_TICK_INT,0);

/* Press Power Key during 6 sec to Power off */
RTC_ioctl(0, RTC_IOC_SET_POWER_OFF_PERIOD, 6, 0);

/* Install the callback function for Power Key Press */
RTC_ioctl(0, RTC_IOC_SET_PSWI_CALLBACK, (UINT32)PowerKeyPress, 0);

/* Enable Hardware Power off */
RTC_ioctl(0, RTC_IOC_ENABLE_HW_POWEROFF, 0, 0);

/* Query Power Key Status */
RTC_ioctl(0, RTC_IOC_GET_POWERKEY_STATUS, (UINT32)&u32PowerKeyStatus, 0);

/* Power Off - S/W can call the API to power off any time he wants */
RTC_ioctl(0, RTC_IOC_SET_POWER_OFF, 0, 0);

/* Enable RTC Relative alarm Interrupt and install call back function */
RTC_ioctl(0,RTC_IOC_SET_RELEATIVE_ALARM, 10, (UINT32)RTC_Releative_AlarmISR);

/*
    Set Delay time Formula Minimum Power key duration =
    0.25*(POWER_KEY_DURATION+1) sec
*/
RTC_ioctl(0,RTC_IOC_SET_POWER_KEY_DELAY, 1, 0);
    
```

22.4 RTC Error Code Table

Code Name	Value	Description
E_RTC_SUCCESS	0	Operation success
E_RTC_ERR_CALENDAR_VALUE	1	Wrong Calendar Value

Code Name	Value	Description
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.

Table 22-4 RTC Error Code Table

23 SDIO Library

23.1 SDIO Library Overview

This library is designed to make user application access N9H26 SDIO(Secure-Digital Input / Output) controller more easily. This interface can directly connect to SD card.

The SDIO library has the following features:

- Support single DMA channel and address in non-word boundary.
- Support SD/SDHC/SDIO/MMCcard.

23.1.1 System Overview

The SDIO controller of N9H26 chip has DMAC unit and SD unit. The DMAC unit provides a DMA (Direct Memory Access) function for SD unit to exchange data between system memory (ex. SDRAM) and shared buffer (128 bytes), and the SD unit control the interface of SD/SDHC/SDIO/MMC. The SDIO controller can support SD/SDHC/SDIO/MMCcard and the SD unit is cooperated with DMAC unit to provide a fast data transfer between system memory and cards. The block diagram of SDIO controller is shown as following:

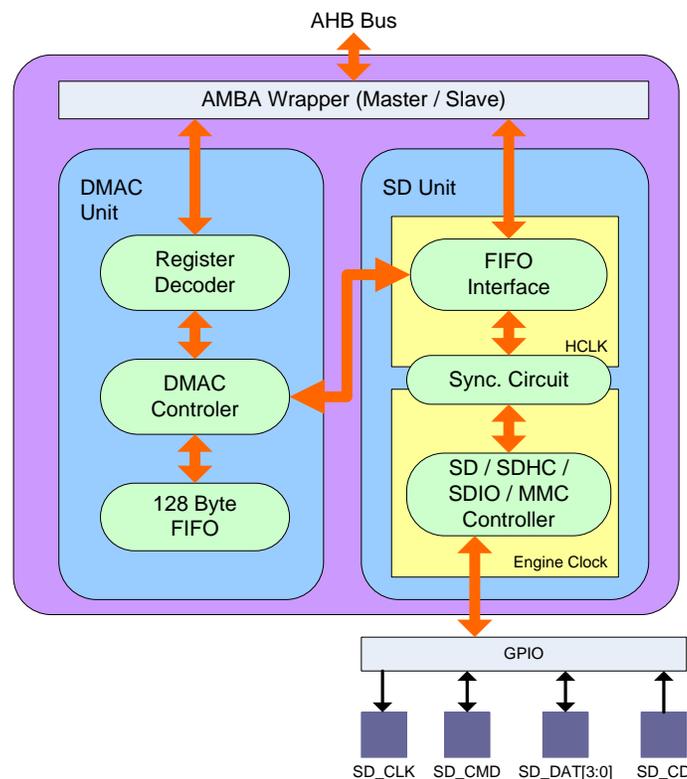


Figure 23-1 block diagram of SDIO controller

23.2 Definition

23.2.1 Constant

IOCTL

Command	Argument 0	Argument 1	Argument 2
SDIO_SET_CLOCK	AHB clock by KHz	None	None
SDIO_SET_CALLBACK	Card type (FMI_SDIO_CARD / FMI_SDIO1_CARD)	SD Card Remove callback function	SD Card Insert callback function
SDIO_GET_CARD_STATUS	Pointer to return value of SD card status	None	None
SDIO0_GET_CARD_STATUS	Pointer to return value of SD card status	None	None
SDIO1_GET_CARD_STATUS	Pointer to return value of SD card status	None	None
SDIO_SET_CARD_DETECT	enable/disable SDIO card detect feature	None	None
SDIO0_SET_CARD_DETECT	enable/disable SDIO0 card detect feature	None	None
SDIO1_SET_CARD_DETECT	enable/disable SDIO1 card detect feature	None	None

Table 23-1 SDIO IOCTL Definition

23.3 API function

23.3.1 SDIO API

sdioOpen

Synopsis

```
void sdioOpen (void)
```

Description

sdioOpen() will initialize the SDIO and DMAC interface hardware. It configures GPIO to SDIO mode, and installs ISR. This function is board dependent. It probably needs some modifications before it can work properly on your target board.

Parameter

None

Return Value

None

Example

```
/* initialize SDIO mode */  
sdiolctl(SDIO_SET_CLOCK, 192000, 0, 0); /* clock from PLL */  
sdioOpen();
```

sdioClose

Synopsis

```
void sdioClose (void)
```

Description

sdioClose() will Close the SDIO and DMAC interface hardware. It configures GPIO to close DMAC and disable ISR for SDIO.

Parameter

None

Return Value

None

Example

```
sdioClose();
```

sdiolctl

Synopsis

```
VOID sdiolctl (
```

```

        INT32 sdioFeature,
        INT32 sdioArg0,
        INT32 sdioArg1,
        INT32 sdioArg2
    )

```

Description

sdioIoctl() allows user set engine clock and callback functions, the support features and arguments listed in Table 23-1 SDIO IOCTL Definition.

Parameter

```

sdioFeature [in]
    Please refer Table 23-1 SDIO IOCTL Definition.
sdioArg0 [in/out]
    Please refer Table 23-1 SDIO IOCTL Definition.
sdioArg1[in]
    Please refer Table 23-1 SDIO IOCTL Definition.
sdioArg2 [in]
    Please refer Table 23-1 SDIO IOCTL Definition.

```

Return Value

For SDIO_GET_CARD_STATUS, the card status assign to sdioArg0. The value TRUE means SD card inserted, FALSE means SD card removed.

Example

Refer to the example code of sdioOpen ().

23.3.2 SDIO SD API

sdioSdOpen

Synopsis

```

INT
sdioSdOpen (void)          open SD card 0
INT
sdioSdOpen0 (void)        open SD card 0
INT
sdioSdOpen1 (void)        open SD card 1

```

Description

This function initialize the SDIO host interface and program the SD card from

identify mode to stand-by mode.

Parameter

None

Return Value

>0 Total sectornumber of SD card
 Otherwise Refer error code defined in Error Code Table

Example

```
if (sdioSdOpen0()<= 0)    /* Open SDIO port 0 */
{
    printf("Error in initializing SD card !! \n");
    /* handle error status */
}
```

sdioSdClose

Synopsis

void sdioSdClose (void) close SD card 0
 void sdioSdClose0 (void) close SD card 0
 void sdioSdClose1 (void) close SD card 1

Description

This function close the SDIO host interface.

Parameter

None

Return Value

None

Example

```
sdioSdClose();            /* Close SDIO port 0 */
```

sdioSdRead

Synopsis

INT
 sdioSdRead (
 INT32 sdSectorNo,
 INT32 sdSectorCount,
 INT32 sdTargetAddr
) for SD card 0

```

INT
sdioSdRead0 (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdTargetAddr
)                                for SD card 0
INT
sdioSdRead1 (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdTargetAddr
)                                for SD card 1
    
```

Description

This function will read the data from SD card.

Parameter

sdSectorNo [in]
 Sector No. to get the data from

sdSectorCount [in]
 Sector count of this access

sdTargetAddr [out]
 The address which data upload to SDRAM

Return Value

0	On success
FMISDIO_TIMEOUT	Access timeout
FMISDIO_NO_SD_CARD	Card removed
FMISDIO_SD_CRC7_ERROR	Command/Response error
FMISDIO_SD_CRC16_ERROR	Data transfer error

Example

```

#define FMI_TEST_SIZE (512*128)
__align(4096) UINT8 fmiReadBackBuffer[FMI_TEST_SIZE];
/* read 128 sectors data from SD card sector address 300 */
status = sdioSdRead(300, FMI_TEST_SIZE/512, (unsigned int)fmiReadBackBuffer);
    
```

sdioSdWrite

Synopsis

```

INT
sdioSdWrite (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdSourceAddr
)                                for SD card 0
INT
sdioSdWrite0 (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdSourceAddr
)                                for SD card 0
INT
sdioSdWrite1 (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdSourceAddr
)                                for SD card 1
    
```

Description

This function writes the data into SD card.

Parameter

```

sdSectorNo [in]
    Sector No. to get the data from
sdSectorCount [in]
    Sector count of this access
sdSourcetAddr [in]
    The address which download data from SDRAM
    
```

Return Value

```

0                                On success
FMISDIO_TIMEOUT                 Access timeout
    
```

FMISDIO_NO_SD_CARD	Card removed
FMISDIO_SD_CRC7_ERROR	Command/Response error
FMISDIO_SD_CRC_ERROR	Data transfer error

Example

```
#define FMI_TEST_SIZE (512*128)
__align(4096) UINT8 fmiFlash_Buf[FMI_TEST_SIZE];
/* write 128 sectors data to SD card sector address 3000 */
status = sdioSdWrite(3000, FMI_TEST_SIZE/512, (unsigned int)fmiFlash_Buf);
```

23.4 SDIO Error Code Table

Code Name	Value	Description
FMISDIO_TIMEOUT	0xFFFF00101	Access timeout
FMISDIO_NO_MEMORY	0xFFFF00102	No available memory
Error Code for SD Card		
FMISDIO_NO_SD_CARD	0xFFFF00110	NoSD card insert
FMISDIO_ERR_DEVICE	0xFFFF00111	Unknown device type
FMISDIO_SD_INIT_TIMEOUT	0xFFFF00112	SD initial time out
FMISDIO_SD_SELECT_ERROR	0xFFFF00113	Select card from identify mode to stand-by mode error
FMISDIO_SD_WRITE_PROTECT	0xFFFF00114	SD card write protection
FMISDIO_SD_INIT_ERROR	0xFFFF00115	SD Card initial and identify error
FMISDIO_SD_CRC7_ERROR	0xFFFF00116	Command/Response error
FMISDIO_SD_CRC16_ERROR	0xFFFF00117	Data reading error
FMISDIO_SD_CRC_ERROR	0xFFFF00118	Data writing error
FMISDIO_SD_CMD8_ERROR	0xFFFF00119	SD command 8 error

Table 23-2 SDIO Error Code Table

24 SIC Library

24.1 SIC Library Overview

This library is designed to make user application access N9H26 Storage Interface Controller (SIC) more easily. This interface can directly connect to SD and NAND Flash.

The SIC library has the following features:

- Support single DMA channel and address in non-word boundary.
- Support SD/SDHC/SDIO/MMCcard.
- Supports SLC and MLC NAND type Flash.
- Adjustable NAND page sizes. (512 / 2048 / 4096 / 8192 bytes +spare area)
- Support up to 4bit/8bit/12bit/15bit / 24bit hardware ECC calculation circuit to protect data communication.
- Programmable NAND/SM timing cycle.

24.1.1 System Overview

The Storage Interface Controller (SIC) of N9H26 chip has SIC_DMAMAC unit and SIC_FMI unit. The SIC_DMAMAC 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 controls the interface of SD/SDHC/SDIO/MMC or NAND/SM. The storage interface controller can support SD/SDHC/SDIO/MMCcard and NAND-type flash and the FMI is cooperated with DMAMAC to provide a fast data transfer between system memory and cards. The block diagram of SIC controller is shown as following:

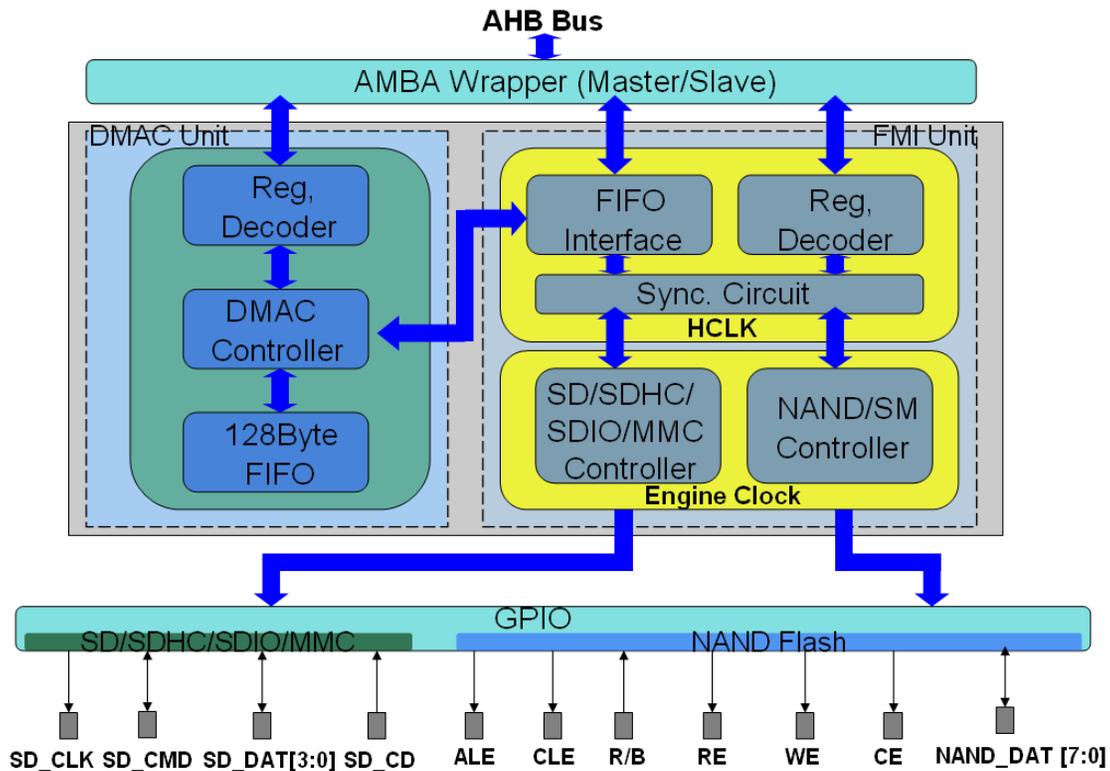


Figure 24-1 block diagram of SIC controller

24.1.2 NAND Driver and GNAND Library

The SIC library provide NAND driver API to access NAND chip directly. However, the NAND driver don't support management features for NAND chip that don't guarantee all blocks are valid. The management features include bad block management, garbage collection, and wear-leveling. We provide GNAND library to support these management features and suggest to use GNAND library before using SIC NAND driver. Please refer to document "N9H26 Non-OS GNAND Library Reference Guide" for GNAND library detail information.

24.2 Definition

24.2.1 Constant

IOCTL

Command	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

Command	Argument 0	Argument 1	Argument 2
SIC_GET_CARD_STATUS	Pointer to return value of SD card status	None	None
SIC_SET_CARD_DETECT	TRUE to enable card detect feature	None	None

Table 24-1 SIC IOCTL Definition

24.3 API function

24.3.1 SIC API

sicOpen

Synopsis

void sicOpen (void)

Description

sicOpen() will initialize the SIC and DMAC interface hardware. It configures GPIO to FMI mode, and installs ISR. This function is board dependent. It probably needs some modifications before it can work properly on your target board.

Parameter

None

Return Value

None

Example

```
/* initialize SIC to FMI (Flash Memory Interface controller) mode */
sicIoctl(SIC_SET_CLOCK, 192000, 0, 0); /* clock from PLL */
sicOpen();
```

sicClose

Synopsis

void sicClose (void)

Description

sicClose() will Close the SIC and DMAC interface hardware. It configures GPIO to close DMAC and disable ISR for SIC.

Parameter

None

Return Value

None

Example

```
sicClose();
```

sicIoctl

Synopsis

```
VOID sicIoctl (
    INT32 sicFeature,
    INT32 sicArg0,
    INT32 sicArg1,
    INT32 sicArg2
)
```

Description

sicIoctl() allows user set engine clock and callback functions, the support features and arguments listed in Table 24-1 SIC IOCTL Definition.

Parameter

sicFeature [in]

Please refer to Table 24-1 SIC IOCTL Definition.

sicArg0 [in/out]

Please refer to Table 24-1 SIC IOCTL Definition.

sicArg1 [in]

Please refer to Table 24-1 SIC IOCTL Definition.

sicArg2 [in]

Please refer to Table 24-1 SIC IOCTL Definition.

Return Value

For SIC_GET_CARD_STATUS, the card status assign to sicArg0. The value TRUE means SD card inserted, FALSE means SD card removed.

Example

Refer to the example code of sicOpen().

24.3.2 SIC / SD API

sicSdOpen

Synopsis

INT sicSdOpen (void) open SD card 0

INT sicSdOpen0 (void) open SD card 0

INT sicSdOpen1 (void) open SD card 1
 INT sicSdOpen2 (void) open SD card 2

Description

This function initializes the SD host interface and program the SD card from identify mode to stand-by mode.

Parameter

None

Return Value

>0 Total sectornumber of SD card
 Otherwise Refer error code defined in Error Code Table

Example

```
if (sicSdOpen0() <= 0)    /* Open SD port 0 */
{
    printf("Error in initializing SD card !! \n");
    /* handle error status */
}
```

sicSdClose

Synopsis

VOID sicSdClose (void) close SD card 0
 VOID sicSdClose0 (void) close SD card 0
 VOID sicSdClose1 (void) close SD card 1
 VOID sicSdClose2 (void) close SD card 2

Description

This function 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 [in]
 Sector No. to get the data from

sdSectorCount [in]
 Sector count of this access

sdTargetAddr [out]
 The address which data upload to SDRAM

Return Value

0	On success
FMI_TIMEOUT	Access timeout
FMI_NO_SD_CARD	Card removed
FMI_SD_CRC7_ERROR	Command/Response error
FMI_SD_CRC16_ERROR	Data transfer error

Example

```
#define FMI_TEST_SIZE (512*128)
__align(4096) UINT8 fmiReadBackBuffer[FMI_TEST_SIZE];
/* read 128 sectors data from SD card sector address 3000. */
status = sicSdRead(3000, FMI_TEST_SIZE/512, (unsigned int)fmiReadBackBuffer);
```

sicSdWrite

Synopsis

```
INT
sicSdWrite (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdSourceAddr
)
for SD card 0

INT
sicSdWrite0 (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdSourceAddr
)
for SD card 0

INT
sicSdWrite1 (
    INT32 sdSectorNo,
    INT32 sdSectorCount,
    INT32 sdSourceAddr
)
for SD card 1

INT
sicSdWrite2 (
    INT32 sdSectorNo,
```

```

        INT32 sdSectorCount,
        INT32 sdSourceAddr
    )                for SD card 2
    
```

Description

This function writes the data into SD card.

Parameter

```

sdSectorNo [in]
    Sector No. to get the data from
sdSectorCount [in]
    Sector count of this access
sdSourcetAddr [in]
    The address which download data from SDRAM
    
```

Return Value

0	On success
FMI_TIMEOUT	Access timeout
FMI_NO_SD_CARD	Card removed
FMI_SD_CRC7_ERROR	Command/Response error
FMI_SD_CRC_ERROR	Data transfer error

Example

```

#define FMI_TEST_SIZE (512*128)
__align(4096) UINT8 fmiFlash_Buf[FMI_TEST_SIZE];
/* write 128 sectors data to SD card sector address 3000. */
status = sicSdWrite(3000, FMI_TEST_SIZE/512, (unsigned int)fmiFlash_Buf);
    
```

24.3.3 SIC / NAND API

nandInit0

Synopsis

```

INT
nandInit0 (
    NDISK_T *NDISK_info
)                for NAND chip 0
INT
nandInit1 (
    NDISK_T *NDISK_info
    
```

) for NAND chip 1

Description

This function configures SIC register to initial DMAC and FMI to NAND mode. It also initializes the internal data structure for future use. Since different NAND chip need different parameters, nandInit0() also read the product ID from NAND chip to try to configure correct parameters for it. This function is NAND chip dependent. It probably needs some modifications before it can work properly on your target NAND chip.

Parameter

NDISK_info [out]

The internal data for NAND disk information. nandInit0() will initial it and reture to caller.

Return Value

0 Success

Otherwise Refer error code defined in Table 23-2 SDIO Error Code Table

Example

```

NDISK_T *ptMassNDisk;
NDISK_T MassNDisk;
ptMassNDisk = (NDISK_T *)&MassNDisk;
if (nandInit0(ptMassNDisk) < 0)
{
    printf("NAND initial fail !!\n");
    /* handle error status */
}
    
```

nand_ioctl

Synopsis

```

INT
nand_ioctl (
    INT param1,
    INT param2,
    INT param3,
    INT param4
)
    
```

Description

nand_ioctl() is reserved for I/O control utility for NAND. It is empty now and could support new functions in the future.

Parameter

- param1 [in]
Depend on feature setting
- param2 [in]
Depend on feature setting
- param3 [in]
Depend on feature setting
- param4 [in]
Depend on feature setting

Return Value

- 0 Success
- Otherwise Refer error code defined in Table 23-2 SDIO 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 read a page of data from NAND.

Parameter

- PBA [in]
physical block address of NAND that read data from.

page [in]

page number in PBA block that read data from.

buff [out]

the RAM address to store the reading data.

Return Value

0 Success

Otherwise Refer error code defined in Table 23-2 SDIO Error Code Table

Example

```

__align(32) UINT8 fmiFlash_Buf[PAGE_SIZE];
/* read a page of data from NAND block 5 page 10 and store at fmiFlash_Buf */
status = nandread0(5, 10, fmiFlash_Buf);
if (status < 0)
{
    /* handle error status */
}
    
```

nandpwrite0

Synopsis

```

INT
nandpwrite0 (
    INT PBA,
    INT page,
    UINT8 *buff
)
    for NAND chip 0
    
```

```

INT
nandpwrite1 (
    INT PBA,
    INT page,
    UINT8 *buff
)
    for NAND chip 1
    
```

Description

This function writes a page of data to NAND.

Parameter

PBA [in]

physical block address of NAND to write data.

page [in]

page number in PBA block to write data.

buff [in]

the RAM address to get the writing data.

Return Value

0 Success

Otherwise Refer error code defined in Table 23-2 SDIO 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 is dirty or not. Dirty page means you cannot write data to it directly. You have to erase this block first to clean it.

Parameter

PBA [in]

physical block address of NAND to check the dirty status.

page [in]

page number in PBA block to check the dirty status.

Return Value

0 Clean page that can write data directly

1 Dirty page that cannot write data directly

Example

```

/* check dirty status for NAND block 5 page 10 */
status = nand_is_page_dirty0(5, 10);
if (status == 0)
{
    printf("This page is clean !! You can write data to it directly.\n");
}
else
{
    printf("This page is dirty !! You cannot write data to it directly.\n");
}
    
```

nand_is_valid_block0

Synopsis

INT

nand_is_valid_block0 (
 INT PBA
) for NAND chip 0

INT

nand_is_valid_block1 (
 INT PBA
) for NAND chip 1

Description

This function checks the redundancy area of the NAND block and return the valid status to indicate whether a block is valid or not. Valid block page means you can write data to it directly or indirectly (maybe need to erase block first). You cannot write data into an invalid block always since it could be a bad block.

Parameter

PBA [in]

physical block address of NAND to check the valid status.

Return Value

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

Example

```

/* check valid status for NAND block 5 */
status = nand_is_valid_block0(5);
if (status == 0)
{
    printf("This block is valid !! You can write data to it directly or
    indirectly.\n");
}
else
{
    printf("This block is invalid !! You cannot write data to it always.\n");
}
    
```

nand_block_erase0

Synopsis

```

INT
nand_block_erase0 (
    INT PBA
)                for NAND chip 0
INT
nand_block_erase1(
    INT PBA
)                for NAND chip 1
    
```

Description

This function erases a block. You should call this API first if you want to write data into a dirty page.

Parameter

PBA [in]
 physical block address of NAND to erase.

Return Value

- 0 Erase block successfully
- Otherwise Refer error code defined in Table 23-2 SDIO Error Code Table

Example

```

/* erase NAND block 5 */
status = nand_block_erase0(5);
if (status == 0)
{
    printf("This block is erased !!\n");
}
else
{
    printf("This block erase fail !!\n");
}
    
```

nand_chip_erase0

Synopsis

INT

nand_chip_erase0(void) for NAND chip 0

INT

nand_chip_erase1(void) for NAND chip 1

Description

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

Parameter

None

Return Value

0 Erase chip successfully

Otherwise Refer error code defined in Table 23-2 SDIO 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");
}
    
```

24.4 SIC 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	NoSD card insert
FMI_ERR_DEVICE	0xFFFF0111	Unknown device type
FMI_SD_INIT_TIMEOUT	0xFFFF0112	SD Initial time out
FMI_SD_SELECT_ERROR	0xFFFF0113	Select card from identify mode to stand-by mode error
FMI_SD_WRITE_PROTECT	0xFFFF0114	SD Write Protect
FMI_SD_INIT_ERROR	0xFFFF0115	SD Card initial and identify error
FMI_SD_CRC7_ERROR	0xFFFF0116	Command/Response error
FMI_SD_CRC16_ERROR	0xFFFF0117	Data reading error
FMI_SD_CRC_ERROR	0xFFFF0118	Data writing error
FMI_SD_CMD8_ERROR	0xFFFF0119	SD command 8 error
Error Code for NAND		
FMI_SM_INIT_ERROR	0xFFFF0120	NAND/SM card initial error
FMI_SM_RB_ERR	0xFFFF0121	NAND don't become ready from busy status
FMI_SM_STATE_ERROR	0xFFFF0122	NAND return fail for write command
FMI_SM_ECC_ERROR	0xFFFF0123	Read data error and uncorrectable by ECC
FMI_SM_STATUS_ERR	0xFFFF0124	NAND return fail for erase command
FMI_SM_ID_ERR	0xFFFF0125	NAND chip ID don't supported
FMI_SM_INVALID_BLOCK	0xFFFF0126	NAND block is invalid to erase or write
FMI_SM_MARK_BAD_BLOCK_ERR	0xFFFF0127	Fail to mark a block to bad
FMI_SM_REGION_PROTECT_ERR	0xFFFF0128	NAND return fail for write command because of region protect

Table 24-2 SIC Error Code Table

25 SPI Library

25.1 SPI Library Overview

This library provides APIs for programmers to access SPI device connecting with N9H26SPI interfaces. The SPI library will get the APB clock frequency from system library, application must set the CPU clock before using SPI library.

25.2 API function

spiOpen

Synopsis

```
INT32
spiOpen (
    SPI_INFO_T *pInfo
)
```

Description

This function initializes the SPI interface.

Parameter

pInfo[in]
 SPI property information

Return Value

= 0 Success
 < 0 Fail

Example

```
spiOpen();
```

spiClose

Synopsis

```
INT32
spiClose (
    UINT8 u8Port
)
```

Description

This function disables SPI engine clock.

Parameter

u8Port [in]
 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 [in]
 Select SPI0 (0) or SPI1 (1)

spiFeature [in]
 SPI_SET_CLOCK

spiArg0 [in]
 APB clock by MHz

spiArg1 [in]
 Device output clock by kHz

Return Value

0success

Example

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

spiEnable

Synopsis

```
INT
spiEnable (
    INT32 spiPort
)
```

Description

This function will active the SPI interface to access device (active CS#).

Parameter

```
spiPort [in]
    Select SPI0 (0) or SPI1 (1)
```

Return Value

0success

Example

```
spiEnable(0);
```

spiDisable

Synopsis

```
INT
spiDisable (
    INT32 spiPort
)
```

Description

This function will inactive the SPI interface (inactive CS#).

Parameter

```
spiPort [in]
    Select SPI0 (0) or SPI1 (1)
```

Return Value

0 success

Example

```
spiDisable(0);
```

spiRead

Synopsis

```

INT
spiRead (
    INT port,
    INT RxBitLen,
    INT len,
    CHAR *pDst
)
    
```

Description

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

Parameter

```

port [in]
    select SPI0 (0) or SPI1 (1)

RxBitLen [in]
    set the receive bit length. SPI_8BIT, SPI_16BIT, SPI_32BIT

len [in]
    data count. SPI_8BIT is byte count; SPI_16BIT is half-word count;
    SPI_32BIT is word count.

pDst [out]
    Read back destination
    
```

Return Value

```

0      Success
    
```

Example

```

/* read 1 byte data from SPI device */
spiRead(0, SPI_8BIT, 1, (CHAR *)&rdata);
    
```

spiWrite

Synopsis

```

INT
spiWrite (
    INT port,
    INT TxBitLen,
    INT len,
    CHAR *pSrc
)
    
```

Description

This function is used to write the data to the SPI interface.

Parameter

port [in]

select SPI0 (0) or SPI1 (1)

TxBitLen [in]

set the transmit bit length. SPI_8BIT, SPI_16BIT, SPI_32BIT

len [in]

data count. SPI_8BIT is byte count; SPI_16BIT is half-word count; SPI_32BIT is word count

pSrc [out]

data source address

Return Value

0 Success

Example

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

spiEnableInt

Synopsis

```
VOID spiEnableInt (
    UINT8 u8Port
)
```

Description

This function is used to enable the SPI interrupt.

Parameter

u8Port [in]

Select SPI0 (0) or SPI1 (1)

Return Value

None

Example

```
spiEnableInt(0);
```

spiDisableInt

Synopsis

```
VOID spiDisableInt (
    UINT8 u8Port
)
```

Description

This function is used to disable the SPI interrupt.

Parameter

u8Port [in]
 Select SPI0 (0) or SPI1 (1)

Return Value

None

Example

```
spiDisableInt(0);
```

spiInstallCallback

Synopsis

```
ERRCODE
spiInstallCallback (
    UINT8 u8Port,
    PFN_DRVSPI_CALLBACK pfncallback,
    PFN_DRVSPI_CALLBACK *pfnOldcallback
)
```

Description

This function is used to install the specified SPI interrupt call back function.

Parameter

u8Port [in]
 Select SPI0 (0) or SPI1 (1)

pfncallback [in]
 The callbackfunction pointer for specified SPI port.

pfnOldcallback [out]
 The previous callbackfunction pointer for specified SPI port.

Return Value

0 Success

Example

```
spiInstallCallBack (0, SPIIRQHandler, &pfnOldcallback);
```

spiSetGo

Synopsis

```
VOID spiSetGo (
    UINT8 u8Port
)
```

Description

This function is used to set GO_BUSY bit to trigger the SPI port.

Parameter

u8Port [in]
 Select SPI0 (0) or SPI1 (1)

Return Value

None

Example

```
spiSetGo(0);
```

spiSetByteEndin

Synopsis

```
VOID spiSetByteEndin (
    UINT8 u8Port,
    E_DRVSPi_OPERATION eOP
)
```

Description

This function is used to enable or disable the byte endin.

Parameter

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

Return Value

None

Example

```
spiSetByteEndin(0, eDRV/SPI_DISABLE);
```

spiSSEnable

Synopsis

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

Description

This function set transfer timing and active the SPI interface to access device (active CS#).

Parameter

```
spiPort [in]
    Select SPI0 (0) or SPI1 (1)
SSPin [in]
    Select SS0 or SS1
ClockMode [in]
    Set transfer timing
```

Return Value

```
0    success
```

Example

```
spiSSEnable(0, 0, 3);
```

spiSSDisable

Synopsis

```
INT
spiSSDisable (
    UINT32 spiPort,
    UINT32 SSPin
)
```

Description

This function will inactive the SPI interface (inactive CS#).

Parameter

spiPort [in]
 Select SPI0 (0) or SPI1 (1)
 SSPin [in]
 Select SS0 or SS1

Return Value

0 success

Example

```
spiSSDisable(0, 0);
```

spiTransfer

Synopsis

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

Description

This function is used to transfer data through the SPI interface.

Parameter

port [in]
 select SPI0 (0) or SPI1 (1)
 TxBitLen [in]
 set the transmit bit length. SPI_8BIT, SPI_16BIT, SPI_32BIT
 len [in]
 data count. SPI_8BIT is byte count; SPI_16BIT is half-word count;
 SPI_32BIT is word count
 RxBuf [out]
 Read back destination
 TxBuf [in]

data source address

Return Value

0 Success

Example

```
/* transfer 1 half-word through SPI interface */
wdata = 0x80ff;
spiTransfer(0, SPI_16BIT, 1, (PUINT8)&rdata, (PUINT8)&wdata);
```

spilsBusy

Synopsis

```
BOOL
spilsBusy (
    UINT8 u8Port
)
```

Description

This function is used to check spi bus state.

Parameter

u8Port [in]
 Select SPI0 (0) or SPI1 (1)

Return Value

0 SPI bus is in idle state
 1 SPI bus is in busy state

Example

```
spilsBusy(0);
```

26 SPI SecureIC Library

26.1 SPI SecureIC Library Overview

This library provides APIs for Winbond W74M RPMC operation.

26.2 API function

RPMC_ReadJEDECID

Synopsis

```
INT32
RPMC_ReadJEDECID (
    PUINT8 data
)
```

Description

This function is used to read SPI Flash JEDEC ID.

Parameter

data [out]
Point to store JEDEC ID.

Return Value

Successful

Example

```
UINT8 u8JID[3];
if ((RPMC_ReadJEDECID(u8JID)) == -1)
{
    sysprintf("read id error !!\n");
    return -1;
}
```

RPMC_ReadUID

Synopsis

```
INT32
RPMC_ReadUID (
    PUINT8 data
)
```

Description

This function is used to read SPI Flash Unique ID.

Parameter

data [out]
 Point to store Unique ID.

Return Value

Successful

Example

```

UINT8 u8UID [8];
if ((RPMC_ReadUID (u8UID)) == -1)
{
    sysprintf("read id error !!\n");
    return -1;
}
    
```

RPMC_ReadCounterData

Synopsis

unsigned int
 RPMC_ReadCounterData (void)

Description

This function is used to read RPMC counter number.

Parameter

None

Return Value

RPMC counter number

Example

```

RPMC_counter = RPMC_ReadCounterData();
    
```

RPMC_ReadRPMCstatus

Synopsis

unsigned int
 RPMC_ReadRPMCstatus (
 unsigned int checkall
)

Description

This function is used to read RPMC status.

Parameter

checkall [in]

- 0 Only read out RPMC status
- 1 Read out counter data, tag, signature information

Return Value

Depend on parameter.

Example

```
RPMC_counter = RPMC_ReadRPMCstatus (0);
```

RPMC_RPMC_ReqCounter

Synopsis

```
void RPMC_ReqCounter (
    unsigned int cadr,
    unsigned char *hmackey,
    unsigned char *input_tag
)
```

Description

This function is used to request RPMC counter .data.

Parameter

cadr [in]

Selected Counter address (1~4).

hmackey [in]

32 byte HMACKEY which is generated by RPMC_UpHMACkey()

input_tag [in]

12 byte input Tag data, which can be time stamp, serial number or random number. These data would repeat after success RPMC_ReqCounter() operation

Return Value

None

Example

None

RPMC_WrRootKey

Synopsis

```
unsigned int
```

```
RPMC_WrRootKey (
    unsigned int cadr,
    unsigned char *rootkey
)
```

Description

This function is used to write RPMC Root Key.

Parameter

```
cadr [in]
    Selected Counter address (1~4).
rootkey [in]
    32 byte rootkey information
```

Return Value

```
0x80      Write rootkey success
Others    Write rootkey fail
```

Example

```
/* initial Rootkey, use first rootkey/counter pair */
RPMCStatus = RPMC_WrRootKey(KEY_INDEX, ROOTKey);
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 is used to update RPMC HAC Key. This function should call in every Gneiss power on

Parameter

cadr [in]

Selected Counter address (1~4).

rootkey [in]

Rootkey use for generate HMAC key

hmac4 [in]

4 byte input hmac message data, which can be time stamp, serial number or random number.

hmackey [in]

32 byte HMACKEY, which would be use for increase/request counter after RPMC_UpHMACkey() operation success

Return Value

0x80 Update HMACkey success

Others Update HMACkey 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 RPMC counter number.

Parameter

cadr [in]

Selected Counter address (1~4).

hmackey [in]

32 byte HMACKEY which is generated by RPMC_UpHMACkey()

input_tag [in]

12 byte input Tag data, which can be time stamp, serial number or random number. These data would repeat after success RPMC_ReqCounter() operation

Return Value

0x80 Increase counter success

Others Increase counter 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 TO RPMC Challenge signature and it is the main security operation

Parameter

```

cadr [in]
    Selected Counter address (1~4).

hmackey [in]
    32 byte HMACKEY which is generated by RPMC_UpHMACkey()

input_tag [in]
    12 byte input Tag data, which can be time stamp, serial number or random number.

```

Return Value

0	Signature match
Others	Signature mismatch

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;
    }
}

```

27 SPI to UART Library

27.1 SPI to UART Library Overview

This library provides APIs to create two virtual UART ports via SPI1 interface connects to Mini58.

27.2 API function

vu_OpenUART

Synopsis

```
void vu_OpenUART (  
    UINT8 UART_port  
)
```

Description

This function is used to create specify virtual UART port.

Parameter

UART_port [in]

UART port number. 0 means virtual UART port 0. 1 means virtual UART port 1. 0xFF means virtual UART port 0 and port 1.

Return Value

None

Example

```
/* Open virtual UART port 0 */  
vu_OpenUART(UART_PORT0);
```

vu_ResetUART

Synopsis

```
void vu_ResetUART (  
    UINT8 UART_port  
)
```

Description

This function is used to clear Tx/Rx buffer.

Parameter

UART_port [in]

UART port number.

Return Value

None

Example

```
vu_ResetUART(UART_PORT0);
```

vu_UARTRead

Synopsis

```
int vu_UARTRead (
    UINT8 UART_port,
    INT32 Max,
    UINT8 *pDst
)
```

Description

The function is used to read data from Rx buffer and the data will be stored in pDst.

Parameter

UART_port [in]
 UART port number.

Max [in]
 The read byte number of data.

pDst [out]
 The buffer to receive the data.

Return Value

Receive byte count.

Example

```
/* Read one byte from virtual UART port 0 */
vu_UARTRead(UART_PORT0, 1, pDst);
```

vu_SetBaudRate

Synopsis

```
void vu_SetBaudRate (
    UINT8 UART_port,
    UINT32 baudrate
)
```

Description

This function is used to set baudrate.

Parameter

- UART_port [in]
UART port number.
- baudrate [in]
The baudrate of UART port.

Return Value

None

Example

```
vu_SetBaudRate(UART_PORT0,115200);
```

vu_UARTWrite

Synopsis

```
int vu_UARTWrite (
    UINT8 UART_port,
    unsigned char *pSrc,
    INT32 len
)
```

Description

The function is used to write data into TX buffer to transmit data by UART.

Parameter

- UART_port [in]
UART port number.
- pSrc [in]
The buffer to send the data to UART.
- len [in]
The byte number of data.

Return Value

Transfer byte count.

Example

```
/* Write one byte to virtual UART port 0 */
vu_UARTWrite(UART_PORT0, pSrc, 1);
```

vu_ClearBuf**Synopsis**

```
void vu_ClearBuf (  
    UINT8 UART_port  
)
```

Description

This function is used to clear Tx/Rx buffer.

Parameter

UART_port [in]
UART port number.

Return Value

None

Example

```
vu_ClearBuf(UART_PORT0);
```

vu_CloseUART**Synopsis**

```
void vu_CloseUART (  
    UINT8 UART_port  
)
```

Description

This function is used to close specify virtual UART port.

Parameter

UART_port [in]
UART port number.

Return Value

None

Example

```
vu_CloseUART(UART_PORT0);
```

vu_GetStatus**Synopsis**

```
int vu_GetStatus (void)
```

Description

This function is used to get the status of send data and receive data command.

Parameter

None

Return Value

- 0 Command status is in ready state
- 1 Command status is in busy state

Example

```
int Status;
Status = vu_GetStatus();
```

vu_GetNotification

Synopsis

char vu_GetNotification (void)

Description

This function is used to get notification status.

Parameter

None

Return Value

- Bit0 : 0 means UART port 0 Rx buffer is empty, 1 means Rx buffer has data
- Bit1 : 0 means UART port 0 Tx buffer has data, 1 means Tx buffer is empty
- Bit2 : 0 means UART port 1 Rx buffer is empty, 1 means Rx buffer has data
- Bit3 : 0 means UART port 1 Tx buffer has data, 1 means Tx buffer is empty

Example

```
char Status;
Status = vu_GetNotification();
```

vu_GetRXAvailLen

Synopsis

```
int vu_GetRXAvailLen (
    UINT8 UART_port
)
```

Description

This function is used to get receive byte count in Rx buffer.

Parameter

UART_port [in]

UART port number.

Return Value

Receive byte count.

Example

```
int Count;  
Count = vu_GetRXAvailLen(UART_PORT0);
```

vu_GetTXFreeLen**Synopsis**

```
int vu_GetTXFreeLen (  
    UINT8 UART_port  
)
```

Description

This function is used to get free byte count in Tx buffer.

Parameter

UART_port [in]
UART port number.

Return Value

Free byte count.

Example

```
int Count;  
Count = vu_GetTXFreeLen(UART_PORT0);
```

28 SPU Library

28.1 SPU Library Overview

This library provides APIs for programmers play PCM audio data from SPU engine. For playing audio this library also provides 10-band equalizer APIs. SPU engine only plays audio, no record function is included.

28.2 Definition

28.2.1 Constant

IOCTL

Command	Argument 0	Argument 1	comment
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
SPU_IOCTL_SET_FRAG_SIZE	Fragment size	Not used	Set the fragment size

Table 28-1 SPU IOCTL Definition

28.3 API function

spuOpen

Synopsis

```
VOID spuOpen (
    UINT32 u32SampleRate
)
```

Description

This function will set the audio clock, play buffer address and install its interrupt.

Parameter

u32SampleRate [in]

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 [in]
Play call back function pointer
- data [in]
Source PCM audio data pointer

Return Value

None

Example

```
int playCallBack(UINT8 * pu8Buffer)
{
    ...
}
spuStartPlay((PFN_DRVSPU_CB_FUNC *) playCallBack, (UINT8 *)SPU_SOURCE);
```

spuStopPlay

Synopsis

```
VOID spuStopPlay (void)
```

Description

Stop play.

Parameter

None

Return Value

None

Example

```
spuStopPlay ();
```

spuClose

Synopsis

VOID spuClose (void)

Description

This function disables SPU engine.

Parameter

None

Return Value

None

Example

```
spuClose ();
```

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 Table 28-1 SPU IOCTL Definition.

Parameter

cmd [in]

Command

arg0 [in/out]

First argument of the command

arg1 [in]

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 be used before calling spuStartPlay().

Parameter

level [in]
 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 [in]
Equalizer band setting

eEqGain [in]
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 ();
```

29 System Library

29.1 System Library Overview

The System library provides a set of APIs to control on-chip functions such as Timers, UARTs, AIC, Cache and power management. With these APIs, user can quickly create a test program to run on demo board or evaluation board.

29.2 Definition

29.2.1 Constant

Timer Channel

Channel name	Value	Description
TIMER0	0	Timer 0
TIMER1	1	Timer 1
TIMER2	2	Timer 2
TIMER3	3	Timer 3
WDTIMER	4	Watch Dog Timer

Table 29-1 Timer Channel Definition

Timer Mode

Channel name	Value	Description
ONE_SHOT_MODE	0	One shot mode.
PERIODIC_MODE	1	Periodic mode.
TOGGLE_MODE	2	Toggle mode.
UNINTERRUPT_MODE	3	Uninterrupt mode.

Table 29-2 Timer Mode Definition

Watch-Dog Interval Select

Channel name	Value	Description
WDT_14BITS	0	Timer 0
WDT_16BITS	1	Timer 1
WDT_18BITS	2	Timer 2
WDT_20BITS	3	Timer 3

Table 29-3 Watch-Dog Interval Select Definition

Watch Dog timer interval

nWdtInterval	Interrupt Timeout	Reset Timeout	Real Time Interval
WDT_INTERVAL_0	2 ¹⁴ clocks	2 ¹⁴ + 1024 clocks	0.371 sec.
WDT_INTERVAL_1	2 ¹⁶ clocks	2 ¹⁶ + 1024 clocks	1.419 sec.
WDT_INTERVAL_2	2 ¹⁸ clocks	2 ¹⁸ + 1024 clocks	5.614 sec.
WDT_INTERVAL_3	2 ²⁰ clocks	2 ²⁰ + 1024 clocks	22.391 sec.

Table 29-4 Watch Dog timer interval

UART Port

Port Name	Value	Description
WB_UART_0	0	UART 0 – High Speed UART
WB_UART_1	1	UART 1 – Normal Speed UART

Table 29-5 UART Port Definition

UART Data Bits

Data Bits	Value	Description
WB_DATA_BITS_5	0	5 Data Bits
WB_DATA_BITS_6	1	6 Data Bits
WB_DATA_BITS_7	2	7 Data Bits
WB_DATA_BITS_8	3	8 Data Bits

Table 29-6 UART Data Bits Definition

UART Stop Bits

Stop Bits	Value	Description
WB_STOP_BITS_1	0x0	1 Stop Bit
WB_STOP_BITS_2	0x4	2 Stop Bits

Table 29-7 UART Stop Bits Definition

UART Parity Bits

Stop Bits	Value	Description
WB_PARITY_NONE	0x0	Non Parity Bit
WB_PARITY_ODD	0x8	Odd Parity Bit
WB_PARITY_EVEN	0x18	Even Parity Bit

Table 29-8 UART Parity Bits Definition

UART FIFO Threshold

FIFO	Value	Description
LEVEL_1_BYTE	0x0	1 Byte FIFO
LEVEL_4_BYTES	0x1	4 Bytes FIFO
LEVEL_8_BYTES	0x2	8 Bytes FIFO
LEVEL_14_BYTES	0x3	14 Bytes FIFO
LEVEL_30_BYTE	0x4	30 Bytes FIFO (High Speed UART Only)
LEVEL_46_BYTES	0x5	46 Bytes FIFO (High Speed UART Only)
LEVEL_62_BYTES	0x6	62 Bytes FIFO (High Speed UART Only)

Table 29-9 UART FIFO Threshold Definition

UART Interrupt type

Interrupt type	Value	Description
UART_INT_RDA	0x0	UART Data Ready
UART_INT_RDTO	0x1	UART Time out
UART_INT_NONE	0xFF	Not to enable UART

Table 29-10 UART Interrupt type Definition

Interrupt No.

AIC Interrupt No	Value	Description
IRQ_WDT	1	Watch Dog Timer Interrupt
IRQ_EXTINT0	2	GPIO Group 0 interrupt
IRQ_EXTINT1	3	GPIO Group 1 interrupt
IRQ_EXTINT2	4	GPIO Group 2 interrupt
IRQ_EXTINT3	5	GPIO Group 3 interrupt
IRQ_IPSEC	6	AES Interrupt
IRQ_SPU	7	SPU Interrupt
IRQ_I2S	8	I2S Interrupt
IRQ_VPOST	9	VPOST Interrupt
IRQ_VIN	10	Video In 0 Interrupt
IRQ_MDCT	11	MDCT Interrupt
IRQ_BLT	12	BLT Interrupt
IRQ_VPE	13	VPE Interrupt
IRQ_HUART	14	High Speed UART Interrupt
IRQ_TMR0	15	Timer 0 Interrupt
IRQ_TMR1	16	Timer 1 Interrupt

AIC Interrupt No	Value	Description
IRQ_UDC	17	USB Device Controller Interrupt
IRQ_SIC	18	Storage Interrupt Controller Interrupt
IRQ_SDIO	19	Secure Digital Input / Output Control Interrupt
IRQ_UHC	20	USB Host Controller Interrupt
IRQ_EHCI	21	Enhanced Host Controller Interface Interrupt
IRQ_OHCI	22	Host Controller Interface Interrupt
IRQ_EDMA0	23	Enhanced DMA 0 Interrupt
IRQ_EDMA1	24	Enhanced DMA 1 Interrupt
IRQ_SPIMS0	25	SPI Master / Slave 0 Interrupt
IRQ_SPIMS1	26	SPI Master / Slave 1 Interrupt
IRQ_AUDIO	27	Audio Record Interrupt
IRQ_TOUCH	28	Touch Controller Interrupt
IRQ_RTC	29	RTC Interrupt
IRQ_UART	30	UART Interrupt
IRQ_PWM	31	PWM Interrupt
IRQ_JPG	32	JPEG Codec Interrupt
IRQ_VDE	33	H264 Decode Interrupt
IRQ_VEN	34	H264 Encode Interrupt
IRQ_SDIC	35	SDIC Interrupt
IRQ_EMCTX	36	EMC TX Interrupt
IRQ_EMCRX	37	EMC RX Interrupt
IRQ_I2C	38	I2C Interrupt
IRQ_KPI	39	Keypad Interrupt
IRQ_RSC	40	RS Codec Interrupt
IRQ_VTB	41	Convolution / Viterbi Codec Interrupt
IRQ_ROT	42	Convolution / Viterbi Codec Interrupt
IRQ_PWR	43	System Wake-Up Interrupt
IRQ_LVD	44	Low Voltage Detector Interrupt
IRQ_VIN1	45	Video In 1 Interrupt
IRQ_TMR2	46	Timer 2 Interrupt
IRQ_TMR3	47	Timer 3 Interrupt

Table 29-11 Interrupt No. Definition

Interrupt Exception Type

Exception Type	Value	Description
WB_SWI	0	Software Interrupt
WB_D_ABORT	1	Data Abort Interrupt
WB_I_ABORT	2	Instruction Abort Interrupt
WB_UNDEFINE	3	Undefined Interrupt

Table 29-12 Interrupt Exception Type Definition

Interrupt Priority

Interrupt Priority	Value	Description
FIQ_LEVEL_0	0	Highest Priority
IRQ_LEVEL_1	1	Level 1 Priority
IRQ_LEVEL_2	2	Level 2 Priority
IRQ_LEVEL_3	3	Level 3 Priority
IRQ_LEVEL_4	4	Level 4 Priority
IRQ_LEVEL_5	5	Level 5 Priority
IRQ_LEVEL_6	6	Level 6 Priority
IRQ_LEVEL_7	7	Lowest Priotity

Table 29-13 Interrupt Priority Definition

Local Interrupt Type

Local Interrupt Type	Value	Description
ENABLE_IRQ	0x7F	Enable ARM Core's IRQ bit
ENABLE_FIQ	0xBF	Enable ARM Core's FIQ bit
ENABLE_FIQ_IRQ	0x3F	Enable ARM core's FIQ and IRQ bit
DISABLE_IRQ	0x80	Disable ARM Core's IRQ bit
DISABLE_FIQ	0x40	Disable ARM Core's FIQ bit
DISABLE_FIQ_IRQ	0xC0	Disable ARM core's FIQ and IRQ bit

Table 29-14 Local Interrupt Type Definition

Interrupt Trigger Type

Interrupt Trigger Type	Value	Description
LOW_LEVEL_SENSITIVE	0x00	Low Level Trigger Type
HIGH_LEVEL_SENSITIVE	0x01	High Level Trigger Type
NEGATIVE_EDGE_TRIGGER	0x02	Falling Edge Trigger Type
POSITIVE_EDGE_TRIGGER	0x03	Rising Edge Trigger Type

Table 29-15 Interrupt Trigger Type Definition

Wakeup Channels.

Wake up channel	Value	Description
WE_EMAC	0x1	Wake up by magic packet
WE_UHC20	0x2	Wake up by device attaced/deattached
WE_GPIO	0x100	Wake up by GPIO level change
WE_RTC	0x200	Wake up by RTC
WE_UART	0x800	Wake up by UART (RTS pin)
WE_UDC	0x1000	Wake up by attached/detached from USB host
WE_UHC	0x2000	Wake up by device attached/detached
WE_ADC	0x4000	Wake up by touch panel touch
WE_KPI	0x8000	Wake up by KPI pressing

Table 29-16 Wakeup Channels Definition

29.2.2 Structure

DateTime_T Structure

Field name	Data Type	Description
year	UINT32	year
mon	UINT32	mon
day	UINT32	day
hour	UINT32	hour
min	UINT32	min
sec	UINT32	sec

Table 29-17 DateTime_T Structure Definition

WB_UART Structure

Field name	Data Type	Description
uart_no	UINT32	UART port to be initialized
freq	UINT32	UART reference clock. Default is 15MHz. If user have different reference clock, used this parameter to change it
baud_rate	UINT32	COM port baud rate. The range is from 9600 to 230400
data_bits	UINT32	data bits
stop_bits	UINT32	stop bits
parity	UINT32	parity check

Field name	Data Type	Description
rx_trigger_level	UINT32	trigger level

Table 29-18 WB_UART Structure Definition

PFN_SYS_UART_CALLBACK Structure

Field name	Data Type	Description
u8Buf	UINT8 *	Received data buffer pointer
u32Len	UINT32	Received data length

Table 29-19 PFN_SYS_UART_CALLBACK Structure Definition

29.3 API function

29.3.1 Timer API functions

sysSetTimerEvent

Synopsis

```

INT32
sysSetTimerEvent (
    UINT32 nTimeNo,
    UINT32 nTimeTick,
    PVOID pvFun
)
    
```

Description

This function is used to set the event of selected timer. nTimeNo is used to select timer 0 to timer 3. The event function which pointed by pvFun shall be executed after nTimeTick system timer tick. The function is useless for WDTIMER.

Parameter

- nTimeNo [in]
Timer 0 ~ timer 3. Please refer Table 29-1 Timer Channel Definition.
- nTimeTick [in]
Tick count before event executed
- pvFun [in]
Event function pointer

Return Value

Eventnumber. Please remember the event number if you want to uninstall the timer event.

Example

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

sysClearTimerEvent

Synopsis

```
VOID sysClearTimerEvent (
    UINT32 nTimeNo,
    UINT32 uTimeEventNo
)
```

Description

This function is used to clear the event of selected timer. nTimeNo is used to select timer 0 ~ timer 3. The event function which indicated by uTimeEventNo shall be cleared. The function is useless for WDTIMER.

Parameter

- nTimeNo [in]
 - TIMER0, TIMER1
- uTimeEventNo [in]
 - Event number which want to clear. The event number is the return value of function-sysSetTimerEvent().

Return value

None

Example

```
/* clear event NO 5*/
sysClearTimerEvent (TIMER0, 5);
```

sysClearWatchDogTimerCount

Synopsis

```
VOID sysClearWatchDogTimerCount (VOID)
```

Description

This function is used to clear watch dog timer reset counter. When interrupt occurred, the system will be reset after 1024 clock cycles. Clear the timer reset counter, the system will not be reset.

Parameter

None

Return value

None

Example

```
sysClearWatchDogTimerCount();
```

sysClearWatchDogTimerInterruptStatus

Synopsis

VOID sysClearWatchDogTimerInterruptStatus (VOID)

Description

This function is used to clear watch dog timer interrupt status. When interrupt occurred, the watch dog timer interrupt flag will be set. Clear this flag, the interrupt will occur again.

Parameter

None

Return value

None

Example

```
sysClearWatchDogTimerInterruptStatus();
```

sysDelay

Synopsis

```
VOID sysDelay (
    UINT32 uTicks
)
```

Description

This function is used to delay a specific period. uTicks is the length of delay time which unit is ten milliseconds. Please notice that the delay period has an extent of error which is less than ten milliseconds.

Parameter

uTicks [in]
 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 [out]

structure pointer which contains the following information. Please refer to Table 29-17 DateTime_T Structure Definition.

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 [in]
TIMER0, TIMER1

Return value

The current selected timer tick count.

Example

```
/* Get current timer 0 tick count */
UINT32 btime;
btime = sysGetTicks(TIMER0);
```

sysInstallWatchDogTimerISR

Synopsis

```
PVOID
sysInstallWatchDogTimerISR (
    INT32 nIntTypeLevel,
    PVOID pvNewISR
)
```

Description

This function is used to set up own watch dog timer interrupt service routine. nIntTypeLevel is select interrupt to be FIQ or IRQ, and level group 0 ~ 7. pvNewISR is the own interrupt service routine's pointer.

Parameter

nIntTypeLevel [in]
FIQ_LEVEL_0, IRQ_LEVEL_1 ~ IRQ_LEVEL_7

pvNewISR [in]

the pointer of watch dog timer interrupt service routine

Return value

a pointer which point to old ISR

Example

```
/* Set watch dog timer interrupt to be IRQ and group level 1 */
PVOID oldVect;
oldVect = sysInstallWatchDogTimerISR(IRQ_LEVEL_1, myWatchDogISR);
```

sysResetTicks

Synopsis

```
INT32
sysResetTicks (
    INT32 nTimeNo
)
```

Description

This function used to reset Timer 0 or Timer 1’s global tick counter. The function is useless for WDTIMER.

Parameter

nTimeNo [in]
 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 [in]

structure which contains the following information. Please refer to Table 29-17 DateTime_T Structure Definition.

Return value

None

Example

```
/* set local time*/
DateTime_T  TimeInfo;
TimeInfo.year = 2006;
TimeInfo.mon = 6;
TimeInfo.day = 12
TimeInfo.hour = 9;
TimeInfo.min = 0;
TimeInfo.sec = 30;
sysSetLocalTime(TimeInfo);
```

sysSetTimerReferenceClock

Synopsis

INT32

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

Description

This function used to set the timer’s reference clock. The default reference clock is system clock (15MHz). The function is useless for WDTIMER.

Parameter

nTimeNo [in]

TIMER0, TIMER1

uClockRate [in]

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 [in]

Please refer to Table 29-4 Watch Dog timer interval.

Return value

Successful

Example

```
/* Set watch dog timer interval to WDT_INTERVAL_0 */
INT32 status;
status = sysSetWatchDogTimerInterval(WDT_INTERVAL_0);
```

sysStartTimer

Synopsis

```
INT32
sysStartTimer (
    INT32 nTimeNo,
    UINT32 uTicksPerSecond,
    INT32 nOpMode
)
```

Description

nTimeNo is used to select Timer 0, Timer 1, Timer 2, Timer3 or What-dog Timer. Because of the chip's timer has four operation modes, the nOpMode is used to set the operation mode. uTicksPerSecond indicates that how many ticks per second.

Parameter

nTimeNo [in]

TIMER0, TIMER1, TIMER2, TIMER3 or WDTIMER.

nTickPerSecond [in]

Tick number per second. It is useless if WDTIMER

nOpMode [in]

Working mode. Please refer the **Error! Reference source not found..** It is useless for WDTIMER.

Return Value

Successful

Example

```
/* Start the timer 1, and set it to periodic mode and 100 ticks per second */
INT32 status;
status = sysStartTimer(TIMER1, 100, PERIODIC_MODE);
```

sysStopTimer

Synopsis

```
INT32
sysStopTimer (
    INT32 nTimeNo
)
```

Description

sysStopTimer will stop the specified timer channel. nTimeNo is used to select timer 0 ~ timer 3 or Watch Dog Timer. After disabling timer, this function will restore the interrupt service routine.

Parameter

nTimeNo [in]

TIMER0, TIMER1, TIMER2, TIMER3 or WDTIMER.

Return Value

Successful

Example

```
/* Stop the timer 1 */
INT32 status;
status = sysStopTimer(TIMER1);
```

sysUpdateTickCount

Synopsis

```

INT32
sysUpdateTickCount (
    INT32 nTimeNo,
    UINT32 uCount
)
    
```

Description

This function used to update Timer 0 or Timer 1’s global tick counter.

Parameter

```

nTimeNo [in]
    TIMER0, TIMER1
uCount [in]
    tick counter value
    
```

Return Value

Successful

Example

```

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

29.3.2 UART API functions

sysGetChar

Synopsis

```

INT8
sysGetChar (VOID)
    
```

Description

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

Parameter

None

Return Value

A character from UART

Example

```

/* get user's input*/
CHAR cUserInput;
    
```

```
cUserInput = sysGetChar();
```

sysInitializeUART

Synopsis

```
INT32
sysInitializeUART (
    WB_UART *uart
)
```

Description

WB_UART is the device initialization structure.

Parameter

uart [in]

Please refer to Table 29-18 WB_UART Structure Definition. Normal speed UART means the baud rate less or equal to 115200 bps. And high speed UART means the baud rate up to 921600 bps.

Return Value

Successful	
WB_INVALID_PARITY	Invalid parity
WB_INVALID_DATA_BITS	Invalid data bits
WB_INVALID_STOP_BITS	Invalid stop bits
WB_INVALID_BAUD	Invalid baud rate

Example

```
WB_UART_T uart;
uart.uart_no = WB_UART_1 ;
uart.uiFreq = APB_SYSTEM_CLOCK;
uart.uiBaudrate = 115200;
uart.uiDataBits = WB_DATA_BITS_8;
uart.uiStopBits = WB_STOP_BITS_1;
uart.uiParity = WB_PARITY_NONE;
uart.uiRxTriggerLevel = LEVEL_1_BYTE;
sysInitializeUART(&uart);  WB_UART_T uart;
```

sysPrintf

Synopsis

```
VOID sysPrintf (
    PINT8 pcStr, ...
```

)

Description

The function sends the specified str to the terminal through the RS-232 interface by interrupt mode.

Parameter

pcStr [in]
 pointer of string which want to display

Return Value

None

Example

```
sysPrintf("Hello World!\n");
```

sysprintf

Synopsis

```
VOID sysPrintf (
    PINT8pcStr, ...
)
```

Description

The function sends the specified str to the terminal through the RS-232 interface by polling mode.

Parameter

pcStr [in]
 pointer of string which want to display

Return Value

None

Example

```
sysprintf("Hello World!\n");
```

sysPutChar

Synopsis

```
VOID sysPutChar (
    UINT8 ucCh
)
```

Description

The function sends the specified ch to the UART.

Parameter

ucCh [in]
 character which want to display

Return Value

None

Example

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

sysUartInstallcallback

Synopsis

```
void sysUartInstallcallback (
    UINT32 u32IntType,
    PFN_SYS_UART_CALLBACK pfnCallback
)
```

Description

The function is used to install the call back function for received data ready or received data time out.

Parameter

u32IntType [in]
 UART interrupt type. Please refer Table 29-10 UART Interrupt type Definition,

pfnCallback [in]
 a function pointer to process the received data ready and received data time out event. Please refer Table 29-19 PFN_SYS_UART_CALLBACK Structure Definition.

Return Value

None

Example

```
sysUartInstallcallback(UART_INT_RDA, UartDataValid_Handler);
sysUartInstallcallback(UART_INT_RDTO, UartDataTimeOut_Handler);
```

sysUartEnableInt

Synopsis

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

Description

The function is used to enable UART interrupt.

Parameter

eIntType [in]

UART interrupt type. Please refer Table 29-10 UART Interrupt type Definition,

Return Value

None

Example

```
sysUartEnableInt(UART_INT_RDA);
sysUartEnableInt(UART_INT_RDTO);
.....
sysUartEnableInt(UART_INT_NONE);
```

sysUartTransfer

Synopsis

```
VOID sysUartTransfer (
    INT8 *pu8buf,
    UINT32 u32Len
)
```

Description

The function is used to transfer data.

Parameter

pu8buf [in]

Transfer data buffer pointer.

u32Len [in]

Transfer data length.

Return Value

None

Example

```
sysUartTransfer(pi8UartBuf, u32Count);
```

sysUartPort

Synopsis

```
VOID sysUartPort (
    UINT32 u32Port
)
```

Description

The function is used to specify UART port to be initialized.

Parameter

u32Port [in]

UART port number. 0 means high speed UART port. 1 means normal speed UART port.

Return Value

None

Example

```
/* Initialize high speed UART port 0 */
sysUartPort(0);
```

sysInitializeHUART

Synopsis

```
INT32
sysInitializeHUART (
    WB_UART *uart
)
```

Description

WB_UART is the device initialization structure.

Parameter

uart [in]

Please refer to Table 29-18 WB_UART Structure Definition. Normal speed UART means the baud rate less or equal to 115200 bps. And high speed UART means the baud rate up to 921600 bps.

Return Value

Successful

WB_INVALID_PARITY	Invalid parity
WB_INVALID_DATA_BITS	Invalid data bits
WB_INVALID_STOP_BITS	Invalid stop bits
WB_INVALID_BAUD	Invalid baud rate

Example

```
WB_UART_T uart;
uart.uart_no = WB_UART_0 ;
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;
sysInitializeHUART(&uart); WB_UART_T uart;
```

sysHUartInstallcallback

Synopsis

```
void sysUartInstallcallback (
    UINT32 u32IntType,
    PFN_SYS_UART_CALLBACK pfnCallback
)
```

Description

The function is used to install the call back function for received data ready or received data time out.

Parameter

u32IntType [in]

UART interrupt type. Please refer Table 29-10 UART Interrupt type Definition,

pfnCallback [in]

a function pointer to process the received data ready and received data time out event. Please refer Table 29-19 PFN_SYS_UART_CALLBACK Structure Definition.

Return Value

None

Example

```
sysHUartInstallcallback(UART_INT_RDA, HuartDataValid_Handler);
sysHUartInstallcallback(UART_INT_RDTO, HuartDataTimeOut_Handler);
```

sysHuartEnableInt

Synopsis

```
VOID sysHuartEnableInt (
```

INT32 eIntType

)

Description

The function is used to enable high speed UART interrupt.

Parameter

eIntType [in]

High speed UART interrupt type. Please refer Table 29-10 UART Interrupt type Definition,

Return Value

None

Example

```
sysHuartEnableInt(UART_INT_RDA);
sysHuartEnableInt(UART_INT_RDTO);
.....
sysHuartEnableInt(UART_INT_NONE);
```

sysHuartTransfer

Synopsis

```
VOID sysHuartTransfer (
    INT8* pu8buf,
    UINT32 u32Len
)
```

Description

The function is used to transfer data through high speed UART port.

Parameter

pu8buf [in]

Transfer data buffer pointer.

u32Len [in]

Transfer data length.

Return Value

None

Example

```
sysHuartTransfer(pi8UartBuf, u32Count);
```

sysHuartReceive

Synopsis

```
INT8
sysHuartReceive (VOID)
```

Description

The function is used to receive character through high speed UART port. It will be blocked in this function until data in.

Parameter

None

Return Value

A received byte

Example

```
Character = sysHuartReceive();
```

29.3.3 AIC API functions

sysDisableInterrupt

Synopsis

```
ERRCODE
sysDisableInterrupt (
    INT_SOURCE_EintNo
)
```

Description

This function is used to disable interrupt source.

Parameter

intNo [in]
 interrupt source number. Please refer the Table 29-11 Interrupt No. Definition.

Return Value

Successful or Fail.

Example

```
/* Disable timer 0 interrupt (source number is 7) */
INT32 status;
status = sysDisableInterrupt(7);
```

sysEnableInterrupt

Synopsis

```
ERRCODE
sysEnableInterrupt (
    INT_SOURCE_EintNo
)
```

Description

This function is used to enable interrupt source. Please refer the Table 29-11 Interrupt No. Definition.

Parameter

intNo [in]
 interrupt source number

Return Value

Successful or Fail.

Example

```
/* Enable timer 0 interrupt (source number is 7) */
INT32 status;
status = sysEnableInterrupt(7);
```

sysGetIbitState

Synopsis

```
BOOL
sysGetIbitState (VOID)
```

Description

This function is used to get the status of interrupt disable bit, I-bit, of CPSR register.

Parameter

None

Return Value

TRUE I-bit is clear
 FALSE I-bit is set.

Example

```
BOOL int_status;
Int_status = sysGetIbitState();
```

sysGetInterruptEnableStatus

Synopsis

UINT32

sysGetInterruptEnableStatus (VOID)

Description

This function is used to get the enable/disable status of low channel interrupts which save in AIC_IMR register.

Parameter

None

Return Value

value of AIC_IMR register

Example

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

sysGetInterruptHighEnableStatus

Synopsis

UINT32

sysGetInterruptHighEnableStatus (VOID)

Description

This function is used to get the enable/disable status of high channel interrupts which save in AIC_IMRH register.

Parameter

None

Return Value

value of AIC_IMRH register

Example

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

sysInstallExceptionHandler

Synopsis

PVOID

```
sysInstallExceptionHandler (
    INT32 nExceptType,
    PVOID pNewHandler
)
```

Description

This function is used to install pNewHandler into exceptType exception.

Parameter

nExceptType [in]

WB_SWI, WB_D_ABORT, WB_I_ABORT, WB_UNDEFINE. Please refer the Table 29-12 Interrupt Exception Type Definition.

pNewHandler [in]

pointer of the new handler

Return Value

a pointer which point to old handler

Example

```
/* Setup own software interrupt handler */
PVOID oldVect;
oldVect = sysInstallExceptionHandler(WB_SWI, pNewSWIHandler);
```

sysInstallFiqHandler

Synopsis

```
PVOID
sysInstallFiqHandler (
    PVOID pNewISR
)
```

Description

Use this function to install FIQ handler into interrupt vector table.

Parameter

pNewISR [in]

pointer of the new ISR handler

Return Value

a pointer which point to old ISR

Example

```
/* Setup own FIQ handler */
```

```
PVOID oldVect;
oldVect = sysInstallFiqHandler(pNewFiqISR);
```

sysInstallIrqHandler

Synopsis

```
PVOID
sysInstallIrqHandler (
    PVOID pNewISR
)
```

Description

Use this function to install FIQ handler into interrupt vector table.

Parameter

pNewISR [in]
pointer of the new ISR handler

Return Value

A pointer which point to old ISR

Example

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

sysInstallISR

Synopsis

```
PVOID
sysInstallISR (
    INT32 nIntTypeLevel,
    INT_SOURCE_EeIntNo,
    PVOID pNewISR
)
```

Description

Interrupt priority level is 0 ~ 7. Level 0 is FIQ, and level 1 ~ 7 are IRQ. The highest priority is 0, and the lowest priority is 7. Use this function to set up interrupt source (intNo) pNewISR handler to AIC interrupt vector table.

Parameter

nIntTypeLevel [in]

FIQ_LEVEL_0, IRQ_LEVEL_1 ~ IRQ_LEVEL_7. Please refer the Table 29-13 Interrupt Priority Definition

eIntNo [in]

interrupt source number. Please refer the Table 29-11 Interrupt No. Definition

pNewISR [in]

Function pointer of new ISR.

Return Value

Afunction pointer which points to old ISR

Example

```
/* Setup timer 0 handler */
PVOID oldVect;
oldVect = sysInstallISR(IRQ_LEVEL_1, IRQ_TMR0, pTimerISR);
```

sysSetAIC2SWMode

Synopsis

ERRCODE
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

None

Return Value

Successful

Example

```
/* Set AIC as software mode */
sysSetAIC2SWMode();
```

sysSetGlobalInterrupt

Synopsis

ERRCODE
sysSetGlobalInterrupt (
 INT32 nIntState
)

Description

The function is used to enable or disable all interrupt sources.

Parameter

nIntState [in]

ENABLE_ALL_INTERRUPTS or DISABLE_ALL_INTERRUPTS

Return Value

Successful

Example

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

sysSetInterruptPriorityLevel

Synopsis

ERRCODE

```
sysSetInterruptPriorityLevel (
    INT_SOURCE_EeIntNo,
    UINT32 uIntLevel
)
```

Description

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

Parameter

eIntNo [in]

interrupt source number. Please refer the Table 29-11 Interrupt No. Definition.

uIntLevel [in]

Interrupt priority. Please refer Table 29-13 Interrupt Priority Definition.

Return Value

Successful or Fail.

Example

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

sysSetInterruptType

Synopsis

```

ERRCODE
sysSetInterruptType (
    INT_SOURCE_EeIntNo,
    UINT32 intSourceType
)
    
```

Description

The interrupt has four kinds of interrupt source types. They are low level sensitive, high level sensitive, negative edge trigger, and positive edge trigger. The default is high level sensitive. This function is used to change the interrupt source type.

Parameter

eIntNo [in]
 interrupt source number. Please refer the Table 29-11 Interrupt No. Definition.

uIntSourceType [in]
 Interrupt trigger type. Please refer the Table 29-15 Interrupt Trigger Type Definition.

Return Value

Successful or Fail.

Example

```

/* Change timer 0 source type to be positive edge trigger */
INT32 status;
status = sysSetInterruptType(IRQ_TMR0, POSITIVE_EDGE_TRIGGER);
    
```

sysSetLocalInterrupt

Synopsis

```

ERRCODE
sysSetLocalInterrupt (
    INT32 nIntState
)
    
```

Description

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

Parameter

nIntState [in]
 Enable or disable ARM core's F and I bit. Please refer Table 29-14 Local Interrupt Type Definition.

Return Value

Successful

Example

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

29.3.4 Cache API functions

sysDisableCache

Synopsis

VOID sysDisableCache (VOID)

Description

This function is used to disable cache.

Parameter

None

Return Value

None

Example

```
/* disabled cache */
sysDisableCache();
```

sysEnableCache

Synopsis

INT32
 sysEnableCache (
 UINT32 uCacheOpMode
)

Description

This function is used to enable cache.

Parameter

uCacheOpMode [in]
 CACHE_WRITE_BACK, CACHE_WRITE_THROUGH

Return Value

Success

Example

```
/* enable cache */
sysEnableCache();
```

sysFlushCache

Synopsis

```
VOID sysFlushCache (
    INT32 nCacheType
)
```

Description

This function is used to flush system cache. The parameter, cacheType is used to select cache which needs to be flushed.

Parameter

nCacheType [in]
I_CACHE, D_CACHE, I_D_CACHE

Return Value

None

Example

```
/* flush cache */
sysFlushCache(I_D_CACHE);
```

sysGetCacheState

Synopsis

```
BOOL
sysGetCacheState (VOID)
```

Description

This function is used to get the enable/disable status of cache.

Parameter

None

Return Value

Cache State

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 [in]

The memory starting address.

size [in]

The memory size.

cache_mode [in]

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);

```

sysGetCacheMode

Synopsis

INT32

sysGetCacheMode (VOID)

Description

This function is used to get cache mode.

Parameter

uCacheOpMode [in]

CACHE_WRITE_BACK, CACHE_WRITE_THROUGH

Return Value

Success

Example

None

29.3.5 Clock Control API functions

sysGetExternalClock

Synopsis

```
UINT32
sysGetExternalClock (VIOD)
```

Description

This function is used to get external clock setting. IBR only support 2 kinds of external clock frequency. 12MHz. So external clock will be 12MHz. The power on setting must meet the external clock.

Parameter

None

Return Value

External clock. Unit : Hz

Example

```
/* Read system clock setting */
UINT32 u32ExtFreq;
u32ExtFreq = sysGetExternalClock();
```

sysSetSystemClock

Synopsis

```
ERRCODE
sysSetSystemClock (
    E_SYS_SRC_CLK eSrcClk,
    UINT32 u32PIIHz,
    UINT32 u32SysHz
)
```

Description

This function is used to write system clock setting includes PLL output frequency, System clock. The function gets the external clock automatically by power on setting.

Parameter

eSrcClk [in]
System clock source.

It could be eSYS_EXT, eSYS_APLL and eSYS_UPLL. They mean the

system clock source come from external clock, APLL and UPLL respectively.

u32PIIHz[in] Set the APLL or UPLL output frequency.

Unit : Hz.

u32SysHz[in] Set the system clock output frequency.

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

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

These frrequency exist multiplication factor. It means $PLL \geq n * SYS$, And HCLK clock is always equal to $SYS \text{ clock} / 2$.

Where n is integer. And HCLK clock is SDR/DDR/DDR2 clock.

PLL clock must under or equal to 432MHz.

System clock must under or equal to the source clock.

HCLKclock depends on the layout and core power. Generally, it can up to 150MHz in core power 1.2V.

Return Value

Successful or Error code

Example

```

/* Write system clock setting */
sysSetSystemClock(eSYS_UPLL,          /* system clock come from UPLL */
                  288000000,          /* UPLL = 288MHz */
                  288000000) /* SYS = 288MHz */
    
```

sysSetDramClock

Synopsis

```

UINT32
sysSetDramClock (
    E_SYS_SRC_CLK eSrcClk,
    UINT32 u32PLLClockHz,
    UINT32 u32DramClock
)
    
```

Description

This function is used to write memory clock setting includes PLL output frequency,

memory clock. The function gets the external clock automatically by power on setting.

Parameter

eSrcClk [in]

System clock source.

It will be limited to eSYS_MPLL.

u32PLLClockHz [in] MPLL output frequency.

Unit : Hz.

u32DramClock [in]: Set the memory clock output frequency.

Unit : Hz. The system clock source can only be MPLL.

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

MCLK clock is equal to half of u32DdrHz.

MCLK need great than HCLK1, HCLK2 and HCLK3.

Max MPLL output clock will be 360MHz.

Return Value

Successful or Error code

Example

```

/* Write memory clock setting */
sysSetDramClock(eSYS_MPLL, /* system clock come from MPLL */
                360000000, /* MPLL = 360MHz */
                360000000) /* DDR = 360MHz */
sysSetSystemClock(eSYS_UPLL, /* system clock come from UPLL */
                  288000000, /* UPLL = 288MHz */
                  288000000) /* SYS = 288MHz */
    
```

sysSetCPUClock

Synopsis

INT32

```

sysSetCPUClock (
    UINT32 u32CPUClock
)
    
```

Description

This function is used to set CPU clock.

Parameter

u32CPUClock [in]

CPU clock.

There are some limitations in the clock function due to hardware's limitation. The CPU clock comes from SYS clock. It must less or equal to SYS clock. The CPU divider only support even divider. It means CPU = SYS, SYS/2, SYS/4,... or SYS/16. HCLK1 clock depends on CPU clock.

If CPU divider =1, HCLK1 = CPU/2.

If CPU divider !=1, HCLK1 = CPU.

Return Value

Successful or Error code

Example

```
/* Write system clock setting */
sysSetSystemClock(eSYS_UPLL, /* E_SYS_SRC_CLK eSrcClk */
    192000000, /* UINT32 u32PIIKHz */
    192000000); /* UINT32 u32SysKHz*/
sysSetCPUClock(192000000);
```

sysSetAPBClock

Synopsis

INT32

```
sysSetAPBClock (
    UINT32 u32APBClock
)
```

Description

This function is used to set APB clock.

Parameter

u32APBClock [in]

APB clock.

There are some limitations in the clock function due to hardware's limitation. The APB clock comes from HCLK1 clock. It must less or equal to HCLK1 clock. Max APB divider is 8.

Return Value

Successful or Error code

Example

```
/* Write system clock setting */
```

```

sysSetSystemClock(eSYS_UPLL,          /* E_SYS_SRC_CLK eSrcClk */
                  288000000,          /* UINT32 u32PllKHz */
                  288000000);        /* UINT32 u32SysKHz */
sysSetCPUClock(288000000);          /* HCLK1 = 144MHz */
sysSetAPBClock(72000000);          /* APB = 72MHz */
    
```

sysGetPLLOutputHz

Synopsis

```

UINT32
sysGetPLLOutputHz (
    E_SYS_SRC_CLK eSysPll,
    UINT32 u32FinHz
)
    
```

Description

This function is used to read PLL output frequency.

Parameter

peSrcClk [in]

Specified PLL wants to know.

It could be eSYS_APLL=2 and eSYS_UPLL = 3.

u32FinHz [in] External clock. Unit : Hz.

Return Value

Specified PLL output clock. Unit : Hz.

Example

```

u32ExtFreq = sysGetExternalClock();
u32PllOutHz = sysGetPLLOutputHz(eSYS_UPLL, u32ExtFreq);
    
```

sysGetSystemClock

Synopsis

```

UINT32
sysGetSystemClock (VOID)
    
```

Description

This function is used to get systemclock.

Parameter

None

Return Value

System clock. Unit: Hz.

Example

```
/* Read system clock setting */
UINT32 u32SysFreq;
u32SysFreq = sysGetSystemClock();
```

sysGetDramClock

Synopsis

UINT32
sysGetDramClock (VOID)

Description

This function is used to get DRAM clock.

Parameter

None

Return Value

DRAM clock. Unit: Hz.

Example

```
/* Read DRAM clock setting */
UINT32 u32DramFreq;
u32DramFreq = sysGetDramClock();
```

sysGetCPUClock

Synopsis

UINT32
sysGetCPUClock (VOID)

Description

This function is used to get CPUclock.

Parameter

None

Return Value

CPU clock. Unit: Hz.

Example

```
/* Read CPU clock setting */
UINT32 u32CPUFreq;
u32CPUFreq = sysGetCPUClock();
```

sysGetHCLK1Clock

Synopsis

UINT32
 sysGetHCLK1Clock (VOID)

Description

This function is used to get HCLK1 clock.

Parameter

None

Return Value

HCLK1 clock. Unit: Hz.

Example

```
/* Read HCLK1 clock setting */
UINT32 u32HCLK1Freq;
u32HCLK1Freq = sysGetHCLK1Clock();
```

sysGetAPBClock

Synopsis

UINT32
 sysGetAPBClock (VOID)

Description

This function is used to get APBclock.

Parameter

None

Return Value

APB clock. Unit: Hz.

Example

```
/* Read HCLK1 clock setting */
UINT32 u32APBFreq;
u32APBFreq = sysGetAPBClock();
```

sysSetPIIClock

Synopsis

UINT32
 sysSetPIIClock (
 E_SYS_SRC_CLK eSrcClk,

```
UINT32 u32TargetHz
```

```
)
```

Description

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

Parameter

eSrcClk [in]

eSYS_APLL = 2 or eSYS_UPLL = 3.

u32TargetHz [in]

Target PLL output frequency. Unit : Hz.

Return Value

Specified PLL output frequency. Unit : Hz. The return value may not same as the specified value due to hardware’s limitation. If not meet the hardware SPEC, library will auto to search the nearly frrequency.

Example

```
/* Write system clock setting */
sysSetSystemClock(eSYS_UPLL,          /* system clock come from UPLL */
                  300000000,          /* UPLL = 300MHz */
                  300000000);        /* SYS = 300MHz */
/*Specified APLL clock */
sysSetPIIClock(eSYS_APLL,
               432000000);           /* SYS = 432MHz */
```

sysCheckPIIConstraint

Synopsis

```
VOID sysCheckPIIConstraint (
    BOOL blsCheck
)
```

Description

This function is used to enable or disable constraint checking for setting PLLclock.

Parameter

None

Return Value

None

Example

```
/* Set PLL clock without constraint check */
```

```

sysCheckPllConstraint(FALSE); /* Disable constraint checking */
sysSetSystemClock(eSYS_UPLL, /* system clock come from UPLL */
    318000000, /* UPLL = 318MHz */
    318000000); /* SYS = 318MHz */
sysCheckPllConstraint(FALSE); /* Enable constraint checking for next time */
    
```

29.3.6 Power management API functions

sysPowerDown

Synopsis

```

INT32
sysPowerDown (
    UINT32u32WakeUpSrc
)
    
```

Description

This function was used to enter standby mode. The function also specified the wake-up channel to wake up system. Programmer need to disable the analog IPs such as TV DAC, ADC and LVD and so on before entry standby mode.

Parameter

u32WakeUpSrc [in]
 Wakeup channels. Please reference **Error! Reference source not found.**

Return Value

Successful

Example

```

sysPowerDown(WE_GPIO);
    
```

29.4 System Error Code Table

Code Name	Value	Description
Successful	0	Successful
Fail	-1	Fail
WB_INVALID_PARITY	0xFFFF0A01	Invalid parity
WB_INVALID_DATA_BITS	0xFFFF0A02	Invalid data bits
WB_INVALID_STOP_BITS	0xFFFF0A03	Invalid stop bits
WB_INVALID_BAUD	0xFFFF0A04	Invalid baud rate
WB_PM_PD_IRQ_Fail	0xFFFF0A051	Invalid power down IRQ

Code Name	Value	Description
WB_PM_Type_Fail	0xFFFF0A06	Invalid power manager type
WB_PM_INVALID_IRQ_NUM	0xFFFF0A07	Invalid IRQ number
WB_MEM_INVALID_MEM_SIZE	0xFFFF0A08	
WB_INVALID_TIME	0xFFFF0A09	
WB_INVALID_CLOCK	0xFFFF0A0A	
E_ERR_CLK	0xFFFF0A0A	Wrong clock setting

Table 29-20 System Error Code Table

30 Touch ADC Library

30.1 System Library Overview

The N9H26Touch ADC library provides a set of APIs to report the X and Y-axis coordinate, battery voltage and analog keypad. With these APIs, user can read the position that was touched in touch panel, get the current battery voltage and get the keypad scancode.

30.2 Definition

30.2.1 Constant

ADC read mode

Field name	Value	Description
eADC_KEY	0	(Unused in the driver)
eADC_TOUCH	1	(Unused in the driver)
eADC_AIN	2	Normal ADC conversion event done type.
eADC_POSITION	3	Converse position event done type
eADC_PRESSURE	4	Conversion pressure event done type

Table 30-1 ADC read mode Definition

30.3 API function

30.3.1 Timer API functions

DrvADC_Open

Synopsis

INT32

DrvADC_Open (void)

Description

This function is used to open the Touch ADC library.

Parameter

None

Return Value

Successful

Example

```
/* Initialize Touch ADC library and enable IP clock*/
```

```
DrvADC_Open();
```

DrvADC_Close

Synopsis

```
INT32  
DrvADC_Close (void)
```

Description

Close the ADC library.

Parameter

None

Return Value

Successful

Example

```
/* Close Touch ADC library*/  
DrvADC_Close();
```

DrvADC_InstallCallback

Synopsis

```
INT32  
DrvADC_InstallCallback (  
    E_ADC_INT_TYPE eIntType,  
    PFN_ADC_CALLBACK pfnCallback,  
    PFN_ADC_CALLBACK* pfnOldCallback  
)
```

Description

This function was used to install callback function that is used to notice the upper layer for event complete.

Parameter

eIntType [in]
Interrupt event type.

pfnCallback [in]
The callback function wants to register

pfnOldCallback [out]
old callback function

Return Value

0 Successful.
 -1 Fail

Example

```

/* Read the x and y-axis coordinate */
static void Pressure_callback(UINT32 u32code)
{
    /* The u32code[14:0] is parameterZ2. u32code[15] is Valid(1) or Invalid(0).
    The u32code[30:16] is Z1 position. u32code[31] is Valid(1) or Invalid(0) */
    ...
}

static void Position_callback(UINT32 u32code)
{
    /* The u32code[14:0] is Y position. u32code[15] is Valid(1) or Invalid(0).
    The u32code[30:16] is X position. u32code[31] is Valid(1) or Invalid(0).*/
    ...
}

DrvADC_Open();
DrvADC_InstallCallback(eADC_POSITION,
    Position_callback,
    &pfnOldCallback);
DrvADC_InstallCallback(eADC_PRESSURE,
    Pressure_callback,
    &pfnOldCallback);
    
```

DrvADC_PenDetection

Synopsis

```

INT32
DrvADC_PenDetection (
    BOOL bls5Wire
)
    
```

Description

This function was used to read the touching position and presssure

Parameter

bls5Wire 5 wires or 4 wires touch panel.
 1: 5 wires touch panel
 0: 4 wires touch panel

Return Value

Successful Pen state is down. Touch ADC is triggering

E_ADC_BUSY	Touch ADC is busy
E_TOUCH_UP	Pen state is up. (No touching event)

Example

```

DrvADC_Open();
DrvADC_InstallCallback(eADC_POSITION,
    Position_callback,
    &pfnOldCallback);
DrvADC_InstallCallback(eADC_PRESSURE,
    Pressure_callback,
    &pfnOldCallback);
btime = sysGetTicks(TIMER0);
etime = btime;
while ((etime - btime) <= 300)
{
    while(TouchPanel_time==TRUE){
        TouchPanel_time = FALSE;
        do{
            ret = DrvADC_PenDetection(bls5Wire);
        }while(ret != Successful);
    }
    etime = sysGetTicks(TIMER0);
}
sysClearTimerEvent(TIMER0, tmp);
    
```

DrvADC_KeyDetection

Synopsis

```

INT32
DrvADC_KeyDetection (
    UINT32 u32Channel,
    UINT32* pu32KeyCode
)
    
```

Description

This function was used to read the scancode of keypad.

Parameter

u32Channel [in]
 Channel number. The value from 1 to 3.

pu32KeyCode [out]

Scancode.

Return Value

Successful	keypad state is down
E_ADC_BUSY	Touch ADC is busy
E_KEYPAD_UP	keypad state is up. (No keypad event)

Example

```

DrvADC_Open();
btime = sysGetTicks(TIMER0);
etime = btime;
while ((etime - btime) <= 300){
    if(DrvADC_KeyDetection(u32Channel, &u32KeyCode)==Successful){ /* ready */
        if(u32KeyCode!=0)
            sysprintf("Key Scan code = 0x%x\n", u32KeyCode);
    }else
        if(u32KeyCode!=0)
            sysprintf("Key Scan code = 0x%x\n", u32KeyCode);
    etime = sysGetTicks(TIMER0);
}
sysClearTimerEvent(TIMER0, tmp);
    
```

DrvADC_VoltageDetection

Synopsis

```

INT32
DrvADC_VoltageDetection (
    UINT32 u32Channel
)
    
```

Description

This function was used to read the conversion value.

Parameter

u32Channel [in]
 Channel number. The value from 1 to 3.

Return Value

Successful	Touch ADC is triggering
E_ADC_BUSY	Touch ADC is busy

Example

```

static void VoltageDetect_callback(UINT32 u32code)
    
```

```

/* u32code is the ADC value */
    blsValidVoltageDet = TRUE;
    u32VoltageValue = u32code;
    u32Count = u32Count+1;
    if(u32code==0){
        sysprintf("Voltage Detect Value %d = %d\n", u32Count, u32code);
    }
    sysprintf("Voltage Detect Value %d = %d\n", u32Count, u32code);
}
DrvADC_Open();
DrvADC_InstallCallback(eADC_AIN,
    VoltageDetect_callback,
    &pfnOldCallback);
btime = sysGetTicks(TIMER0);
etime = btime;
while ((etime - btime) <= 1000)
{
    while(VoltageDetect_time==TRUE){
        VoltageDetect_time = FALSE;
        if(DrvADC_VoltageDetection(u32Channel)==Successful){/* ready */
            if(blsValidVoltageDet==TRUE){ /* Key code has been updated */
                blsValidVoltageDet = FALSE;
            }
        }
    }
    etime = sysGetTicks(TIMER0);
}
sysClearTimerEvent(TIMER0, tmp);

```

DrvADC_Wakeup

Synopsis

```

INT32
DrvADC_Wakeup (
    E_ADC_WAKEUPeWakeupSrc
)

```

Description

This function was used to set the wake up source.

Parameter

eWakeupSrc [in]

eADC_WAKEUP_TOUCH or eADC_WAKEUP_KEY.

Return Value

eADC_WAKEUP_KEY	Wake up source from touch panel
eADC_WAKEUP_TOUCH	Wake up source from touch key

Example

None

IsPenDown

Synopsis

```
INT32
IsPenDown (void)
```

Description

This function was used to check if pen (finger) touched or not.

Parameter

None

Return Value

0	Pen up
1	Pen down
E_ADC_BUSY	Touch ADC is busy

Example

None

adc_read

Synopsis

```
INT32
adc_read (
    unsigned char mode,
    unsigned short int *x,
    unsigned short int *y
)
```

Description

This function was used to read back the touched x and y axis coordinate.

Parameter

mode [in]

useless.

x [out]

x axis coordinate.

y [out]

y axis coordinate.

Return Value

- 0 Pen up
- 1 Pen down
- E_ADC_BUSY Touch ADC is busy

Example

None

30.4 Touch ADC Error Code Table

Code Name	Value	Description
E_ADC_BUSY	0xB800F001	Touch ADC is busy
E_TOUCH_UP	1	Pen state is up. (No touching event)
E_KEYPAD_UP	1	Keypad state is up
Successful	0	Pen state is down. Touch ADC is triggering.

Table 30-2 Touch ADC Error Code Table

31 UDC Library

31.1 UDC Library Overview

This library is designed to make user application to use N9H26UDC more easily. The UDC library has the following features:

- Support all Basic USB operations.
- Pass USB-IF Chapter 9.

SDK Non-OS provide two usb class libraries for the USB class reference sample. User can refer to the libraries to develop him own class libraries. Mass Storage Class device: mscd library.

- Pass the USB-IF Mass Storage Class Test
- Provide flash options to build MSC device as a Composite device with RAM disk, NAND Disk, and SD Card Reader.

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.

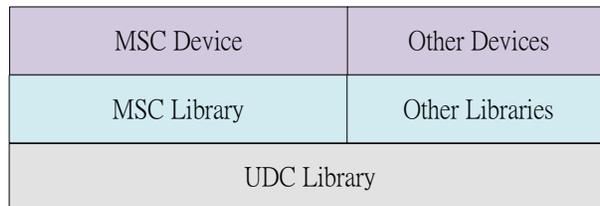


Figure 31-1 UDC Library Frame work

31.1.1 System Overview

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

- pfnHighSpeedInit
- mscdHighSpeedInit
- pfnFullSpeedInit
- mscdFullSpeedInit

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

- mscdInit

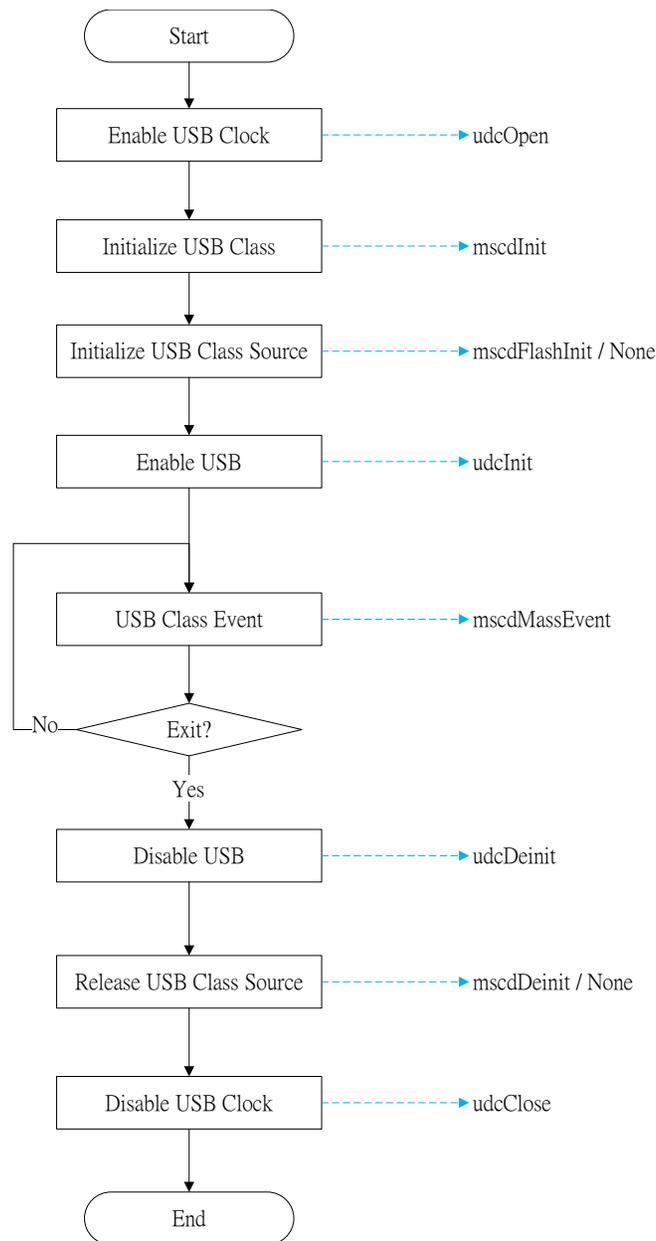


Figure 31-2 UDC Library Control Flow

31.2 Definition

31.2.1 Structure

USB_D_INFO_T Structure

Field name	Data Type	Description
Descriptor pointer		
<i>pu32DevDescriptor</i>	PUINT32	Device Descriptor pointer

Field name	Data Type	Description
<i>pu32QualDescriptor</i>	PUINT32	Device Qualifier Descriptor pointer
<i>pu32HSConfDescriptor</i>	PUINT32	Standard Configuration Descriptor pointer for High speed
<i>pu32FSConfDescriptor</i>	PUINT32	Standard Configuration Descriptor pointer for Full speed
<i>pu32HOSConfDescriptor</i>	PUINT32	Other Speed Configuration Descriptor pointer for High speed
<i>pu32FOSConfDescriptor</i>	PUINT32	Other Speed Configuration Descriptor pointer for Full speed
<i>pu32StringDescriptor[7]</i>	PUINT32	String Descriptor pointer
Descriptor length		
<i>u32DevDescriptorLen</i>	UINT32	Device Descriptor Length
<i>u32QualDescriptorLen</i>	UINT32	Device Qualifier Descriptor pointer Length
<i>u32HSConfDescriptorLen</i>	UINT32	Standard Configuration Descriptor Length for High speed
<i>u32FSConfDescriptorLen</i>	UINT32	Standard Configuration Descriptor Length for Full speed
<i>u32HOSConfDescriptorLen</i>	UINT32	Other Speed Configuration Descriptor Length for High speed
<i>u32FOSConfDescriptorLen</i>	UINT32	Other Speed Configuration Descriptor Length for Full speed
<i>u32HIDDescriptorLen</i>	UINT32	HID Descriptor Length
<i>u32HIDRPTDescriptorLen[7]</i>	UINT32	HIDReport Descriptor Length
<i>u32StringDescriptorLen[7]</i>	UINT32	String Descriptor Length
USB D Init		

Field name	Data Type	Description
<i>pfnHighSpeedInit</i>	PFN_USBD_CALLBACK	High speed USB Device Initialization function
<i>pfnFullSpeedInit</i>	PFN_USBD_CALLBACK	Full speed USB Device Initialization function
Endpoint Number		
<i>i32EPA_Num</i>	INT32	Endpoint Number for EPA (-1 : Not used)
<i>i32EPB_Num</i>	INT32	Endpoint Number for EPB (-1 : Not used)
<i>i32EPC_Num</i>	INT32	Endpoint Number for EPC (-1 : Not used)
<i>i32EPD_Num</i>	INT32	Endpoint Number for EPD (-1 : Not used)
Endpoint Call Back		
<i>pfnEPACallBack</i>	PFN_USBD_EP_CALLBACK	Callback function pointer for Endpoint A Interrupt
<i>pfnEPBCallBack</i>	PFN_USBD_EP_CALLBACK	Callback function pointer for Endpoint B Interrupt
<i>pfnEPCCallBack</i>	PFN_USBD_EP_CALLBACK	Callback function pointer for Endpoint C Interrupt
<i>pfnEPDCallBack</i>	PFN_USBD_EP_CALLBACK	Callback function pointer for Endpoint D Interrupt
Class Call Back		
<i>pfnClassDataINCallBack</i>	PFN_USBD_CALLBACK	Callback function pointer for Class Data IN
<i>pfnClassDataOUTCallBack</i>	PFN_USBD_CALLBACK	Callback function pointer for Class Data OUT
<i>pfnDMACompletion</i>	PFN_USBD_CALLBACK	Callback function pointer for DMA Complete
<i>pfnReset</i>	PFN_USBD_CALLBACK	Callback function pointer for USB Reset Interrupt
<i>pfnSOF</i>	PFN_USBD_CALLBACK	Callback function pointer for USB SOF Interrupt
<i>pfnPlug</i>	PFN_USBD_CALLBACK	Callback function pointer for USB Plug Interrupt

Field name	Data Type	Description
<i>pfnUnplug</i>	PFN_USBD_CALLBACK	Callback function pointer for USB Un-Plug Interrupt
VBus status		
<i>u32VbusStatus</i>	UINT32	VBus Status

Table 31-1 USBD_INFO_T Structure Definition

31.3 API function

31.3.1 USB Device (UDC) API functions

udcOpen

Synopsis

VOID udcOpen (void)

Description

This function enables the engine clock.

Parameter

None

Return Value

None

Example

```
udcOpen ();
```

udcClose

Synopsis

VOID udcClose (void)

Description

This function disables the engine clock.

Parameter

None

Return Value

None

Example

```
udcClose ();
```

udcInit

Synopsis

VOID udcInit(void)

Description

This function initializes the software resource, enables its interrupt, and set VBus detect function.

Parameter

None

Return Value

None

Example

```
udcInit();
```

udcDeinit

Synopsis

VOID udcDeinit (void)

Description

Disable VBus detect function

Parameter

None

Return Value

None

Example

```
udcDeinit ();
```

udcIsAttached

Synopsis

BOOL
udcIsAttached (void)

Description

This function can get USB attach status.

Parameter

None

Return Value

TRUE USB is attached.
 FALSE USB isn't attached.

Example

```
/* Check USB attach status */
if(udclsAttached ())
    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(udclsAttachedToHost ())
    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 to check the status, i.e., user needs to polling the status during the timeout time.

udcSetSupendCallback

Synopsis

```
VOID udcSetSupendCallBack (
    PFN_USBD_CALLBACK pfun
)
```

Description

Set suspend interrupt call back function

Parameter

pfun [in]

suspend interrupt call back function pointer

Return Value

None

Example

```
void USB_Suspend(void)
{
    sysprintf("*** Suspend **\n");
}

udcSetSuspendCallBack(USB_Suspend);
```

31.3.2 Mass Storage Class (MSCD) API functions

mscdInit

Synopsis

VOID mscdInit (VOID)

Description

This function initializes software source (descriptors, callback functions, buffer configuration)

Parameter

None

Return Value

None

Example

```
mscdInit ();
```

mscdDeinit

Synopsis

VOID mscdDeinit (void)

Description

This function release software source (allocated by mscdInit)

Parameter

None

Return Value

None

Example

```
mscdDeinit();
```

mscdFlashInit

Synopsis

```
UINT8
mscdFlashInit (
    NDISK_T *pDisk,
    INT SDsector
)
```

Description

Initial the Flash capacity for usb device controller use.(Onechip selector NAND flash and one port SD)

Parameter

pDisk [in]
The internal data for NAND disk information.

SDsector [in]
Total sector for SD disk.

Return Value

0 Fail
1 Success

Example

```
NDISK_T MassNDisk;
INT SDsector;
SDsector = sicSdOpen0();
mscdFlashInit(&MassNDisk, SDsector);
```

Note

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable. (Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. The API can only single port SD and single CS NAND by mscdSdEnable or mscdNandEnable.

4. If user wants to export only SD, please link MSCSD library.
5. If user wants to export only NAND, please link MSCNAND library.
6. If user wants to export both SD and NAND, please link N9H26_MSC_NAND_SD library.

mscdFlashInitNAND

Synopsis

```

UINT8
mscdFlashInitNAND (
    NDISK_T *pDisk,
    NDISK_T *pDisk1,
    NDISK_T *pDisk2,
    INT SDsector
);
    
```

Description

Initial the Flash capacity for usb device controller use (thress chip selector NAND flash and one port SD).

Parameter

pDisk [in]
 The internal data for NAND disk information for CS0.

pDisk1 [in]
 The internal data for NAND disk information for CS1.

pDisk2 [in]
 The internal data for NAND disk information for CS2.

SDsector [in]
 Total sector for SD disk.

Return Value

0 Fail
 1 Success

Example

```

NDISK_T MassNDisk,MassNDisk1,MassNDisk2;
INT SDsector;
SDsector = sicSdOpen0();
mscdFlashInitNAND (&MassNDisk, &MassNDisk1, &MassNDisk2, SDsector);
    
```

Note

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable. (Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. The API can only single port SD by mscdSdEnable.
4. If user wants to export only SD, please linkMSCSD library.
5. If user wants to export only NAND, please link MSC NAND library.
6. If user wants to export both SD and NAND, please link MSC NAND & SD library.

mscdFlashInitExtend

Synopsis

```

UINT8
mscdFlashInitExtend (
    NDISK_T *pDisk,
    NDISK_T *pDisk1,
    NDISK_T *pDisk2,
    INT SDsector0,
    INT SDsector1,
    INT SDsector2,
    INT RamSize
);
    
```

Description

Initial the Flash capacity for usb device controller use (thress chip selector NAND flash and three ports SD).

Parameter

- pDisk [in]
The internal data for NAND disk information for CS0.
- pDisk1 [in]
The internal data for NAND disk information for CS1.
- pDisk2 [in]
The internal data for NAND disk information for CS2.
- SDsector0 [in]

Total sector for SD0 disk.
 SDsector1 [in]
 Total sector for SD1 disk.
 SDsector2 [in]
 Total sector for SD2 disk.
 RamSize [in]
 MSC_RAMDISK_1M~ MSC_RAMDISK_64M

Return Value

0 Fail
 1 Success

Example

```

NDISK_T MassNDisk, MassNDisk2;
INT SDsector;
status0= sicSdOpen0();
mscdFlashInitExtend(&MassNDisk, 0, & MassNDisk2, status0, 0,0 ,0);
    
```

Note

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable. (Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. If user wants to export only SD, please link MSC SD library.
4. If user wants to export only NAND, please link MSC NAND library.
5. If user wants to export both SD and NAND, please link MSC NAND& SD library.
6. If user wants to export only all flash, please link MSCAll library.

mscdSdEnable

Synopsis

```

VOID mscdSdEnable (
    UINT32 u32Enable
)
    
```

Description

This function enables the SD port for MSC.

Parameter

u32Enable [in]

MSC_SD_MP_PORT0 ~ MSC_SD_MP_PORT2

MSC_SD_PORT0 ~ MSC_SD_PORT2

Return Value

None

Example

```
/* Export two SD ports (Multiple partition) */
mscdSdEnable (MSC_SD_MP_PORT0|MSC_SD_MP_PORT1);
/* Export one SD port (Single partition) */
mscdSdEnable (MSC_SD_PORT0);
```

mscdNandEnable

Synopsis

```
VOID mscdNandEnable (
    UINT32 u32Enable
)
```

Description

This function enables the NAND CS for MSC.

Parameter

u32Enable [in]

MSC_NAND_CS0 ~ MSC_NAND_CS2

Return Value

None

Example

```
/* Export two NAND CS */
mscdNandEnable (MSC_NAND_CS0|MSC_NAND_CS2);
/* Export one NAND CS */
mscdNandEnable (MSC_NAND_CS0);
```

mscdSdUserWriteProtectPin

Synopsis

```
VOID mscdSdUserWriteProtectPin (
    UINT32 u32SdPort,
    BOOL bEnable,
    UINT32 u32GpioPort,
```

```

        UINT32 u32GpioPin
    )

```

Description

This function enables/disables the SD write protect function and SD write protect pin for MSC.

Parameter

```

u32SdPort [in]
    MSC_SD_PORT0 ~ MSC_SD_PORT2
bEnable[in]
    Enable or Disable Write Protection function (TRUE/FALSE)
u32GpioPort [in]
    MSC_SD_GPIO_PORTA ~ MSC_SD_GPIO_PORTE,
    MSC_SD_GPIO_PORTG, MSC_SD_GPIO_PORTH
u32GpioPin [in]
    GPIO pin number : 0~15

```

Return Value

None

Example

```

/* Set GPIOA Pin 2 for SD port0 Write Protect Pin */
mscdSdUserWriteProtectPin (MSC_SD_PORT0, TRUE, MSC_SD_GPIO_PORTA, 2);
/* Disable SD Port 0 Write Protect Pin function*/
mscdSdUserWriteProtectPin (MSC_SD_PORT0, FALSE, 0, 0);

```

Note

Only SD Port0 has default Write Protection pin and Write Protection function is default enable and use the default pin (GPA0).

mscdSdUserCardDetectPin

Synopsis

```

VOID mscdSdUserCardDetectPin (
    UINT32 u32SdPort,
    BOOL bEnable,
    UINT32 u32GpioPort,
    UINT32 u32GpioPin
)

```

Description

This function enables/disables the SD card detection function for MSC.

Parameter

u32SdPort [in]

MSC_SD_PORT0~ MSC_SD_PORT2

bEnable [in]

Enable or Disable card detection (TRUE/FALSE)

u32GpioPort [in]

MSC_SD_GPIO_PORTA ~ MSC_SD_GPIO_PORTE,

MSC_SD_GPIO_PORTG, MSC_SD_GPIO_PORTH

u32GpioPin [in]

GPIO pin number : 0~15

Return Value

None

Example

```

/* Set GPIOA Pin 2 for SD port0 Card detect Pin */
mscdSdUserCardDetectPin (MSC_SD_PORT0, TRUE, MSC_SD_GPIO_PORTA, 2);
/* Disable SD Port 0 Card detect function*/
mscdSdUserCardDetectPin (MSC_SD_PORT0, FALSE, 0, 0);
    
```

Note

1. Only SD Port0/2 has default Card detect pin and Card detect function is default enable and use the default pin (GPA1 for Port 0 and GPE11for Port2).
2. If user disable the Card detect function, MSC will consider that the SD card is always exist.

mscdMassEvent

Synopsis

```

VOID mscdMassEvent (
    PFN_USBD_EXIT_CALLBACK *callback_fun
)
    
```

Description

This function processes all the mass storage class commands such as read, write, inquiry, etc. The function has loop in it and it exits the loop according to the return value of the callback function.

Parameter

callback_fun [in]

The callback function for the Mass Event Exit condition. If it returns FALSE, the mass event service is disabled.

Return Value

None

Example

```
mscdMassEvent(udclsAttached);
```

Note

The API must be called when all APIs about MSC is completed.

mscdFlashInitCDROM

Synopsis

UINT8

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

Description

Initial the Flash capacity for usb device controller use with CDROM disk.(Onechip selector NAND flash and one port SD)

Parameter

pDisk [in]

The internal data for NAND disk information.

SDsector [in]

Total sector for SD disk.

pfnCallBack [in]

The CDROM call back function

CdromSizeInByte [in]

CDROM size

Return Value

- 0 Fail
- 1 Success

Example

```

NDISK_T MassNDisk;
INT SDsector;
SDsector = sicSdOpen0();
mscdFlashInitCDROM(&MassNDisk, SDsector, CDROM_Read, u32CdromSize);
    
```

Note

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable. (Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. The API can only single port SD and single CS NAND by mscdSdEnable or mscdNandEnable.
4. If user wants to export only SD, please link MSCSD with CDROM library.
5. If user wants to export only NAND, please link MSCNAND with CDROM library.
6. If user wants to export both SD and NAND, please link MSCNAND&SD with CDROM library.

mscdFlashInitExtendCDROM

Synopsis

```

UINT8
mscdFlashInitExtendCDROM(
    NDISK_T *pDisk,
    NDISK_T *pDisk1,
    NDISK_T *pDisk2,
    INT SDsector0,
    INT SDsector1,
    INT SDsector2,
    INT RamSize,
    PFN_MSCD_CDROM_CALLBACK pfnCallBack,
    INT CdromSizeInByte
);
    
```

Description

Initial the Flash capacity for usb device controller use (thress chip selector NAND flash and three ports SD).

Parameter

pDisk [in]

The internal data for NAND disk information for CS0.

pDisk1 [in]

The internal data for NAND disk information for CS1.

pDisk2 [in]

The internal data for NAND disk information for CS2.

SDsector0 [in]

Total sector for SD0 disk.

SDsector1 [in]

Total sector for SD1 disk.

SDsector2 [in]

Total sector for SD2 disk.

RamSize [in]

MSC_RAMDISK_1M ~ MSC_RAMDISK_64M

pfnCallBack [in]

The CDROM call back function

CdromSizeInByte [in]

CDROM size

Return Value

0 Fail

1 Success

Example

```

NDISK_T MassNDisk,MassNDisk1,MassNDisk2;
INT SDsector;
SDsector = sicSdOpen0();
mscdFlashInitExtendCDROM(NULL,NULL,NULL, 0,0,0,0,CDROM_Read,u32CdromSize);
    
```

Note

1. User can assign the export NAND flash (CS0/CS1/CS2) by mscdNandEnable. (Default is CS0 if user doesn't use mscdNandEnable)
2. User can assign the export SD card (Port0/Port1/Port2) by mscdSdEnable. (Default is Port0 if user doesn't use mscdSdEnable)
3. If user wants to export only SD, please link MSC SD with CDROM library.

4. If user wants to export only NAND, please link MSC NANDwith CDROM library.
5. If user wants to export both SD and NAND, please link MSC NAND& SD with CDROM library.
6. If user wants to export only all flash, please link MSCAll library.

mscdBlcokModeEnable

Synopsis

```
VOID mscdBlcokModeEnable (
    BOOLbEnable
)
```

Description

This function enables the SD port for MSC.

Parameter

bEnable[in]
 Enable / Disable

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
```

32 USB Core Library

32.1 USB Core Library Overview

The USB Core library is composed of four major parts, which are OHCI driver, EHCI driver, USB driver, and USB hub device driver. Each of these four drivers also represents one of the three-layered USB driver layers.

32.2 Definition

32.2.1 Constant

USB Host Like Pin Control Definition

Name	Value	Description
HOST_LIKE_PORT1_0	0x0	USB Host Like Port 1 output from GPA10 & GPA11
HOST_LIKE_PORT2_0	0x2	USB Host Like Port 2 output from GPD3 & GPD4
HOST_LIKE_PORT2_1	0x3	USB Host Like Port 2 output from GPA3 & GPA4
HOST_LIKE_PORT2_2	0x4	USB Host Like Port 2 output from GPD14 & GPD15
HOST_LIKE_PORT1_DISABLE	0xFF	Disable Port 1
HOST_LIKE_PORT2_DISABLE	0xFF	Disable Port 2

Table 32-1 USB Host Like Pin Control Definition

32.3 API function

32.4 USB 1.1 Host Like API function

USB_PortInit

Synopsis

```

INT
USB_PortInit (
    UINT32 u32PortType
)
    
```

Description

The function is used to specified USB host port type.

Parameter

u32PortType:

Please refer to Table 32-1 USB Host Like Pin Control Definition.

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 hoost ports if the port is useless.

Parameter

blsDisPort0 [in]
 TRUE to disable port 0. FALSE to enable port 0
 blsDisPort1 [in]
 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();
```

32.5 USB 1.1 & 2.0 Host API function

InitUsbSystem

Synopsis

```
INT
InitUsbSystem (VOID)
```

Description

Initialize the USB hardware and USB core library. This function must be invoked before any other functions. The USB library will scan device at this time, but the device will not be activated until the corresponding device driver was registered by USB_RegisterDriver().

Parameter

None

Return Value

0	Success
Otherwise	Failure

Example

```

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

DelnitUsbSystem

Synopsis

INT
DelnitUsbSystem (VOID) for USB Host 1.1 Like

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();
InitUsbSystem();
UMAS_InitUmasDriver();
.....
/* De-Initialize USB core library */
DelnitUsbSystem();
    
```

UMAS_InitUmasDriver

Synopsis

INT
 UMAS_InitUmasDriver (VOID)

Description

Initialize the USB mass storage driver. fsInitFileSystem() and InitUsbSystem() must be called prior to this API. Once an USB mass storage device detected, USB core library will initialize it and mount it to NVTFAT file system automatically.

Parameter

None

Return Value

0	Success
Otherwise	Failure

Example

```

/*
Initialize NVTFAT FAT file system, USB core system, and USB mass storage driver
*/
fsInitFileSystem();
InitUsbSystem();
UMAS_InitUmasDriver();
    
```

USB_RegisterDriver

Synopsis

INT
 USB_RegisterDriver (
 USB_DRIVER_T *driver
)

Description

Register a device driver with the USB library. In this function, USB library will also try to associate the newly registered device driver with all connected USB devices that have no device driver associated with it. Note that a connected USB device can be detected by USB library but may not work until it was associated with its corresponding device driver.

Parameter

driver [out]
 The USB device driver to be registered with USB core library

Return Value

0	Success
Otherwise	Failure

Example

```
static USB_DRIVER_T usblp_driver =
{
    "usblp",
    usblp_probe,
    usblp_disconnect,
    {NULL,NULL},
    {0},
    NULL,
    usblp_ids,
    NULL,
    NULL
};

INT UsbPrinter_Init(){
    if (USB_RegisterDriver(&usblp_driver))return -1;
    return 0;
}
```

USB_DeregisterDriver

Synopsis

```
VOID USB_DeregisterDriver (
    USB_DRIVER_T *driver
)
```

Description

Deregister a device driver.

Parameter

driver [out]
 The device driver to be deregistered

Return Value

0	Success
Otherwise	Failure

Example

None

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 [in]

The number of isochronous frames in a single URB. For other transfer types, this value must be zero.

Return Value

NULL	Failure
Otherwise	A pointer to the newly allocated URB

Example

```
_W99683_Camera->sbuf[i].urb = USB_AllocateUrb(FRAMES_PER_DESC);
if (_W99683_Camera->sbuf[i].urb == NULL)
{
    UART_printf("%s - USB_AllocateUrb(%d.) failed.\n", proc,
                FRAMES_PER_DESC);
    Return -1;
};
```

USB_FreeUrb

Synopsis

```
VOID USB_FreeUrb (
    URB_T *urb
)
```

Description

Frees the memory used by a urb.

Parameter

None

Return Value

None

Example

None

USB_SubmitUrb

Synopsis

```
INT
USB_SubmitUrb (
    URB_T *urb
)
```

Description

Submit a URB for executing data transfer

Parameter

urb [in]
 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 [out]
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
)    for USB Host 2.0 Only
    
```

Description

Builds a bulk urb, sends it off and waits for completion. This function sends a simple bulk message to a specified endpoint and waits for the message to complete, or timeout. Don't use this function from within an interrupt context.

Parameter

```

dev [in]
    pointer to the usb device to send the message to
pipe [in]
    endpoint "pipe" to send the message to
data [in]
    pointer to the data to send
len [in]
    length in bytes of the data to send
actual_length [out]
    pointer to a location to put the actual length transferred in bytes
timeout [in]
    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("dabus: usb_bulk_msg failed(%d)",ret);
    if (usb_set_interface (s->usbdev, _DABUSB_IF, 1) < 0) {
        err("set_interface failed");
        return -EINVAL;
    }
}

```

USB_malloc

Synopsis

```

VOID *USB_malloc (
    INT wanted_size,
    INT boundary
)

```

Description

Allocate a non-cacheable memory block started from assigned boundary. The total size of the USB library managed memory block is 256 KB.

Parameter

wanted_size [in]

The wanted size of non-cacheable memory block

boundary [in]

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 [out]
pointer to the USB_malloc() allocated memory block to be freed.

Return Value

None

Example

Same as USB_malloc()

33 VPE Library

33.1 VPE Library Overview

The VPE Library has the following features:

- Support format conversion
 - Input format
 - ◆ Y only
 - ◆ Planar YUV420
 - ◆ Planar YUV411
 - ◆ Planar YUV422
 - ◆ Planar YUV422 Transpose
 - ◆ Planar YUV444
 - ◆ Packet YUV422
 - ◆ Packet RGB555
 - ◆ Packet RGB565
 - ◆ Packet RGB888
 - Output format
 - ◆ Packet YUV422
 - ◆ Packet RGB555
 - ◆ Packet RGB565
 - ◆ Packet RGB888
- Support rotation
 - Normal
 - Right
 - Left
 - Upside down
 - Horizontal mirror
 - 180 degree
- Support MMU or Non-MMU mode.
 - MMU.
 - ◆ Block base mode with better memory usage. Please note that MMU feature only for Linux operation system.
 - Non-MMU mode
 - ◆ Line base mode with better performance.
- Support source and destination line offset

- Support limited on the fly mode with codec.

33.2 Definition

33.2.1 Constant

Interrupt type

eIntType	Value	Description
VPE_INT_COMP	0x0	Conversion complete
VPE_INT_PAGE_FAULT	0x1	Page fault if MMU on.
VPE_INT_PAGE_MISS	0x2	Page miss if MMU on.
VPE_INT_MB_COMP	0x3	Macro block complete if on the fly with codec.
VPE_INT_MB_ERR	0x4	Macro block error if on the fly with codec.

Table 33-1 Interrupt type Definition

VPE I/O Control Table

u32Cmd	u32Arg0	u32Arg1	u32Arg2	Description
VPE_IOCTL_TRIGGER	(Useless)	(Useless)	(Useless)	Trigger VPE
VPE_IOCTL_CHECK_TRIGGER	(Useless)	(Useless)	(Useless)	Check VPE complete Return 0, meaning VPE complete
VPE_IOCTL_SET_MMU_ENTRY	TRUE: Enable MMU operation	TLB entry	(Useless)	Enable VPE MMU operation
VPE_IOCTL_SET_TLB_ENTRY	Component entry: 0~7 <ul style="list-style-type: none"> ● 0 and 4: For packet or Y component ● and 5: For U component ● and 6: For V component ● 3 and 7: For destination image 	(Useless)	(Useless)	Specified the component entry if page fault. It has been handled by library
VPE_IOCTL_SET_SRCBUF_ADDR	Source packet buffer start address or planar Y buffer address	Source planar U buffer start address	Source planar V buffer start address	Specified the source buffer base address
VPE_IOCTL_SET_DSTBUF_ADDR	Destination packet buffer start address.	(Useless)	(Useless)	Specified the input order, input format and output format
VPE_IOCTL_SET_SRC_OFFSET	Left offset for source image	Right offset for source image	(Useless)	Specified the left and right offset for source image.

u32Cmd	u32Arg0	u32Arg1	u32Arg2	Description
VPE_IOCTL_SET_SRC_DIMENSION	Width of source image	Height of source image	(Useless)	Specified the dimension of source image.
VPE_IOCTL_SET_DST_DIMENSION	Width of destination image	Height of destination image	(Useless)	Specified the dimension of destination image.
VPE_IOCTL_SET_COLOR_RANGE	Source format color range. <ul style="list-style-type: none"> TRUE: Y 16~235, U and V 16~240. FALSE: source format is full range. 	Destination format color range. <ul style="list-style-type: none"> TRUE: Conversion to Y 16~235, U and V 16~240. FALSE: Full range for destination format 	(Useless)	Specified color range
VPE_IOCTL_SET_FILTER	<ul style="list-style-type: none"> VPE_SCALE_DDA: Directly drop algorithm VPE_SCALE_BILINEAR: Bilinear algorithm 	(Useless)	(Useless)	Specified upscale or downscale algorithm
VPE_IOCTL_SET_FMT	Source format	Destination format	(Useless)	Specified the source format and destination format
VPE_IOCTL_SET_MACRO_BLOCK	Y macro block if on the fly	X macro block if on the fly	(Useless)	It is useful if on the fly with codec.
VPE_IOCTL_HOST_OP	<ul style="list-style-type: none"> VPE_HOST_FRAME: Block base mode VPE_HOST_VDEC_LINE: Line base mode 	Rotation direction.	(Useless)	Specified the operation mode and rotation direction.

Table 33-2 VPE I/O Control Table Definition

33.3 API function

vpeOpen

Synopsis

```
ERRCODE
vpeOpen (void)
```

Description

Open VPE library.

Parameter

None

Return Value

Successful or error code.

Example

```
vpeOpen(); /* Enable VPE clock and interrupt */
```

vpeClose

Synopsis

ERRCODE
vpeClose (void)

Description

Close VPE library.

Parameter

None

Return Value

None

Example

```
vpeOpen();  
...  
vpeClose();
```

vpelInstallCallback

Synopsis

ERRCODE
vpelInstallCallback (
 E_VPE_INT_TYPE eIntType,
 PFN_VPE_CALLBACK pfnCallback,
 PFN_VPE_CALLBACK* pfnOldCallback
)

Description

Install call back function for user layer. The function let the VPE library call back to upper lay to inform user for registered interrupt event. However, page fault and page missing will be handled in library. Upper layer can ignore both

Parameter

eIntType [in]
 Please refer to Table 33-1 Interrupt type Definition.
pfnCallback [in]

Functionpointer for callback function.

pfnOldCallback [out]

Old callback function.

Return Value

Successful or error code.

Example

```
/* Install call back function for conversion done */
vpeOpen();
vpeInstallCallback(VPE_INT_COMP,
                  vpeCompleteCallback,
                  &OldVpeCallback);
```

vpeEnableInt

Synopsis

```
ERRCOD
vpeEnableInt (
    E_VPE_INT_TYPE eIntType
)
```

Description

Enable specified interrupt type.

Parameter

eIntType [in]

Please refer to Table 33-1 Interrupt type Definition.

Return Value

Successful or error code

Example

```
/* Enable frame end interrupt */
vpeEnableInt(VPE_INT_COMP);
vpeEnableInt(VPE_INT_PAGE_FAULT);
vpeEnableInt(VPE_INT_PAGE_MISS);
```

vpeDisableInt

Synopsis

```
ERRCODE
vpeDisableInt (
```

```

        E_VPE_INT_TYPE eIntType
    )

```

Description

Disable specified interrupt type.

Parameter

eIntType [in]

Please refer to Table 33-1 Interrupt type Definition.

Return Value

Successful or error code

Example

```

/* Disable frame end interrupt */
vpeDisableInt(VPE_INT_COMP);

```

vpeloctl

Synopsis

```

ERRCODE
vpeloctl (
    UINT32 u32Cmd,
    UINT32 u32Arg0,
    UINT32 u32Arg1,
    UINT32 u32Arg2
)

```

Description

VPE IO control function. The function is used to set some parameters for VPE hardware IP.

Parameter

Please refer to **Error! Reference source not found..**

Return Value

None.

Example

```

/* Setup hardware IP through IO ctrl */
vpeloctl(VPE_IOCTL_HOST_OP,
        VPE_HOST_FRAME,
        VPE_OP_NORMAL,
        NULL);

```

```

vpeIoctl(VPE_IOCTL_SET_SRCBUF_ADDR,
          (UINT32)pi8Y, /* They are virtual address if MMU on */
          (UINT32)pi8U,
          (UINT32)pi8V);
vpeIoctl(VPE_IOCTL_SET_FMT,
          VPE_SRC_PLANAR_YUV420, /* Src Format */
          VPE_DST_PACKET_RGB565, /* Dst Format */
          0);
vpeIoctl(VPE_IOCTL_SET_SRC_OFFSET,
          0, /* Src Left offset */
          0, /* Src right offset */
          NULL);
vpeIoctl(VPE_IOCTL_SET_DST_OFFSET,
          0, /* Dst Left offset */
          0, /* Dst right offset */
          NULL);
vpeIoctl(VPE_IOCTL_SET_SRC_DIMENSION,
          2048,
          1536,
          NULL);
vpeIoctl(VPE_IOCTL_SET_DST_DIMENSION,
          640,
          480,
          NULL);
vpeIoctl(VPE_IOCTL_SET_COLOR_RANGE,
          FALSE, /* Soure image is full range */
          FALSE, /* Destination image is full range */
          NULL);
vpeIoctl(VPE_IOCTL_SET_FILTER,
          VPE_SCALE_BILINEAR,
          NULL,
          NULL);
vpeIoctl(VPE_IOCTL_SET_DST_OFFSET,
          0, /* left offset */
          0, /* Right offset */
          NULL);
vpeIoctl(VPE_IOCTL_SET_DSTBUF_ADDR,
          piDstAddr,
          NULL,

```

```

        NULL);
        vpeioctl(VPE_IOCTL_TRIGGER,
        NULL,
        NULL,
        NULL);
do{
    ERRCODE errcode;
    errcode = vpeioctl(VPE_IOCTL_CHECK_TRIGGER,
        NULL, NULL, NULL);
    if(errcode==0)
        break;
}while(1);
    
```

33.4 VPE Error Code Table

Code Name	Value	Description
ERR_VPE_OPEN	0xFFFF1B01	VPE has been opened
ERR_VPE_CLOSE	0xFFFF1002	VPE has been closed.
ERR_VPE_SRC_FMT	0xFFFF1003	Invalid source format
ERR_VPE_DST_FMT	0xFFFF1004	Invalid destination format
ERR_VPE_OP	0xFFFF1005	Invalid operation mode
ERR_VPE_IOCTL	0xFFFF1006	Invalid ioctl
E_VPE_INVALID_INT	0xFFFF1007	Invalid interrupt

Table 33-3 VPE Error Code Table

34 VPOST Library

34.1 VPOST Library Overview

Display Interface Controller VPOST (include LCD Controller & TV encoder Controller) is used to display the video/image data to LCD device or to generate the composite signal to the TV system. The LCD timing can be 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 and register settings. The frame buffer is stored in system memory (SDRAM). The TV picture and LCD picture can display individual image source simultaneously when the timing is synchronized with TV timing.

34.2 Definition

34.2.1 Constant

VPOST

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_RESOLUTINO_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

Name	Value	Description
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_DRVPOST_PARALLEL_SYNCLCM_INTERFACE		
eDRVPOST_PRGB_16BITS	0x0	16 pin parallel RGB data bus
eDRVPOST_PRGB_18BITS	0x1	18 pin parallel RGB data bus
eDRVPOST_PRGB_24BITS	0x2	24 pin parallel RGB data bus
E_DRVPOST_SYNCLCM_DATABUS		
eDRVPOST_SYNC_8BITS	0x0	8 bit sync-type LCD
eDRVPOST_SYNC_9BITS	0x1	9 bit sync-type LCD
eDRVPOST_SYNC_16BITS	0x2	16 bit sync-type LCD
eDRVPOST_SYNC_18BITS	0x3	18 bit sync-type LCD
eDRVPOST_SYNC_24BITS	0x4	24 bit sync-type LCD
E_DRVPOST_INT		
eDRVPOST_HINT	0x1	Horizontal interrupt
eDRVPOST_VINT	0x2	Vertical interrupt

Name	Value	Description
eDRVVPOST_TVFIELDINT	0x3	TV field interrupt
eDRVVPOST_MPUCPLINT	0x10	MPU complete interrupt
E_DRVVPPOST_MPULCM_DATABUS		
eDRVVPOST_MPU_8_8	0x0	Transfer in 8-8 format for 16 bit color in 8 bit bus width
eDRVVPOST_MPU_2_8_8	0x1	Transfer in 2-8-8 format for 18 bit color in 8 bit bus width
eDRVVPOST_MPU_6_6_6	0x2	Transfer in 6-6-6 format for 18 bit color in 8 bit bus width
eDRVVPOST_MPU_8_8_8	0x3	Transfer in 8-8-8 format for 24 bit color in 8 bit bus width
eDRVVPOST_MPU_9_9	0x4	Transfer in 9-9 format for 18 bit color in 9 bit bus width
eDRVVPOST_MPU_16	0x5	Transfer in 16 format for 16 bit color in 16 bit bus width
eDRVVPOST_MPU_16_2	0x6	Transfer in 16-2 format for 18 bit color in 16 bit bus width
eDRVVPOST_MPU_2_16	0x7	Transfer in 2-16 format for 18 bit color in 16 bit bus width
eDRVVPOST_MPU_16_8	0x8	Transfer in 16-8 format for 24 bit color in 16 bit bus width
eDRVVPOST_MPU_18	0x9	Transfer in 18 format for 18 bit color in 18 bit bus width
eDRVVPOST_MPU_18_6	0xA	Transfer in 18-6 format for 124 bit color in 18 bit bus width
eDRVVPOST_MPU_24	0xB	Transfer in 24 format for 24 bit color in 24 bit bus width
E_DRVVPPOST_FRAME_DATA_TYPE		
eDRVVPOST_FRAME_RGB555	0x0	RGB555 Frame buffer data format
eDRVVPOST_FRAME_RGB565	0x1	RGB565 Frame buffer data format
eDRVVPOST_FRAME_RGBX888	0x2	RGB_Dummy888 Frame buffer data format

Name	Value	Description
eDRVVPOST_FRAME_RGB888X	0x3	RGB888_Dummy Frame buffer data format
eDRVVPOST_FRAME_CBYCRY	0x4	Cb0Y0Cr0Y1 Frame buffer data format
eDRVVPOST_FRAME_YCBYCR	0x5	Y0Cb0Y1Cr0 Frame buffer data format
eDRVVPOST_FRAME_CRYCBY	0x6	Cr0Y0Cb0Y1 Frame buffer data format
eDRVVPOST_FRAME_YCRYCB	0x7	Y0Cr0Y1Cb0 Frame buffer data format
E_DRVPOST_DATABUS		
eDRVVPOST_DATA_8BITS	0x0	8 bits data bus
eDRVVPOST_DATA_9BITS	0x1	9 bits data bus
eDRVVPOST_DATA_16BITS	0x2	16 bits data bus
eDRVVPOST_DATA_18BITS	0x3	18 bits data bus
eDRVVPOST_DATA_24BITS	0x4	24 bits data bus

Table 34-1 VPOST Definition

34.2.2 Structure

LCDFORMATEX Structure

Field Name	Data Type	Description
ucVASrcFormat	UINT32	User input Display source format
nScreenWidth	UINT32	Driver outputLCD width
nScreenHeight	UINT32	Driver outputLCD height
nFrameBufferSize	UINT32	Driver outputFrame buffer size
ucROT90	UINT8	Rotate 90 degree or not

Table 34-2 LCDFORMATEX Structure Definition

S_DRVPOST_SYNCLCM_HTIMING Structure

Field Name	Data Type	Description
u8PulseWidth	UINT8	Horizontal sync pulse width
u16BackPorch	UINT16	Horizontal back porch
u16FrontPorch	UINT16	Horizontal front porch

Table 34-3 S_DRVPOST_SYNCLCM_HTIMING Structure Definition

S_DRVPOST_SYNCLCM_VTIMING Structure

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

Table 34-4 S_DRVPOST_SYNCLCM_VTIMING Structure Definition

S_DRVPOST_SYNCLCM_WINDOW Structure

Field Name	Data 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 34-5 S_DRVPOST_SYNCLCM_WINDOW Structure Definition

S_DRVPOST_SYNCLCM_POLARITY Structure

Field Name	Data Type	Description
blsVsyncActiveLow	BOOL	Vsync polarity
blsHsyncActiveLow	BOOL	Hsync polarity
blsVDenActiveLow	BOOL	VDEN polarity
blsDClockRisingEdge	BOOL	Clock polarity

Table 34-6 S_DRVPOST_SYNCLCM_POLARITY Structure Definition

S_DRVPOST_MPULCM_WINDOW Structure

Field Name	Data 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 34-7 S_DRVPOST_MPULCM_WINDOW Structure Definition

S_DRVPOST_MPULCM_TIMING Structure

Field Name	Data 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 34-8 S_DRVPOST_MPULCM_TIMING Structure Definition

S_DRVPOST_MPULCM_CTRL Structure

Field Name	Data 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 34-9 S_DRVPOST_MPULCM_CTRL Structure Definition

34.3 API function

vpostGetFrameBuffer

Synopsis

VOID* vpostGetFrameBuffer (VOID)

Description

Get the display frame buffer address

Parameter

None

Return Value

Display frame buffer address.

Example

None.

vpostSetFrameBuffer

Synopsis

```
VOID vpostSetFrameBuffer (
    UINT32 pFramebuf
)
```

Description

Set the display frame buffer address

Parameter

pFramebuf [in]
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.

vpostVAStartTrigger_MPUContinue

Synopsis

VOID vpostVAStartTrigger_MPUContinue (VOID)

Description

The function will send the frame data to MPU panel continuously.

Parameter

None

Return Value

None

Example

None.

vpostVAStartTrigger_MPUSingle

Synopsis

VOID vpostVAStartTrigger_MPUSingle (VOID)

Description

The function will trigger to send one frame data to MPU panel.

Parameter

None

Return Value

None

Example

None.

vpostVAStopTriggerMPU

Synopsis

VOID vpostVAStopTriggerMPU (VOID)

Description

The function will stop sending the frame data to MPU panel.

Parameter

None

Return Value

None

Example

None.

vpostsetLCM_TimingType

Synopsis

```
VOID vpostsetLCM_TimingType (
    E_DRVPOST_TIMING_TYPE eTimingTpye
)
```

Description

The function sets if LCD timing to be synchronous with TV timing.

Parameter

eTimingTpye [in]
Input the timing type.

Return Value

None

Example

```
vpostsetLCM_TimingType(eDRVPOST_SYNC_TV);
```

vpostsetLCM_TypeSelect

Synopsis

```
VOID vpostsetLCM_TypeSelect (
    DRVPOST_LCM_TYPE eType
)
```

Description

The function sets the LCD type.

Parameter

eType [in]
 Input the LCD type.

Return Value

None

Example

```
vpostsetLCM_TypeSelect (eDRVVPOST_HIGH_RESOLUTINO_SYNC);
```

vpostSetSerialSyncLCM_Interface

Synopsis

```
VOID vpostSetSerialSyncLCM_Interface (
    E_DRVVPOST_8BIT_SYNCLCM_INTERFACE eInterface
)
```

Description

The function sets 8-bit serial LCD interface.

Parameter

eInterface [in]
 Input the LCD interface.

Return Value

None

Example

```
vpostSetSerialSyncLCM_Interface(eDRVVPOST_SRGB_YUV422);
```

vpostSetParalelSyncLCM_Interface

Synopsis

```
VOID vpostSetParalelSyncLCM_Interface (
    E_DRVVPOST_PARALLEL_SYNCLCM_INTERFACE eInterface
)
```

Description

The function sets parallel LCD data bus interface.

Parameter

eInterface [in]
 Input the LCD interface.

Return Value

None

Example

```
vpostSetParalelSynclCM_Interface (eDRVVPOST_PRGB_16BITS);
```

vpostSetFrameBuffer_DataType

Synopsis

```
VOID vpostSetFrameBuffer_DataType (
    E_DRVVPOST_FRAME_DATA_TYPE eType
)
```

Description

The function sets the data type of frame buffer.

Parameter

eType [in] Input the data type.

Return Value

None

Example

```
vpostSetFrameBuffer_DataType (eDRVVPOST_FRAME_RGB565);
```

vpostSetFrameBuffer_BaseAddress

Synopsis

```
VOID vpostSetFrameBuffer_BaseAddress (
    UINT32 u32BufferAddress
)
```

Description

The function sets the base address of frame buffer.

Parameter

u32BufferAddress [in]
Input the base address.

Return Value

None

Example

```
vpostSetFrameBuffer_BaseAddress(0x5000000);
```

vpostSetYUVEndianSelect

Synopsis

```
VOID vpostSetYUVEndianSelect (
    E_DRVVPOST_ENDIAN eEndian
)
```

Description

The function sets the endian selection of frame buffer.

Parameter

eEndian [in]
Input the endian selection.

Return Value

None

Example

```
vpostSetYUVEndianSelect(eDRVVPOST_YUV_LITTLE_ENDIAN);
```

vpostSetDataBusPin

Synopsis

```
VOID vpostSetDataBusPin (
    E_DRVVPOST_DATABUS eDataBus
)
```

Description

The function sets the LCD data bus and related control signal pins.

Parameter

eDataBus [in]
Input the pin selection.

Return Value

None

Example

```
vpostSetDataBusPin(eDRVVPOST_PRGB_16BITS);
```

vpostSetSyncLCM_HTiming

Synopsis

```
VOID vpostSetSyncLCM_HTiming (
    S_DRVVPOST_SYNCLCM_HTIMING *psHTiming
)
```

Description

The function sets the horizontal timing of synchronized type LCD.

Parameter

psHTiming [in]

Input the pointer of timing setting table.

Return Value

None

Example

```
vpostSetSyncLCM_HTiming(&HTimingTable);
```

vpostSetSyncLCM_VTiming

Synopsis

```
VOID vpostSetSyncLCM_VTiming (
    S_DRVPOST_SYNCLCM_VTIMING *psVTiming
)
```

Description

The function sets the vertical timing of synchronized type LCD.

Parameter

psVTiming [in]

Input the pointer of timing setting table.

Return Value

None

Example

```
vpostSetSyncLCM_VTiming(&VTimingTable);
```

vpostSetSyncLCM_ImageWindow

Synopsis

```
VOID vpostSetSyncLCM_ImageWindow (
    S_DRVPOST_SYNCLCM_WINDOW *psWindow
)
```

Description

The function sets the image window of synchronized type LCD.

Parameter

psWindow [in]

Input the pointer of image window.

Return Value

None

Example

```
vpostSetSyncLCM_ImageWindow (&ImageWindow);
```

vpostSetSyncLCM_SignalPolarity

Synopsis

```
VOID vpostSetSyncLCM_SignalPolarity (
    S_DRVVPOST_SYNCLCM_POLARITY *psPolarity
)
```

Description

The function sets the polarity of control signal.

Parameter

psPolarity [in]

Input the pointer of polarity settings.

Return Value

None

Example

```
vpostSetSyncLCM_SignalPolarity (&PolaritySetting);
```

vpostSetLCM_ImageSource

Synopsis

```
VOID vpostSetLCM_ImageSource (
    E_DRVVPOST_IMAGE_SOURCE eSource
)
```

Description

The function sets the image source.

Parameter

eSource [in]

Input the image source.

Return Value

None

Example

```
vpostSetLCM_ImageSource(eDRVVPOST_FRAME_BUFFER);
```

vpostEnableInt

Synopsis

```
VOID vpostEnableInt (
    E_DRVVPOST_INT eInt
)
```

Description

Enable VPOST selected interrupt source.

Parameter

eInt [in]
Select enabled interrupt source.

Return Value

None

Example

```
vpostEnableInt(LCDCInt_VINTEN);
```

vpostDisableInt

Synopsis

```
VOID vpostDisableInt (
    E_DRVVPOST_INT eInt
)
```

Description

Disable VPOST selected interrupt source.

Parameter

eInt [in]
Select disabled interrupt source.

Return Value

None

Example

```
vpostDisableInt(LCDCInt_VINTEN);
```

vpostClearInt

Synopsis

```
VOID vpostClearInt (
    E_DRVVPOST_INT eInt
)
```

)

Description

Clear VPOST selected interrupt flag.

Parameter

eInt [in]

Select cleared interrupt flag.

Return Value

None

Example

```
vpostClearInt(LCDCInt_VINTEN);
```

vpostInstallCallback

Synopsis

int

```
vpostInstallCallback (
    E_DRVPOST_INT eIntSource,
    PFN_DRVPOST_INT_CALLBACK pfnCallback,
    PFN_DRVPOST_INT_CALLBACK *pfnOldCallback
)
```

)

Description

Install VPOST interrupt callback function.

Parameter

eIntSource [in]

Select interrupt source.

pfnCallback [in]

Install current callback function.

pfnOldCallback [out]

return old callback function pointer.

Return Value

0 Success

1 Fail

Example

```
vpostInstallCallback(eDRVPOST_VINT, pfnVpostCallback, &pfnOldCallback);
```

vpostSetMPULCM_ImageWindow

Synopsis

```
VOID vpostSetMPULCM_ImageWindow (
    S_DRVPOST_MPULCM_WINDOW *psWindow
)
```

Description

The function sets the image window of MPU type LCD.

Parameter

psWindow [in]
Input the pointer of image window.

Return Value

None

Example

```
vpostSetMPULCM_ImageWindow(&ImageWindow);
```

vpostSetMPULCM_TimingConfig

Synopsis

```
VOID
vpostSetMPULCM_TimingConfig (
    S_DRVPOST_MPULCM_TIMING *psTiming
)
```

Description

The function sets the timing settings of MPU type LCD.

Parameter

psTiming [in]
Input the pointer of timing settings.

Return Value

None

Example

```
vpostSetMPULCM_TimingConfig(&TimingSetting);
```

vpostSetMPULCM_BusModeSelect

Synopsis

```
VOID vpostSetMPULCM_BusModeSelect (
```

```
E_DRVPOST_MPULCM_DATABUS eBusMode
)
```

Description

The function sets the data bus mode of MPU type LCD.

Parameter

eBusMode[in]
Input the data bus mode.

Return Value

None

Example

```
vpostSetMPULCM_BusModeSelect(eDRVPOST_MPU_8_8);
```

vpostSetFrameBuffer_Size

Synopsis

```
VOID vpostSetFrameBuffer_Size (
    S_DRVPOST_FRAME_SIZE* psSize
)
```

Description

The function sets the frame size.

Parameter

psSize [in]
Input the pointer of frame size settings.

Return Value

None

Example

```
vpostSetFrameBuffer_Size(&FrameBufSize);
```

34.4 Error Code Table

Code Name	Value	Description
ERR_NULL_BUF	0xFFF06004	memory location error
ERR_NO_DEVICE	0xFFF06005	No device error
ERR_BAD_PARAMETER	0xFFF06006	Bad parameter error
ERR_POWER_STATE	0xFFF06007	Power state control error

Table 34-10 Error Code Table

35 Supporting Resources

The N9H26 system related issues can be posted in Nuvoton's forum:

- ARM7/9 forum at: <http://forum.nuvoton.com/viewforum.php?f=12>.

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.*