NuMicro™ NUC122 Series Driver Reference Guide

V1.00.002

Publication Release Date: May. 2011

Support Chips:

NuMicro™ NUC122

Support Platforms:

Nuvoton

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1. Overview

1.1. Organization

This document describes the NuMicro™ NUC122 series driver reference manual. System-level software developers can use the NuMicro™ NUC122 series driver to do the fast application software development, instead of using the register level programming, which can reduce the total development time significantly. In this document, a description, usage and an illustrated example code are provided for each driver application interface. The full driver samples and driver source codes can be found in the BSP (Board Support Package) of the NuMicro™ NUC122 series.

This document is organized into several chapters. Chapter 1 is an overview. From Chapter 2 to Chapter 12 are the detailed driver descriptions including the followings: System Driver, UART Driver, Timer Driver, GPIO Driver, SPI Driver, I2C Driver, RTC Driver, PWM Driver, PS2 Driver, FMC Driver and USB Driver.

Finally, for the NuMicro™ NUC122 series selection guide and product identity list are described in Appendix.

1.2. Relative Documents

User can find the following documents in our website for other relative information.

- NuMicro™ NUC122 series Technical Reference Manual (TRM)
- NuMicro™ NUC100 series Application Notes

1.3. Abbreviations and Glossaries

AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
BOD	Brown Out Detection
BUF	Buffer
CFG	Configuration
DSQ	Data Sequence
EP	End Point
FIFO	First-In-First-Out
FLD	Float-Detection
FMC	Flash Memory Controller
GPIO	General Purpose Input/Output
I2C	Inter Integrated Circuit
LIN	Local Interconnect Network
LVR	Low Voltage Reset
PDID	Product Device Identity
PHY	Physical layer
PLL	Phase-Locked Loop
POR	Power On Reset
PWM	Pulse-Width Modulation
PS/2	IBM Personal System/2
SPI	Serial Peripheral Interface
TOG	Toggle
TRIG	Trigger
UART	Universal Asynchronous Receiver/Transmitter

1.4. Data Type Definition

The definition of all basic data types used in our drivers follows the definition of ANSI C and compliant with ARM CMSIS (Cortex Microcontroller Software Interface Standard). The definitions of function-dependent enumeration types are defined in each chapter. The basic data types are listed as follows.

Type	Definition	Description
int8_t	singed char	8 bits signed integer
int16_t	signed short	16 bits signed integer
int32_t	signed int	32 bits signed integer
uint8_t	unsigned char	8 bits unsigned integer
uint16_t	unsigned short	16 bits unsigned integer
uint32_t	unsigned int	32 bits unsigned integer

2. SYS Driver

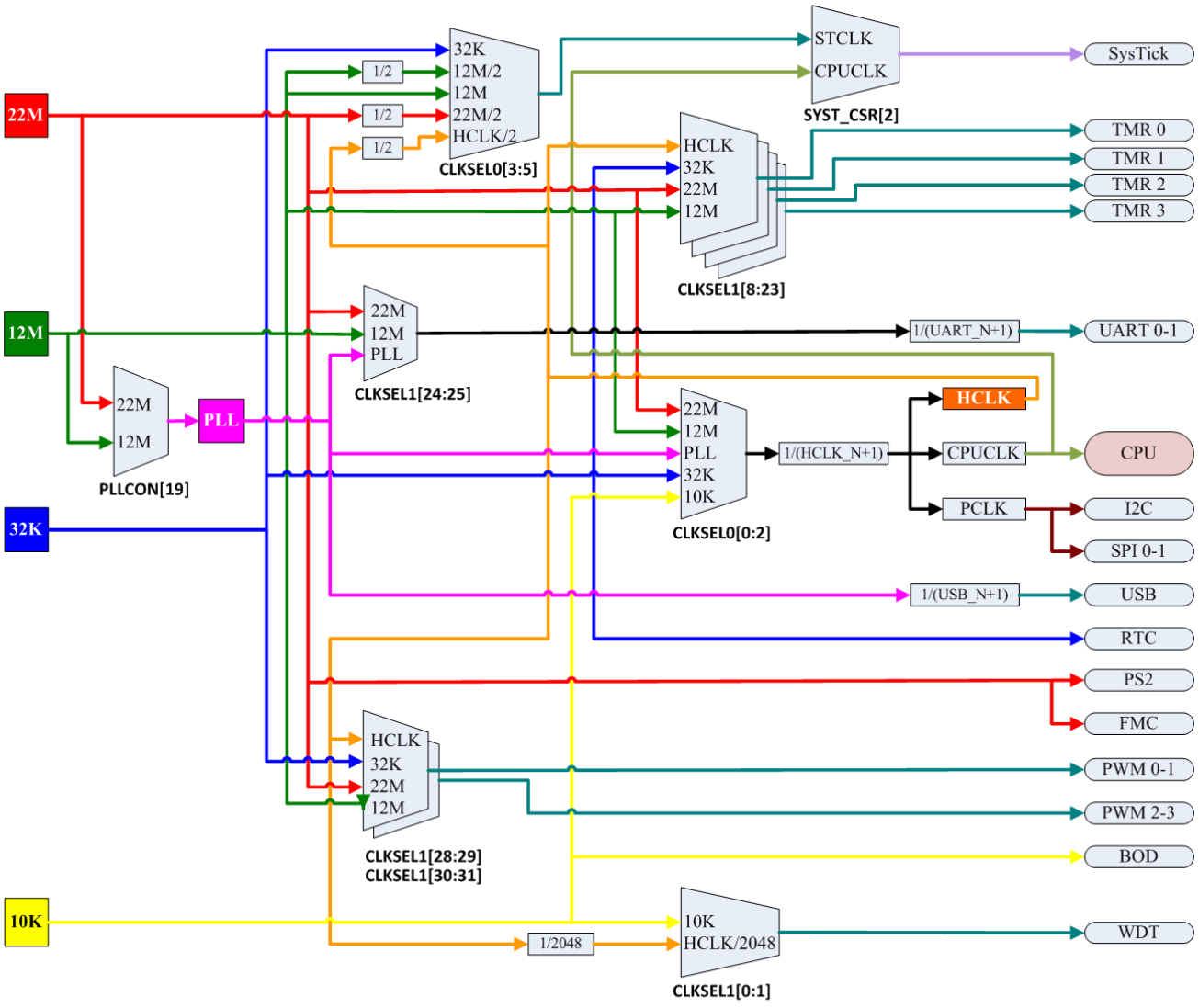
2.1. Introduction

The following functions are included in System Manager and Clock Controller section,

- Product Device ID
- System management registers for chip and module functional reset.
- Brown-Out and chip miscellaneous control.
- Clock generator
- System clock and peripherals clock
- Power down mode

2.2. Clock Diagram

The clock diagram shows all relative clocks for the whole chip, including system clocks (CPU clock, HCLK, and PCLK) and all peripheral clocks. Here, 12M means the external crystal clock source and it is connected with external crystal clock (4 ~ 24 MHz). 22M means internal 22 MHz RC clock source and its frequency is 22.1184 MHz with 3% deviation. 32K means the external 32.768 KHz crystal for RTC purpose. 10K means internal 10KHz RC clock source with 50% deviation.



2.3. Type Definition

E_SYS_IP_RST

Enumeration identifier	Value	Description
E_SYS_GPIO_RST	1	GPIO reset
E_SYS_TMR0_RST	2	Timer0 reset
E_SYS_TMR1_RST	3	Timer1 reset
E_SYS_TMR2_RST	4	Timer2 reset
E_SYS_TMR3_RST	5	Timer3 reset
E_SYS_I2C_RST	9	I2C reset
E_SYS_SPI0_RST	12	SPI0 reset
E_SYS_SPI1_RST	13	SPI1 reset
E_SYS_UART0_RST	16	UART0 reset
E_SYS_UART1_RST	17	UART1 reset
E_SYS_PWM03_RST	20	PWM0~3 reset
E_SYS_PS2_RST	23	PS2 reset
E_SYS_USBD_RST	27	USB Device reset

E_SYS_IP_CLK

Enumeration identifier	Value	Description
E_SYS_WDT_CLK	0	Watch Dog Timer clock enable control
E_SYS_RTC_CLK	1	RTC clock enable control
E_SYS_TMR0_CLK	2	Timer0 clock enable control
E_SYS_TMR1_CLK	3	Timer1 clock enable control
E_SYS_TMR2_CLK	4	Timer2 clock enable control
E_SYS_TMR3_CLK	5	Timer3 clock enable control
E_SYS_I2C_CLK	9	I2C clock enable control
E_SYS_SPI0_CLK	12	SPI0 clock enable control
E_SYS_SPI1_CLK	13	SPI1 clock enable control
E_SYS_UART0_CLK	16	UART0 clock enable control
E_SYS_UART1_CLK	17	UART1 clock enable control
E_SYS_PWM01_CLK	20	PWM01 clock enable control
E_SYS_PWM23_CLK	21	PWM23 clock enable control
E_SYS_USBD_CLK	27	USB device clock enable control
E_SYS_PS2_CLK	31	PS2 clock enable control
E_SYS_ISP_CLK	34	Flash ISP controller clock enable control

E_SYS_PLL_CLKSRC

Enumeration identifier	Value	Description
E_SYS_EXTERNAL_12M	0	PLL source clock is from external crystal clock (4~24 MHz)
E_SYS_INTERNAL_22M	1	PLL source clock is from internal 22.1184 MHz

E_SYS_IP_DIV

Enumeration identifier	Value	Description
E_SYS_UART_DIV	0	UART source clock divider setting
E_SYS_USBD_DIV	1	USBD source clock divider setting
E_SYS_HCLK_DIV	2	HCLK source clock divider setting

E_SYS_IP_CLKSRC

Enumeration identifier	Value	Description
E_SYS_WDT_CLKSRC	0	Watch Dog Timer clock source setting
E_SYS_TMR0_CLKSRC	1	Timer0 clock source setting
E_SYS_TMR1_CLKSRC	2	Timer1 clock source setting
E_SYS_TMR2_CLKSRC	3	Timer2 clock source setting
E_SYS_TMR3_CLKSRC	4	Timer3 clock source setting
E_SYS_UART_CLKSRC	5	UART clock source setting
E_SYS_PWM01_CLKSRC	6	PWM01 clock source setting
E_SYS_PWM23_CLKSRC	7	PWM23 clock source setting

E_SYS_CHIP_CLKSRC

Enumeration identifier	Value	Description
E_SYS_XTL12M	0	Select External Crystal Clock (4 ~ 24 MHz)
E_SYS_XTL32K	1	Select External 32K Crystal
E_SYS_OSC22M	2	Select Internal 22M Oscillator
E_SYS_OSC10K	3	Select Internal 10K Oscillator
E_SYS_PLL	4	Select PLL clock

E_SYS_PD_TYPE

Enumeration identifier	Value	Description
E_SYS_IMMEDIATE	0	Enter power down immediately
E_SYS_WAIT_FOR_CPU	1	Enter power down wait CPU sleep command

2.4. Functions

DrvSYS_ReadProductID

Prototype

uint32_t DrvSYS_ReadProductID (void);

Description

To read product device identity. The Product Device ID is depended on Chip part number. Please refer to PDID Table of Appendix in details.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

Product Device ID

Example

```
uint32_t u32data;
u32data = DrvSYS_ReadProductID ( );          /* Read Product Device ID */
```

DrvSYS_GetResetSource

Prototype

uint32_t DrvSYS_GetResetSource (void);

Description

To identify reset source from last operation. The corresponding reset source bits are listed in Register 'RSTSRC' of TRM in details.

Bit Number	Description
Bit 0	Power On Reset
Bit 1	RESET Pin
Bit 2	Watch Dog Timer
Bit 3	Low Voltage Reset
Bit 4	Brown-Out Detector Reset
Bit 5	Cortex M0 Kernel Reset
Bit 6	Reserved
Bit 7	CPU Reset

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The value in RSTSRC register.

Example

```
uint32_t u32data;
u32data = DrvSYS_GetResetSource ( );          /* Get reset source from last operation */
```

DrvSYS_ClearResetSource

Prototype

uint32_t DrvSYS_ClearResetSource (uint32_t u32Src);

Description

Clear reset source by writing a '1'.

Parameter

u32Src [in]

The corresponding bit of reset source.

Include

Driver/DrvSYS.h

Return Value

0 Succeed

Example

```
DrvSYS_ClearResetSource (1 << 3);     /* Clear Bit 3 (Low Voltage Reset) */
```

DrvSYS_ResetIP

Prototype

void DrvSYS_ResetIP (E_SYS_IP_RST eIpRst);

Description

To reset IP that includes GPIO, Timer0, Timer1, Timer2, Timer3, I2C, SPI0, SPI1, UART0, UART1, PWM03, PS2, and USBD.

Parameter

eIpRst [in]

Enumeration for IP reset, reference the E_SYS_IP_RST of Section 2.3.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_ResetIP (E_SYS_I2C_RST);          /* Reset I2C */
DrvSYS_ResetIP (E_SYS_SPI0_RST);         /* Reset SPI0 */
DrvSYS_ResetIP (E_SYS_UART0_RST);        /* Reset UART0 */
```

DrvSYS_ResetCPU

Prototype

```
void    DrvSYS_ResetCPU (void);
```

Description

To reset CPU. Software will set CPU_RST (IPRSTC1 [1]) to reset Cortex-M0 CPU kernel and Flash memory controller (FMC).

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_ResetCPU ( );       /* Reset CPU and FMC */
```

DrvSYS_ResetChip

Prototype

```
void    DrvSYS_ResetChip (void);
```

Description

To reset whole chip, including Cortex-M0 CPU kernel and all peripherals.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_ResetChip ( );       /* Reset whole chip */
```

DrvSYS_SelectBODVolt

Prototype

```
void     DrvSYS_SelectBODVolt (uint8_t u8Volt);
```

Description

To select Brown-Out threshold voltage.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u8Volt [in]

3: 4.5V, 2: 3.8V, 1: 2.7V, 0: 2.2V

Include

Driver/DrvSYS.h

Return Value

None.

Example

```
DrvSYS_SelectBODVolt (0);      /* Set Brown-Out Detector voltage 2.2V */
```

```
DrvSYS_SelectBODVolt (1);     /* Set Brown-Out Detector voltage 2.7V */
DrvSYS_SelectBODVolt (2);     /* Set Brown-Out Detector voltage 3.8V */
```

DrvSYS_SetBODFunction

Prototype

void DrvSYS_SetBODFunction (int32_t i32Enable, int32_t i32Flag, BOD_CALLBACK bodcallbackFn);

Description

To enable Brown-out detector and select Brown-out reset function or interrupt function. If Brown-Out interrupt function is selected, this will install call back function for BOD interrupt handler. When the voltage of AVDD Pin is lower than selected Brown-Out threshold voltage, Brown-out detector will reset chip or assert an interrupt. User can use DrvSYS_SelectBODVolt ( ) to select Brown-Out threshold voltage.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

i32Enable [in]

1: enable, 0: disable

i32Flag [in]

1: enable Brown-out reset function, 0: enable Brown-out interrupt function

bodcallbackFn [in]

Install Brown-Out call back function when interrupt function is enabled.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Enable Brown-Out Detector , select Brown-Out Interrupt function and install callback function 'BOD_CallbackFn' */
DrvSYS_SetBODFunction (1, 0, BOD_CallbackFn);
/* Enable Brown-Out Detector and select Brown-Out reset function */
DrvSYS_SetBODFunction (1, 1, NULL);
/* Disable Brown-Out Detector */
DrvSYS_SetBODFunction (0, 0, NULL);
```

DrvSYS_EnableBODLowPowerMode

Prototype

void DrvSYS_EnableBODLowPowerMode (void);

Description

To enable Brown-out Detector low power mode. The Brown-Out Detector consumes about 100uA in normal mode, the low power mode can reduce the current to about 1/10 but slow the Brown-Out Detector response.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_EnableBODLowPowerMode ( );      /* Enable Brown-Out low power mode */
```

DrvSYS_DisableBODLowPowerMode

Prototype

void DrvSYS_DisableBODLowPowerMode (void);

Description

To disable Brown-out Detector low power mode.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_DisableBODLowPowerMode ( ); /* Disable Brown-Out low power mode */

DrvSYS_EnableLowVoltReset

Prototype

void DrvSYS_EnableLowVoltReset (void);

Description

To enable low voltage reset function reset the chip when input voltage is lower than LVR circuit. The typical threshold is 2.0V. The characteristics of LVR threshold voltage is shown in DC Electrical Characteristics Section of TRM.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_EnableLowVoltRst ( ); /* Enable low voltage reset function */

DrvSYS_DisableLowVoltReset

Prototype

void DrvSYS_DisableLowVoltReset (void);

Description

To disable low voltage reset function.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_DisableLowVoltRst ( );      /* Disable low voltage reset function */
```

DrvSYS_GetBODState

Prototype

```
uint32_t DrvSYS_GetBODState (void);
```

Description

To get Brown-out Detector state.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

1: the detected voltage is lower than BOD threshold voltage.

0: the detected voltage is higher than BOD threshold voltage.

Example

```
uint32_t u32flag;
/* Get Brown-out state if Brown-out detector function is enabled */
u32flag = DrvSYS_GetBODState ( );
```

DrvSYS_UnlockProtectedReg

Prototype

```
int32_t      DrvSYS_UnlockProtectedReg (void);
```

Description

To unlock the protected registers. Some of the system control registers need to be protected to avoid inadvertent write and disturb the chip operation. These system control registers are locked after the power on reset. If user needs to modify these registers, user must UNLOCK them. These protected registers are listed in Register 'REGWRPROT' of System Manager Section of TRM in details.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

Example

```
int32_t i32ret;
/* Unlock protected registers */
i32ret = DrvSYS_UnlockProtectedReg ( );
```

DrvSYS_LockProtectedReg

Prototype

int32_t DrvSYS_LockProtectedReg (void);

Description

To re-lock the protected registers. Recommend user to re-lock the protected register after modifying these registers

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

Example

```
int32_t i32ret;
/* Lock protected registers */
i32ret = DrvSYS_LockProtectedReg ( );
```

DrvSYS_IsProtectedRegLocked

Prototype

int32_t DrvSYS_IsProtectedRegLocked (void);

Description

To check the protected registers are locked or not.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

1: The Protected Registers are unlocked.

0: The Protected Registers are locked.

Example

```
int32_t i32flag;
/* Check the protected registers are unlocked or not */
i32flag = DrvSYS_IsProtectedRegLocked ( );
If (i32flag)
   /* do something for unlock */
else
   /* do something for lock */
```

DrvSYS_EnablePOR

Prototype

void DrvSYS_EnablePOR (void);

Description

To re-enable power-on-reset control.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_EnablePOR ( ); /* Enable power-on-reset control */

DrvSYS_DisablePOR

Prototype

void DrvSYS_DisablePOR (void);

Description

To disable power-on-reset control. When power on, the POR circuit generates a reset signal to reset the whole chip function, but noise on the power may cause the POR active again. User can disable the POR control circuit for this condition.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_DisablePOR ( ); /* Disable power-on-reset control */

DrvSYS_SetIPClock

Prototype

void DrvSYS_SetIPClock (E_SYS_IP_CLK eIpClk, int32_t i32Enable);

Description

To enable or disable IP clock that includes Watch Dog Timer, RTC, Timer0, Timer1, Timer2, Timer3, I2C, SPI0, SPI1, UART0, UART1, PWM01, PWM23, USBD, PS2 and Flash ISP controller.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API to enable or disable the clock of Watch Dog Timer. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

eIpClk [in]

Enumeration for IP clock, reference the E_SYS_IP_CLK of Section 2.3.

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_SetIPClock (E_SYS_I2C_CLK, 1);       /* Enable I2C engine clock */
DrvSYS_SetIPClock (E_SYS_I2C_CLK, 0);       /* Disable I2C engine clock */
DrvSYS_SetIPClock (E_SYS_SPI0_CLK, 1);      /* Enable SPI0 engine clock */
DrvSYS_SetIPClock (E_SYS_SPI0_CLK, 0);      /* Disable SPI0 engine clock */
DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);      /* Enable TIMER0 engine clock */
DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 0);      /* Disable TIMER0 engine clock */
```

DrvSYS_SelectHCLKSource

Prototype

int32_t DrvSYS_SelectHCLKSource (uint8_t u8ClkSrcSel);

Description

To select HCLK clock source from external crystal clock (4 ~ 24 MHz), external 32K crystal clock, PLL clock, internal 10K oscillator clock, or internal 22M oscillator clock. Please refer to the Clock Diagram for HCLK usage in details.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u8ClkSrcSel [in]

0: External crystal clock (4 ~ 24 MHz)

1: External 32K clock

2: PLL clock

3: Internal 10K clock

7: Internal 22M clock

Include

Driver/DrvSYS.h

Return Value

0	Succeed
< 0	Incorrect parameter

Example

```
DrvSYS_SelectHCLKSource (0);     /* Change HCLK clock source to be external crystal */
DrvSYS_SelectHCLKSource (2);     /* Change HCLK clock source to be PLL */
```

DrvSYS_SelectSysTickSource

Prototype

int32_t DrvSYS_SelectSysTickSource (uint8_t u8ClkSrcSel);

Description

To select Cortex-M0 SysTick clock source from external crystal clock (4 ~ 24 MHz), external 32K crystal clock, external crystal clock (4 ~ 24 MHz)/2, HCLK/2, or internal 22M oscillator clock/2. The SysTick timer is a standard timer included by Cortex-M0.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u8ClkSrcSel [in]

0: External crystal clock (4 ~ 24 MHz)

1: External 32K clock

2: External crystal clock (4 ~ 24 MHz) / 2

3: HCLK / 2

7: Internal 22M clock / 2

Include

Driver/DrvSYS.h

Return Value

0	Succeed
< 0	Incorrect parameter

Example

```
DrvSYS_SelectSysTickSource (0);  /* Change SysTick clock source to be external crystal */
```

```
DrvSYS_SelectSysTickSource (3);    /* Change SysTick clock source to be HCLK / 2 */
```

DrvSYS_SelectIPClockSource

Prototype

int32_t DrvSYS_SelectIPClockSource (E_SYS_IP_CLKSRC eIpClkSrc, uint8_t u8ClkSrcSel);

Description

To select IP clock source that includes Watch Dog Timer, Timer 0~3, UART, PWM01 and PWM23. Please refer to the Clock Diagram for IP clock source. The settings of IP's corresponding clock source are listed in Registers 'CLKSEL1' and 'CLKSEL2' of TRM in details.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API to select the clock source of Watch Dog Timer. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked().

Parameter

eIpClkSrc [in]

E_SYS_WDT_CLKSRC / E_SYS_TMR0_CLKSRC / E_SYS_TMR1_CLKSRC
E_SYS_TMR2_CLKSRC / E_SYS_TMR3_CLKSRC / E_SYS_UART_CLKSRC
E_SYS_PWM01_CLKSRC / E_SYS_PWM23_CLKSRC

u8ClkSrcSel [in]

IP's corresponding clock source.

u8ClkSrcSel	0	1	2	3	7
Watch Dog Timer	Reserved	Reserved	HCLK/2048	Internal 10K	X
Timer	External 12M	External 32K	HCLK	Reserved	Internal 22M
UART	External 12M	PLL	Reserved	Internal 22M	X
PWM	External 12M	External 32K	HCLK	Internal 22M	X

Include

Driver/DrvSYS.h

Return Value

0 Succeed

< 0 Incorrect parameter

Example

```
/* Select UART clock source from external crystal clock (4 ~ 24 MHz) */
DrvSYS_SelectIPClockSource (E_SYS_UART_CLKSRC, 0x00);
/* Select TIMER0 clock source from HCLK */
DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0x02);
```

DrvSYS_SetClockDivider

Prototype

int32_t DrvSYS_SetClockDivider (E_SYS_IP_DIV eIpDiv, uint8_t u8value);

Description

To set IP engine clock divide number from IP clock source.

The IP clock frequency is calculated by:

IP clock source frequency / (u8value + 1).

Parameter

eIpDiv [in]

E_SYS_UART_DIV / E_SYS_USBD_DIV / E_SYS_HCLK_DIV

i32value [in]

Divide number: 0~15

Include

Driver/DrvSYS.h

Return Value

0	Succeed
< 0	Incorrect parameter

Example

```
/* Set UART clock divide number 0x02; UART clock = UART source clock / (2+1) */
DrvSYS_SetClockDivider (E_SYS_UART_DIV, 0x02);
/* Set HCLK clock divide number 0x03; HCLK clock = HCLK source clock / (3+1) */
DrvSYS_SetIPClockSource (E_SYS_HCLK_DIV, 0x03);
```

DrvSYS_SetOscCtrl

Prototype

int32_t DrvSYS_SetOscCtrl (E_SYS_CHIP_CLKSRC eClkSrc, int32_t i32Enable);

Description

To enable or disable internal oscillator and external crystal include internal 10K and 22M oscillator, or external 32K and 4~24 MHz crystal.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

eOscCtrl [in]

E_SYS_XTL12M / E_SYS_XTL32K / E_SYS_OSC22M / E_SYS_OSC10K.

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

0	Succeed
< 0	Incorrect parameter

Example

```
DrvSYS_SetOscCtrl (E_SYS_XTL12M, 1); /* Enable external crystal clock (4 ~ 24 MHz) */
DrvSYS_SetOscCtrl (E_SYS_XTL12M, 0); /* Disable external crystal clock (4 ~ 24 MHz) */
```

DrvSYS_SetPowerDownWakeUpInt

Prototype

void DrvSYS_SetPowerDownWakeUpInt (int32_t i32Enable, PWRWU_CALLBACK pdwucallbackFn, int32_t i32enWUDelay);

Description

To enable or disable power down wake up interrupt function, and install its callback function if power down wake up is enable, and enable clock cycles delay to wait the system clock stable. The delayed clock cycle is 4096 clock cycles when chip work at external 4~24 MHz crystal, or 256 clock cycles when chip work at internal 22.1184 MHz oscillator. The power down wake up interrupt will occur when GPIO, USB, UART, WDT, BOD or RTC wakeup.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

i32Enable [in]

1: enable, 0: disable

pdwucallbackFn [in]

Install power down wake up call back function when interrupt function is enabled.

i32enWUDelay [in]

1: enable clock cycles delay, 0: disable clock cycles delay

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Enable Power down Wake up Interrupt function, install callback function
'PWRWU_CallbackFn', and enable clock cycles delay */

DrvSYS_SetPowerDownWakeUpInt (1, PWRWU_CallbackFn, 1);

/* Disable Power down Wake up Interrupt function, and uninstall callback function */

DrvSYS_SetPowerDownWakeUpInt (0, NULL, 0);
```

DrvSYS_EnterPowerDown

Prototype

void DrvSYS_EnterPowerDown (E_SYS_PD_TYPE ePDType);

Description

To enter system power down mode immediately or after CPU enters sleep mode. When chip enters power down mode, the LDO, external crystal clock (4 ~ 24 MHz), and 22M oscillator will be disabled. Please refer to Application Note, *AN_1007_EN_Power_Management*, for application.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

ePDType [in]

E_SYS_IMMEDIATE: Chip enters power down mode immediately.

E_SYS_WAIT_FOR_CPU: Chip keeps active till the CPU sleep mode is also active and then the chip enters power down mode.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Chip enter power mode immediately */
DrvSYS_EnterPowerDown (E_SYS_ IMMEDIATE);
/* Wait for CPU enters sleep mode, then Chip enter power mode */
DrvSYS_EnterPowerDown (E_SYS_WAIT_FOR_CPU);
```

DrvSYS_SelectPLLSource

Prototype

```
void    DrvSYS_SelectPLLSource (E_SYS_PLL_CLKSRC ePllSrc);
```

Description

To select PLL clock source include internal 22M oscillator and external crystal clock (4 ~ 24 MHz).

Parameter

ePllSrc [in]

E_SYS_EXTERNAL_12M / E_SYS_INTERNAL_22M

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Select PLL clock source from external crystal clock (4 ~ 24 MHz) */
DrvSYS_ SelectPLLSource (E_SYS_EXTERNAL_12M);
/* Select PLL clock source from 22M */
DrvSYS_ SelectPLLSource (E_SYS_INTERNAL_22M);
```

DrvSYS_SetPLLMode

Prototype

```
void    DrvSYS_SetPLLMode (int32_t i32Flag);
```

Description

To set PLL operate in power down mode or normal mode.

Parameter

i32Flag [in]

1: PLL is in power down mode.

0: PLL is in normal mode.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Enable PLL power down mode, PLL operates in power down mode */
DrvSYS_SetPLLMode (1);
/* Disable PLL power down mode, PLL operates in normal mode */
DrvSYS_SetPLLMode (0);
```

DrvSYS_GetExtClockFreq

Prototype

uint32_t DrvSYS_GetExtClockFreq (void);

Description

To get external crystal clock frequency. The unit is in Hz.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The external crystal clock frequency

Example

```
uint32_t u32clock;
u32clock = DrvSYS_GetExtClockFreq ( ); /* Get external crystal clock frequency */
```

DrvSYS_GetPLLContent

Prototype

uint32_t DrvSYS_GetPLLContent(E_SYS_PLL_CLKSRC ePllSrc, uint32_t u32PllClk);

Description

To calculate the nearest PLL frequency to fit the target PLL frequency that is defined by u32PllClk.

Parameter

ePllSrc [in]

E_SYS_EXTERNAL_12M / E_SYS_INTERNAL_22M

u32PllClk [in]

The target PLL clock frequency. The unit is in Hz. The range of u32PllClk is 25 MHz ~ 500 MHz.

Include

Driver/DrvSYS.h

Return Value

The PLL control register setting.

Example

```
uint32_t u32PllCr;

/* Get PLL control register setting for target PLL clock 50 MHz */

u32PllCr = DrvSYS_GetPLLContent (E_SYS_EXTERNAL_12M, 50000000);
```

DrvSYS_SetPLLContent

Prototype

```
void    DrvSYS_SetPLLContent (uint32_t u32PllContent);
```

Description

To set PLL settings. User can use DrvSYS_GetPLLContent ( ) to get proper PLL setting and use DrvSYS_GetPLLClockFreq ( ) to get actual PLL clock frequency.

Parameter

u32PllContent [in]

The PLL register setting for the target PLL clock frequency.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
uint32_t u32PllCr;

/* Get PLL control register setting for target PLL clock 50 MHz */
```

```
u32PllCr = DrvSYS_GetPLLContent (E_DRVSYS_EXTERNAL_12M, 50000000);
/* Set PLL control register setting to get nearest PLL clock */
DrvSYS_SetPLLContent (u32PllCr);
```

DrvSYS_GetPLLClockFreq

Prototype

```
uint32_t    DrvSYS_GetPLLClockFreq (void);
```

Description

To get PLL clock output frequency.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The PLL clock output frequency in Hz

Example

```
uint32_t u32clock;
u32clock = DrvSYS_GetPLLClockFreq ( );      /* Get actual PLL clock */
```

DrvSYS_GetHCLKFreq

Prototype

```
uint32_t      DrvSYS_GetHCLKFreq (void);
```

Description

To get HCLK clock frequency.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The HCLK clock frequency in Hz

Example

```
uint32_t u32clock;
```

```
u32clock = DrvSYS_GetHCLKFreq ( );      /* Get current HCLK clock */
```

DrvSYS_Open

Prototype

int32_t DrvSYS_Open (uint32_t u32Hclk);

Description

To configure the PLL setting according to the PLL source clock and target HCLK clock. Due to hardware limitation, the actual HCLK clock may be different to target HCLK clock.

The DrvSYS_GetPLLClockFreq ( ) could be used to get actual PLL clock.

The DrvSYS_GetHCLKFreq ( ) could be used to get actual HCLK clock.

The DrvSYS_SetClockDivider ( ) could be used to get lower HCLK clock.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u32Hclk [in]

The target HCLK clock frequency. The unit is in Hz. The range of u32Hclk is 25 MHz ~ 60 MHz.

Include

Driver/DrvSYS.h

Return Value

E_SUCCESS	Succeed
E_DRVSYS_ERR_OUT_OF_RANGE	The clock setting is out of range
E_DRVSYS_ERR_REG_PROTECTED	The Write Protection function is enabled

Example

```
/* Set PLL clock 60 MHz, and switch HCLK source clock to PLL */
DrvSYS_Open (60000000);
```

DrvSYS_EnableHighPerformanceMode

Prototype

void DrvSYS_EnableHighPerformanceMode (void);

Description

To enable chip high performance mode. When this function is enable, internal RAM and GPIO access is working with zero wait state.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Enable high performance mode */
DrvSYS_EnableHighPerformanceMode ( );
```

DrvSYS_DisableHighPerformanceMode

Prototype

```
void      DrvSYS_DisableHighPerformanceMode (void);
```

Description

To disable chip high performance mode.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Disable high performance mode */
DrvSYS_DisableHighPerformanceMode ( );
```

DrvSYS_Delay

Prototype

void DrvSYS_Delay (uint32_t us);

Description

Use the SysTick timer of Cortex-M0 to generate the delay time and the unit is in us. The SysTick clock source is default to be from HCLK clock. If the SysTick clock source is changed by user, the delay time may be not correct.

Parameter

us [in]

Delay time. The maximal delay time is 279000 us.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_Delay (5000);    /* Delay 5000us */
```

DrvSYS_GetChipClockSourceStatus

Prototype

int32_t DrvSYS_GetChipClockSourceStatus (E_SYS_CHIP_CLKSRC eClkSrc);

Description

To monitor if the chip clock source is stable or not. The chip clock source includes internal 10K, 22M oscillator, external 32K, 4~24M crystal, or PLL clock.

Parameter

eClkSrc [in]

E_SYS_XTL12M / E_SYS_XTL32K / E_SYS_OSC22M / E_SYS_OSC10K / E_SYS_PLL

Include

Driver/DrvSYS.h

Return Value

0	Clock source is not stable or not enabled
1	Clock source is stable
< 0	Incorrect parameter

Example

```
/* Enable external crystal clock (4 ~ 24 MHz) */
DrvSYS_SetOscCtrl (E_SYS_XTL12M, 1);
/* Waiting for External Crystal stable */
while (DrvSYS_GetChipClockSourceStatus (E_SYS_XTL12M) != 1);
/* Disable PLL power down mode */
DrvSYS_SetPLLMode (0);
/* Waiting for PLL clock stable */
while (DrvSYS_GetChipClockSourceStatus (E_SYS_PLL) != 1);
```

DrvSYS_GetClockSwitchStatus

Prototype

uint32_t DrvSYS_GetClockSwitchStatus (void);

Description

To get if switch target clock is successful or failed when software switches system clock source.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

0: Clock switch success

1: Clock switch fail

Example

```
uint32_t u32flag;
DrvSYS_SelectHCLKSource (2);      /* Change HCLK clock source to be PLL */
u32flag = DrvSYS_GetClockSwitchStatus ( );      /* Get clock switch flag */
If (u32flag)
    /* do something for clock switch fail */
```

DrvSYS_ClearClockSwitchStatus

Prototype

void DrvSYS_ClearClockSwitchStatus (void);

Description

To clear the Clock Switch Fail Flag.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
uint32_t u32flag;
DrvSYS_SelectHCLKSource (0);      /* Change HCLK clock source to be external crystal */
u32flag = DrvSYS_GetClockSwitchStatus ( );      /* Get clock switch fail flag */
if (u32flag)
    DrvSYS_ClearClockSwitchStatus ( );     /* Clear clock switch fail flag */
```

DrvSYS_GetVersion

Prototype

uint32_t DrvSYS_GetVersion (void);

Description

Get this version of DrvSYS driver.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

Version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

3. UART Driver

3.1. UART Introduction

The Universal Asynchronous Receiver/Transmitter (UART) performs a serial-to-parallel conversion on data characters received from the peripheral such as MODEM, and a parallel-to-serial conversion on data characters received from the CPU.

Details please refer to the section in the target chip specification titled UART.

3.2. UART Feature

The UART includes following features:

- 64 bytes(UART0)/16 bytes(UART1) entry FIFOs for received and transmitted data payloads
- Auto flow control/flow control function (CTS, RTS) are supported.
- Fully programmable serial-interface characteristics:
  -- 5-, 6-, 7-, or 8-bit character
  -- Even, odd, or no-parity bit generation and detection
  -- 1-, 1&1/2, or 2-stop bit generation
  -- Baud rate generation
  -- False start bit detection.
- Full-prioritized interrupt system controls
- Loop back mode for internal diagnostic testing
- Support IrDA SIR Function
- Support LIN (Local interconnect network) master mode.
- Programmable baud-rate generator that allows the clock to be divided by programmable divider

3.3. Constant Definition

Constant Name	Value	Description
MODE_TX	1	IRDA or LIN function transmit mode
MODE_RX	2	IRDA or LIN function Recevie mode

3.4. Type Definition

E_UART_PORT

Enumeration identifier	Value	Description
UART_PORT0	0x000	UART port 0
UART_PORT1	0x100000	UART port 1

E_INT_SOURCE

Enumeration identifier	Value	Description
DRVUART_RDAINT	0x1	Receive Data Available Interrupt and Time-out Interrupt
DRVUART_THREINT	0x2	Transmit Holding Register Empty Interrupt
DRVUART_WAKEUPINT	0x40	Wake up interrupt enable
DRVUART_RLSINT	0x4	Receive Line Interrupt
DRVUART_MOSINT	0x8	MODEM Interrupt
DRVUART_TOUTINT	0x10	Time-out Interrupt.
DRVUART_BUFERRINT	0x20	Buffer Error Interrupt Enable
DRVUART_LININT	0x100	LIN RX Break Field Detected Interrupt Enable

E_DATABITS_SETTINGS

Enumeration identifier	Value	Description
DRVUART_DATABITS_5	0x0	Word length select: Character length is 5 bits.
DRVUART_DATABITS_6	0x1	Word length select: Character length is 6 bits.
DRVUART_DATABITS_7	0x2	Word length select: Character length is 7 bits.
DRVUART_DATABITS_8	0x3	Word length select: Character length is 8 bits.

E_PARITY_SETTINGS

Enumeration identifier	Value	Description
DRVUART_PARITY_NONE	0x0	None parity
DRVUART_PARITY_ODD	0x1	Odd parity enable

DRVUART_PARITY_EVEN	0x3	Even parity enable
DRVUART_PARITY_MARK	0x5	Parity mask
DRVUART_PARITY_SPACE	0x7	Parity space

E_STOPBITS_SETTINGS

Enumeration identifier	Value	Description
DRVUART_STOPBITS_1	0x0	Number of stop bit: Stop bit length is 1 bit.
DRVUART_STOPBITS_1_5	0x1	Number of stop bit: Stop bit length is 1.5 bit when character length is 5 bits.
DRVUART_STOPBITS_2	0x1	Number of stop bit: Stop bit length is 2 bit when character length is 6, 7 or 8 bits.

E_FIFO_SETTINGS

Enumeration identifier	Value	Description
DRVUART_FIFO_1BYTES	0x0	RX FIFO interrupt trigger level is 1 byte
DRVUART_FIFO_4BYTES	0x1	RX FIFO interrupt trigger level is 4 bytes
DRVUART_FIFO_8BYTES	0x2	RX FIFO interrupt trigger level is 8 bytes
DRVUART_FIFO_14BYTES	0x3	RX FIFO interrupt trigger level is 14 bytes
DRVUART_FIFO_30BYTES	0x4	RX FIFO interrupt trigger level is 30 bytes
DRVUART_FIFO_46BYTES	0x5	RX FIFO interrupt trigger level is 46 bytes
DRVUART_FIFO_62BYTES	0x6	RX FIFO interrupt trigger level is 62 bytes

E_UART_FUNC

Enumeration identifier	Value	Description
FUN_UART	0	Select UART function
FUN_LIN	1	Select LIN function
FUN_IRDA	2	Select IrDA function
FUN_RS485	3	Select RS485 function

E_MODE_RS485

Enumeration identifier	Value	Description
MODE_RS485_NMM	1	RS-485 Normal Multidrop Operation Mode
MODE_RS485_AAD	2	RS-485 Auto Address Detection Operation Mode
MODE_RS485_AUD	4	RS-485 Auto Direction Mode

3.5. Macros

_DRVUART_SENDBYTE

Prototype

void _DRVUART_SENDBYTE (u32Port, byData);

Description

Send 1 byte data from UART.

Include

Driver/DrvUART.h

Return Value

None.

Example

```
/* Using UART port0 to send one byte 0x55 */
_DRVUART_SENDBYTE (UART_PORT0, 0x55);
```

_DRVUART_RECEIVEBYTE

Prototype

uint8_t _DRVUART_RECEIVEBYTE (u32Port);

Description

Receive 1 byte data from specified UART FIFO.

Include

Driver/DrvUART.h

Return Value

One byte data.

Example

```
/* Using UART port0 to receive one byte */
uint8_t u8data;
u8data = _DRVUART_RECEIVEBYTE (UART_PORT0);
```

_DRVUART_SET_DIVIDER

Prototype

void _DRVUART_SET_DIVIDER (u32Port, u16Divider);

Description

To set the UART divider to control UART baud-rate

Include

Driver/DrvUART.h

Return Value

None.

Example

```
/* Set the divider of UART is 6 */
_DRVUART_SET_DIVIDER (UART_PORT0, 6);
```

_DRVUART_RECEIVEAVAILABLE

Prototype

int8_t _DRVUART_RECEIVEAVAILABLE (u32Port);

Description

To get current Rx FIFO pointer

Include

Driver/DrvUART.h

Return Value

Rx FIFO pointer value.

Example

```
/* To get UART channel 0 current Rx FIFO pointer */
_DRVUART_RECEIVEAVAILABLE (UART_PORT0);
```

_DRVUART_WAIT_TX_EMPTY

Prototype

void _DRVUART_WAIT_TX_EMPTY (u32Port);

Description

Polling Tx empty flag to check Tx FIFO is empty.

Include

Driver/DrvUART.h

Return Value

Return Tx empty flag status.

Example

```
/* Send 0x55 from UART0 and check TX FIFO is empty */
_DRVUART_SENDBYTE (UART_PORT0, 0x55);
_DRVUART_WAIT_TX_EMPTY (UART_PORT0);
```

3.6. Functions

DrvUART_Open

Prototype

```
int32_t
DrvUART_Open (
    E_UART_PORT u32Port,
    UART_T *sParam
);
```

Description

The function is used to initialize UART. It consists of baud-rate, parity, data-bits, stop-bits, rx-trigger-level and timeout interval settings.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

sParam [in]

Specify the property of UART. It includes

u32BaudRate: Baud rate (Hz)

u8cParity: NONE/EVEN/ODD parity

It could be

DRVUART_PARITY_NONE (None parity).

DRVUART_PARITY_EVEN (Even parity)

DRVUART_PARITY_ODD(Odd parity).

u8cDataBits: data bit setting

It could be

DRVUART_DATA_BITS_5 (5 data bits).

DRVUART_DATA_BITS_6 (6 data bits)

DRVUART_DATA_BITS_7 (7 data bits).

DRVUART_DATA_BITS_8 (8 data bits).

u8cStopBits: stop bits setting

It could be

DRVUART_STOPBITS_1 (1 stop bit).

DRVUART_STOPBITS_1_5 (1.5 stop bit)

DRVUART_STOPBITS_2 (2 stop bits).

u8cRxTriggerLevel: Rx FIFO interrupt trigger Level

LEVEL_X_BYTE means the trigger level of UART channel is X bytes

It could be

DRVUART_FIFO_1BYTE, DRVUART_FIFO_4BYTES

DRVUART_FIFO_8BYTES, DRVUART_FIFO_14BYTES

DRVUART_FIFO_30BYTES, DRVUART_FIFO_46BYTES

DRVUART_FIFO_62BYTES

In UART0 , it could be LEVEL_1_BYTE to LEVEL_62_BYTES.

Others, it could be LEVEL_1_BYTE to LEVEL_14_BYTES.

u8TimeOut: Time out value "N". It represents N-clock cycle and the counting clock is baud rate.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success.

E_DRVUART_ERR_PORT_INVALID: Wrong UART port configure

E_DRVUART_ERR_PARITY_INVALID: Wrong party setting

E_DRVUART_ERR_DATA_BITS_INVALID: Wrong Data bit setting

E_DRVUART_ERR_STOP_BITS_INVALID: Wrong Stop bit setting

E_DRVUART_ERR_TRIGGERLEVEL_INVALID: Wrong trigger level setting

Example

```
/* Set UART0 under 115200bps, 8 data bits ,1 stop bit and none parity and 1 byte Rx trigger level settings. */

STR_UART_T sParam;

sParam.u32BaudRate      = 115200;

sParam.u8cDataBits      = DRVUART_DATABITS_8;

sParam.u8cStopBits      = DRVUART_STOPBITS_1;
```

```
sParam.u8cParity            = DRVUART_PARITY_NONE;
sParam.u8cRxTriggerLevel    = DRVUART_FIFO_1BYTES;
DrvUART_Open (UART_PORT0, &sParam);
```

DrvUART_Close

Prototype

```
void DrvUART_Close (
    E_UART_PORT u32Port
);
```

Description

The function is used to disable UART clock, disable ISR and clear callback function pointer after checking the TX empty.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

Include

Driver/DrvUART.h

Return Value

None

Example

```
/* Close UART channel 0 */
DrvUART_Close (UART_PORT0);
```

DrvUART_EnableInt

Prototype

```
void      DrvUART_EnableInt (
    E_UART_PORT u32Port,
    uint32_t          u32InterruptFlag,
    PFN_DRVUART_CALLBACK pfncallback
);
```

Description

The function is used to enable specified UART interrupt, install the callback function and enable NVIC UART IRQ.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

pfncallback [in]

Call back function pointer

Include

Driver/DrvUART.h

Return Value

None

Note

Use **"|"** to connect the interrupt flags to enable multiple interrupts simultaneously.

If you call the function twice in a project, the settings is depend on the second setting.

Example

```
/* Enable UART channel 0 RDA and THRE interrupt. Finally, install UART_INT_HANDLE function to be callback function. */

DrvUART_EnableInt(UART_PORT0, (DRVUART_RDAINT | DRVUART_THREINT ),UART_INT_HANDLE);
```

DrvUART_DisableInt

Prototype

```
void      DrvUART_DisableInt (
   E_UART_PORT u32Port
   uint32_t          u32InterruptFlag
```

);

Description

The function is used to disable UART specified interrupt, uninstall the call back function and disable NVIC UART IRQ.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

None

Note

Use **"|"** to connect the interrupt flags to disable multiple interrupts simultaneously.

Example

```
/* To disable the THRE interrupt enable flag. */
DrvUART_DisableInt (UART_PORT0, DRVUART_THREINT);
```

DrvUART_ClearIntFlag

Prototype

```
uint32_t
DrvUART_ClearIntFlag (
    E_UART_PORT u32Port
    uint32_t        u32InterruptFlag
);
```

Description

The function is used to clear UART specified interrupt flag.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt.

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

E_SUCESS Success

Example

```
/* To disable the THRE interrupt enable flag. */
DrvUART_DisableInt (UART_PORT0, DRVUART_THREINT);
```

DrvUART_GetIntStatus

Prototype

```
int32_t
DrvUART_GetIntStatus (
    E_UART_PORT u32Port
    uint32_t    u32InterruptFlag
);
```

Description

The function is used to get the specified UART interrupt status.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt.

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

0: The specified interrupt did not happen.

1: The specified interrupt happened.

E_DRVUART_ARGUMENT: Error Parameter.

Note

It is recommended to poll one interrupt at a time.

Example

```
/* To get the THRE interrupt enable flag. */
If(DrvUART_GetIntStatus (UART_PORT0, DRVUART_THREINT))
   printf("THRE INT is happened!\n");
else
   printf("THRE INT is not happened or error parameter\n");
```

DrvUART_GetCTS

Prototype

```
void
DrvUART_GetCTS(
   E_UART_PORT          u32Port,
   uint8_t              *pu8CTSValue,
   uint8_t              *pu8CTSChangeState
}
```

Description

The function is used to get CTS pin value and detect CTS change state

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

pu8CTSValue [out]

Specify the buffer to receive the CTS value.Retrun current CTS pin state.

pu8CTSChangeState [out]

Specify the buffer to receive the CTS change state. Return CTS pin state is changed or not. 1 means changed and 0 means not yet.

Include

Driver/DrvUART.h

Return Value

None

Example

```
/* To get CTS pin status and save to u8CTS_value. To get detect CTS change flag and save to u8CTS_state. */

uint8_t u8CTS_value, u8CTS_state;

DrvUART_GetCTS (UART_PORT1,& u8CTS_value,& u8CTS_state);
```

DrvUART_SetRTS

Prototype

```
void
DrvUART_SetRTS (
    E_UART_PORT u32Port,
    uint8_t     u8Value,
    uint16_t    u16TriggerLevel
}
```

Description

The function is used to set RTS setting.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

u8Value [in]

Set 0: Drive RTS pin to logic 1 (If the LEV_RTS set to low level triggered).

Drive RTS pin to logic 0 (If the LEV_RTS set to high level triggered).
Set 1: Drive RTS pin to logic 0 (If the LEV_RTS set to low level triggered).
Drive RTS pin to logic 1 (If the LEV_RTS set to high level triggered).
Note. LEV_RTS is RTS Trigger Level. 0 is low level and 1 is high level.

u16TriggerLevel [in]

RTS Trigger Level :DRVUART_FIFO_1BYTES to DRVUART_FIFO_62BYTES

Include

Driver/DrvUART.h

Return Value

None

Example

```
/* Condition: Drive RTS to logic 1 in UART channel 1 and Set RTS trigger level is 1 bytes*/
DrvUART_SetRTS (UART_PORT1,1, DRVUART_FIFO_1BYTES);
```

DrvUART_Read

Prototype

```
int32_t
DrvUART_Read (
    E_UART_PORT         u32Port
    uint8_t             *pu8RxBuf,
    uint32_t            u32ReadBytes
);
```

Description

The function is used to read Rx data from RX FIFO and the data will be stored in pu8RxBuf.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

pu8RxBuf [out]

Specify the buffer to receive the data of receive FIFO.

u32ReadBytes [in]

Specify the read bytes number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success.

E_DRVUART_TIMEOUT: FIFO polling timeout.

Example

```
/* Condition: Read RX FIFO 1 byte and store in bInChar buffer.    */
uint8_t bInChar[1];
DrvUART_Read(UART_PORT0,bInChar,1);
```

DrvUART_Write

Prototype

```
int32_t
DrvUART_Write(
    E_UART_PORT u32Port
    uint8_t             *pu8TxBuf,
    uint32_t            u32WriteBytes
);
```

Description

The function is to write data into TX buffer to transmit data by UART

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

pu8TxBuf [in]

Specify the buffer to send the data to UART transmission FIFO.

u32WriteBytes [in]

Specify the byte number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success

E_DRVUART_TIMEOUT: FIFO polling timeout

Example

```
/* Condition: Send 1 byte from bInChar buffer to TX FIFO. */
uint8_t bInChar[1] = 0x55;
DrvUART_Write(UART_PORT0,bInChar,1);
```

DrvUART_SetFnIRDA

Prototype

```
void
DrvUART_SetFnIRDA (
    E_UART_PORT u32Port
    STR_IRCR_T      str_IRCR
);
```

Description

The function is used to configure IRDA relative settings. It consists of TX or RX mode and Inverse TX or RX signals.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

str_IRCR [in]

The structure of IrDA

It includes of

u8cTXSelect : 1 : Enable IrDA transmit function. It becomes TX mode

0 : Disable IrDA transmit function. It becomes RX mode.

u8cInvTX : Invert Tx signal function TRUE or FASLE

u8cInvRX : Invert Rx signal function (Default value is TRUE) TRUE or FASLE

Include

Driver/DrvUART.h

Return Value

None

Note

Before using the API, you should configure UART setting firstly. And make sure the baud-rate setting is used mode 0 (UART divider is 16)in baud-rate configure.

Example

```
/* Change UART1 to IRDA function and Inverse the RX signals. */
STR_IRCR_T sIrda;
sIrda.u8cTXSelect    = ENABLE;
sIrda.u8cInvTX       = FALSE;
sIrda.u8cInvRX       = TRUE;
DrvUART_SetFnIRDA(UART_PORT1,&sIrda);
```

DrvUART_SetFnRS485

Prototype

void

DrvUART_OpenRS485 (

E_UART_PORT u32Port,

STR_RS485_T *str_RS485

);

Description

The function is used to set RS485 relative setting

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

str_RS485 [in]

The structure of RS485

It includes of

u8cModeSelect: Select operation mode

MODE_RS485_NMM: RS-485 Normal Multi-drop Mode

MODE_RS485_AAD: RS-485 Auto Address Detection Mode

MODE_RS485_AUD: RS-485 Auto Direction Mode

u8cAddrEnable: Enable or Disable RS-485 Address Detection

u8cAddrValue: Set Address match value

u8cDelayTime: Set transmit delay time value

u8cRxDisable: Enable or Disable receiver function.

Include

Driver/DrvUART.h

Return Value

None

Note

None

Example

/* Condition: Change UART1 to RS485 function. Set relative setting as below.*/

```
STR_RS485_T sParam_RS485;
sParam_RS485.u8cAddrEnable      = ENABLE;
sParam_RS485.u8cAddrValue       = 0xC0;              /* Address */
sParam_RS485.u8cModeSelect      = MODE_RS485_AAD|MODE_RS485_AUD;
sParam_RS485.u8cDelayTime       = 0;
sParam_RS485.u8cRxDisable       = TRUE;
DrvUART_SetFnRS485(UART_PORT1,&sParam_RS485);
```

DrvUART_SetFnLIN

Prototype

```
void
DrvUART_SetFnLIN (
    E_UART_PORT u32Port
    uint16_t u16Mode,
    uint16_t u16BreakLength
);
```

Description

The function is used to set LIN relative setting

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1

u16Mode [in]

Specify LIN direction : MODE_TX and/or MODE_RX

u16BreakLength [in]

Specify break count value. It should be larger than 13 bit time according LIN protocol.

Include

Driver/DrvUART.h

Return Value

None

Example

```
/* Change UART1 to LIN function and set to transmit the header information. */
DrvUART_SetFnLIN(uart_ch,MODE_TX | MODE_RX,13);
```

DrvUART_GetVersion

Prototype

int32_t

DrvUART_GetVersion (void);

Description

Return the current version number of driver.

Include

Driver/DrvUART.h

Return Value

Version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

4. TIMER/WDT Driver

4.1. TIMER/WDT Introduction

The timer module includes four channels, TIMER0~TIMER3, which allow you to easily implement a counting scheme for use. The timer can perform functions like frequency measurement, event counting, interval measurement, clock generation, delay timing, and so on. The timer can generate an interrupt signal upon timeout, or provide the current value of count during operation. And also support external timer counting mode.

The purpose of Watchdog Timer (WDT) is to perform a system reset after the software running into a problem. This prevents system from hanging for an infinite period of time.

4.2. TIMER/WDT Feature

- 4 sets of 32-bit timers with 24-bit up-timer and one 8-bit pre-scale counter.
- Independent clock source for each timer.
- Provides one-shot, periodic, toggle and continuous counting operation modes.
- Time out period =
  (Period of timer clock input) * (8-bit pre-scale counter + 1) * (24-bit TCMP).
- Maximum counting cycle time = (1 / T MHz) * (2^8) * (2^24), T is the period of timer clock.
- 24-bit timer value is readable through TDR (Timer Data Register).
- Support event counting function to count the event from external pin.
- 18-bit free running counter to avoid CPU from Watchdog timer reset before the delay time expires.
- Selectable time-out interval (2^4 ~ 2^18) and the time out interval is 104 ms ~ 26.3168 s (if WDT_CLK = 10 kHz).
- Reset period = (1/10 kHz) * 63, if WDT_CLK = 10 kHz.

4.3. Type Definition

E_TIMER_CHANNEL

Enumeration Identifier	Value	Description
E_TMR0	0x0	Specify the timer channel - 0
E_TMR1	0x1	Specify the timer channel - 1
E_TMR2	0x2	Specify the timer channel - 2

E_TMR3	0x3	Specify the timer channel - 3

E_TIMER_OPMODE

Enumeration Identifier	Value	Description
E_ONESHOT_MODE	0x0	Set timer to One-Shot mode
E_PERIODIC_MODE	0x1	Set timer to Periodic mode
E_TOGGLE_MODE	0x2	Set timer to Toggle mode
E_CONTINUOUS_MODE	0x3	Set timer to Continuous Counting mode

E_WDT_CMD

Enumeration Identifier	Value	Description
E_WDT_IOC_START_TIMER	0x0	Start WDT counting
E_WDT_IOC_STOP_TIMER	0x1	Stop WDT counting
E_WDT_IOC_ENABLE_INT	0x2	Enable WDT interrupt
E_WDT_IOC_DISABLE_INT	0x3	Disable WDT interrupt
E_WDT_IOC_ENABLE_WAKEUP	0x4	Enable WDT time-out wake up function
E_WDT_IOC_DISABLE_WAKEUP	0x5	Disable WDT time-out wake up function
E_WDT_IOC_RESET_TIMER	0x6	Reset WDT counter
E_WDT_IOC_ENABLE_RESET_FUNC	0x7	Enable WDT reset function when WDT time-out
E_WDT_IOC_DISABLE_RESET_FUNC	0x8	Disable WDT reset function when WDT time-out
E_WDT_IOC_SET_INTERVAL	0x9	Set the WDT time-out interval

E_WDT_INTERVAL

Enumeration Identifier	Value	Description
E_LEVEL0	0x0	Set WDT time-out interval is 2^4 WDT_CLK
E_LEVEL1	0x1	Set WDT time-out interval is 2^6 WDT_CLK
E_LEVEL2	0x2	Set WDT time-out interval is 2^8 WDT_CLK
E_LEVEL3	0x3	Set WDT time-out interval is 2^10 WDT_CLK
E_LEVEL4	0x4	Set WDT time-out interval is 2^12 WDT_CLK
E_LEVEL5	0x5	Set WDT time-out interval is 2^14 WDT_CLK

E_LEVEL6	0x6	Set WDT time-out interval is 2^16 WDT_CLK
E_LEVEL7	0x7	Set WDT time-out interval is 2^18 WDT_CLK

4.4. Functions

DrvTIMER_Init

Prototype

void DrvTIMER_Init (void)

Description

User must to call this function before any timer operations after system boot up.

Parameter

None

Include

Driver/DrvTIMER.h

Return Value

None

Example:

```
/* Info the system can accept Timer APIs after calling DrvTIMER_Init() */
DrvTIMER_Init ();
```

DrvTIMER_Open

Prototype

```
int32_t DrvTIMER_Open (
        E_TIMER_CHANNEL ch,
        uint32_t        uTicksPerSecond,
        E_TIMER_OPMODE  op_mode
        )
```

Description

Open the specified timer channel with specified operation mode.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

uTicksPerSecond [in]

This value means how many timer interrupt ticks in one second

op_moode [in]

E_TIMER_OPMODE, E_ONESHOT_MODE / E_PERIODIC_MODE / E_TOGGLE_MODE / E_CONTINUOUS_MODE

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

E_DRVTIMER_CLOCK_RATE: Calculate initial value fail

Example

```
/* Using TIMER0 at PERIODIC_MODE, 2 ticks / sec */
   DrvTIMER_Open (E_TMR0, 2, E_PERIODIC_MODE);
```

DrvTIMER_Close

Prototype

int32_t DrvTIMER_Close (E_TIMER_CHANNEL ch)

Description

The function is used to close the timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Close the specified timer channel */
DrvTIMER_Close (E_TMR0);
```

DrvTIMER_SetTimerEvent

Prototype

```
int32_t DrvTIMER_SetTimerEvent (
        E_TIMER_CHANNEL ch,
        uint32_t        uInterruptTicks,
        TIMER_CALLBACK  pTimerCallback ,
        uint32_t        parameter
        )
```

Description

Install the interrupt callback function of the specified timer channel. And trigger timer callback function when interrupt occur *uInterruptTicks* times.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

uInterruptTicks [in]

Number of timer interrupt occurred

pTimerCallback [in]

The function pointer of the interrupt callback function

parameter [in]

A parameter of the callback function

Include

Driver/DrvTIMER.h

Return Value

uTimerEventNo:	The timer event number
E_DRVTIMER_EVENT_FULL:	The timer event is full

Example

```
/* Install callback "TMR_Callback" and trigger callback
      when timer interrupt happen twice */
uTimerEventNo = DrvTIMER_SetTimerEvent (E_TMR0, 2,
(TIMER_CALLBACK)TMR_Callback, 0);
```

DrvTIMER_ClearTimerEvent

Prototype

```
void DrvTIMER_ClearTimerEvent (
        E_TIMER_CHANNEL ch,
        uint32_t        uTimerEventNo
        )
```

Description

Clear the timer event of the specified timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

uTimerEventNo [in]

The timer event number

Include

Driver/DrvTIMER.h

Return Value

None

Example

```
/* Close the specified timer event */
DrvTIMER_ClearTimerEvent (E_TMR0, uTimerEventNo);
```

DrvTIMER_EnableInt

Prototype

int32_t DrvTIMER_EnableInt (E_TIMER_CHANNEL ch)

Description

This function is used to enable the specified timer interrupt.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Enable Timer-0 interrupt function */
   DrvTIMER_EnableInt (E_TMR0);
```

DrvTIMER_DisableInt

Prototype

int32_t DrvTIMER_DisableInt (E_TIMER_CHANNEL ch)

Description

This function is used to disable the specified timer interrupt.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Disable Timer-0 interrupt function */
   DrvTIMER_DisaleInt (E_TMR0);
```

DrvTIMER_GetIntFlag

Prototype

int32_t DrvTIMER_GetIntFlag (E_TIMER_CHANNEL ch)

Description

Get the interrupt flag status from the specified timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

iIntStatus: 0 is “No interrupt”, 1is “Interrupt occurred”

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Get the interrupt flag status from Timer-0 */
   u32TMR0IntFlag = DrvTIMER_GetIntFlag (E_TMR0);
```

DrvTIMER_ClearIntFlag

Prototype

int32_t DrvTIMER_ClearIntFlag (E_TIMER_CHANNEL ch)

Description

Clear the interrupt flag of the specified timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Clear Timer-0 interrupt flag */
   DrvTIMER_ClearIntFlag (E_TMR0);
```

DrvTIMER_Start

Prototype

int32_t DrvTIMER_Start (E_TIMER_CHANNEL ch)

Description

Start to count the specified timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Start to count the Timer-0 */
   DrvTIMER_Start (E_TMR0);
```

DrvTIMER_GetIntTicks

Prototype

uin32_t DrvTIMER_GetIntTicks (E_TIMER_CHANNEL ch)

Description

This function is used to get the number of interrupt occurred after the timer interrupt function is enabled.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

uTimerTick: Return the interrupt ticks

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Get the current interrupt ticks from Timer-1 */
   u32TMR1Ticks = DrvTIMER_GetIntTicks (E_TMR1);
```

DrvTIMER_ResetIntTicks

Prototype

int32_t DrvTIMER_ResetIntTicks (E_TIMER_CHANNEL ch)

Description

This function is used to clear interrupt ticks to 0.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Reset the interrupt ticks of Timer-1 to 0 */
   DrvTIMER_ResetIntTicks (E_TMR1);
```

DrvTIMER_Delay

Prototype

void DrvTIMER_Delay (E_TIMER_CHANNEL ch, uint32_t uIntTicks)

Description

This function is used to add a delay loop by specified interrupt ticks of the timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2 / E_TMR3

uIntTicks [in]

The delay ticks

Include

Driver/DrvTIMER.h

Return Value

None

Example

```
/* Delay Timer-0 3000 ticks */
   DrvTIMER_Delay (E_TMR0, 3000);
```

DrvTIMER_OpenCounter

Prototype

```
int32_t DrvTIMER_OpenCounter (
                E_TIMER_CHANNEL ch,
                uint32_t uCounterBoundary,
                E_TIMER_OPMODE op_mode
   );
```

Description

This function is used to open the timer channel with the specified operation mode.
And the counting source of timer is from the external event/counter.
The TIMER clock source should be set as HCLK.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2

uCounterBoundary [in]

The parameter is used to determine how many counts occurred will toggle once timer interrupt

op_mode [in]

E_TIMER_OPMODE, it's included E_ONESHOT_MODE / E_PERIODIC_MODE / E_CONTINUOUS_MODE

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

E_DRVTIMER_EIO: Timer has not been initialized

Example

```
/* Set Timer-0 run in One-Shot mode by external counter.
   And when the counter counting to 123, Timer-0 interrupt will occurred */
   DrvTIMER_OpenCounter (E_TMR0, 123, E_ONESHOT_MODE);
```

DrvTIMER_StartCounter

Prototype

in32_t DrvTIMER_StartCounter (E_TIMER_CHANNEL ch)

Description

Start counting of the specified timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Start to count the Timer-0 by external counter */
DrvTIMER_StartCounter (E_TMR0);
```

DrvTIMER_GetCounters

Prototype

uin32_t DrvTIMER_GetCounters (E_TIMER_CHANNEL ch)

Description

This function is used to get the current counters of the specified timer channel.

Parameter

ch [in]

E_TIMER_CHANNEL, it could be E_TMR0 / E_TMR1 / E_TMR2

Include

Driver/DrvTIMER.h

Return Value

u32Counters: Return current counters

E_DRVTIMER_CHANNEL: Invalid timer channel

Example:

```
/* Get the current counts of Timer-0 */
u32TMR0ExtTicks = DrvTIMER_GetCounters (E_TMR0);
```

DrvTIMER_GetVersion

Prototype

uint32_t DrvTIMER_GetVersion (void)

Description

Get the version number of Timer/WDT driver.

Include

Driver/DrvTIMER.h

Return Value

Version number :

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

```
/* Get the current version of Timer Driver */
u32Version = DrvTIMER_GetVersion ();
```

DrvWDT_Open

Prototype

int32_t DrvWDT_Open (E_WDT_INTERVAL WDTlevel)

Description

Enable WDT engine clock and set WDT time-out interval.
All bits in WDT register are write-protected. User must to check the REGWRPROT bit is enabled or disabled if write the specified WDT bit fail.

Parameter

WDTlevel [in]

E_WDT_INTERVAL, enumerate the WDT time-out interval. Refer to WDT_INTERVAL enumeration for detail time-out value.

Include

Driver/DrvTIMER.h

Return Value

None

Example

```
/* Set the WDT time-out interval is (2^16)*WDT_CLK */
DrvWDT_Open (E_WDT_LEVEL6);
```

DrvWDT_Close

Prototype

void DrvWDT_Close (void)

Description

The function is used to stop/disable WDT relative functions.
All bits in WDT register are write-protected. User must to check the REGWRPROT bit is enabled or disabled if write the specified WDT bit fail.

Parameter

None

Include

Driver/DrvTIMER.h

Return Value

None

Example

```
/* Close Watch Dog Timer */
   DrvWDT_Close ();
```

DrvWDT_InstallISR

Prototype

void DrvWDT_InstallISR (WDT_CALLBACK pvWDTISR)

Description

The function is used to install WDT interrupt service routine.
All bits in WDT register are write-protected. User must to check the REGWRPROT bit is enabled or disabled if write the specified WDT bit fail.

Parameter

pvWDTISR [in]

The function pointer of the interrupt service routine

Include

Driver/DrvTIMER.h

Return Value

None

Example

```
/* Install the WDT callback function */
   DrvWDT_InstallISR ((WDT_CALLBACK)WDT_Callback);
```

DrvWDT_Ioctl

Prototype

int32_T DrvWDT_Ioctl (E_WDT_CMD uWDTCmd, uint32_t uArgument)

Description

The function is used to operate more WDT applications, it could be the start/stop the WDT, enable/disable WDT interrupt function, enable/disable WDT time-out wake up function, enable/disable system reset when WDT time-out and set the WDT time-out interval.
All bits in WDT register are write-protected. User must to check the REGWRPROT bit is enabled or disabled if write the specified WDT bit fail.

Parameter

uWDTCmd [in]

E_WDT_CMD commands, it could be the one of the follow commands

E_WDT_IOC_START_TIMER，
E_WDT_IOC_STOP_TIMER，
E_WDT_IOC_ENABLE_INT，

E_WDT_IOC_DISABLE_INT，
E_WDT_IOC_ENABLE _WAKEUP，
E_WDT_IOC_DISABLE_WAKEUP，
E_WDT_IOC_RESET_TIMER，
E_WDT_IOC_ENABLE_RESET_FUNC，
E_WDT_IOC_DISABLE_RESET_FUNC，
E_WDT_IOC_SET_INTERVAL

uArgument [in]

Set the argument for the specified WDT command

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_ DRVWDT_CMD: Invalid WDT command

Example

```
/* Start to count WDT by calling WDT_IOC_START_TIMER command */
   DrvWDT_Ioctl (E_WDT_IOC_START_TIMER, 0);
```

5. GPIO Driver

5.1. GPIO introduction

NUC122 Series has up to 41 General Purpose I/O pins can be shared with other function pins; it depends on the chip configuration. These 41 pins are arranged in 4 ports named with GPIOA, GPIOB, GPIOC and GPIOD. Each port equips maximum 16 pins.

5.2. GPIO Feature

- Each one of the GPIO pins is independent and has the corresponding register bits to control the pin mode function and data.
- The I/O type of each of I/O pins can be independently software configured as input, output, open-drain or quasi-bidirectional mode.

5.3. Type Definition

E_DRVGPIO_PORT

Enumeration Identifier	Value	Description
E_GPA	0	Define GPIO Port A
E_GPB	1	Define GPIO Port B
E_GPC	2	Define GPIO Port C
E_GPD	3	Define GPIO Port D

E_DRVGPIO_IO

Enumeration Identifier	Value	Description
E_IO_INPIT	0	Set GPIO as Input mode
E_IO_OUTPUT	1	Set GPIO as Output mode
E_IO_OPENDRAIN	2	Set GPIO as Open-Drain mode
E_IO_QUASI	3	Set GPIO as Quasi-bidirectional mode

E_DRVGPIO_INT_TYPE

Enumeration Identifier	Value	Description

E_IO_RISING	0	Set interrupt enable by Rising Edge or Level High
E_IO_FALLING	1	Set interrupt enable by Falling Edge or Level Low
E_IO_BOTH_EDGE	2	Set interrupt enable by Both Edges(Rising and Falling)

E_DRVGPIO_INT_MODE

Enumeration Identifier	Value	Description
E_MODE_EDGE	0	Set interrupt mode is Edge trigger
E_MODE_LEVEL	1	Set interrupt mode is Level trigger

E_DRVGPIO_DBCLKSRC

Enumeration Identifier	Value	Description
E_DBCLKSRC_HCLK	0	De-bounce counter clock source is from HCLK
E_DBCLKSRC_10K	1	De-bounce counter clock source is from internal 10 KHz

E_DRVGPIO_FUNC

Enumeration Identifier	Pins assignment	Description
E_FUNC_GPIO	All GPIO pins	Set all GPIO pins as GPIO functions
E_FUNC_I2C1	GPA.10~11	Enable I2C1 function
E_FUNC_UART0	GPB.0~3	Enable UART0 RX, TX, RTS and CTS
E_FUNC_UART0_RX_TX	GPB.0~1	Enable UART0 RX, TX
E_FUNC_UART0_RTS_CTS	GPB.2~3	Enable UART0 RTS, CTS
E_FUNC_UART1	GPB.4~7	Enable UART1 RX, TX, RTS and CTS
E_FUNC_UART1_RX_TX	GPB.4~5	Enable UART1 RX, TX
E_FUNC_UART1_RTS_CTS	GPB.6~7	Enable UART1 RTS, CTS
E_FUNC_SPI0	GPC.0~3	Enable SPI0 SS0, CLK, MISO0 and MOSI0
E_FUNC_SPI0_SS1	GPB.10	Enable SPI0 SS1 function
E_FUNC_SPI1	GPC.8~11	Enable SPI1 SS0, CLK, MISO0 and MOSI0
E_FUNC_SPI1_SS1_PB9	GPB.9	Enable SPI1 SS1 function
E_FUNC_SPI1_SS1_PB4_QFN33	GPB.4	Enable GPB.4 as SPI1 SS1 function for QFN33 package only
E_FUNC_EXTINT0 / E_FUNC_EXTINT1	GPB.14 / GPB.15	Enable External INT0/INT1 functions
E_FUNC_TMR0 / E_FUNC_TMR1 / E_FUNC_TMR2	GPB.8~10	Enable TIMER0/TIMER1/TIMER2 as Toggle/Counter mode
E_FUNC_PWM01 / E_FUNC_PWM23 /	GPA.12~13 / GPA.14~15 /	Enable PWM01/PWM23 functions
E_FUNC_PWM0 / E_FUNC_PWM1 / E_FUNC_PWM2 / E_FUNC_PWM3	GPA.12 / GPA.13 / GPA.14 / GPA.15	Enable PWM0/PWM1/PWM2/PWM3

5.4. Macros

_DRVGPIO_DOUT

Prototype

_DRVGPIO_DOUT (PortNum, PinNum)

Description

This macro is used to control I/O Bit Output Control Register of the specified pin. User can set output data value of the specified pin by calling _DRVGPIO_DOUT macro, if the GPIO pin is configured as output mode. Or get the input data value by calling _DRVGPIO_DOUT directly, if the GPIO pin is configured as input mode.

Parameter

PortNum [in]

Specify the GPIO port. It could be 0~3 to correspond to the GPIO-A/B/C/D.

PinNum [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Example

```
/* Configure GPA-1 to output mode */
DrvGPIO_Open (E_GPA, 1, E_IO_OUTPUT);
/* Set GPA-1 to high */
_DRVGPIO_DOUT (E_GPA, 1) = 1;
/* ..... */
/* Configure GPB-3 to input mode */
uint8_t u8PinValue;
DrvGPIO_Open (E_GPB, 3, E_IO_INPUT);
/* Get GPB-3 pin value */
u8PinValue = _DRVGPIO_DOUT (E_GPB, 3);
```

GPA_[n] / GPB_[n] / GPC_[n] / GPD_[n]

Prototype

GPA_0~GPA_15 / GPB_0~GPB_15 / GPC_0~GPC_15 / GPD_0~GPD_15

Description

These macros are the same as _DRVGPIO_DOUT macro but without any parameters. User can use the macro define directly like GPA_0 to output data to the specified pin, or get pin value from this specified pin.

Parameter

None

Include

Driver/DrvGPIO.h

Example

```
/* Configure GPA-1 to output mode */
DrvGPIO_Open (E_GPA, 1, E_IO_OUTPUT);
/* Set GPA-1 to high */
GPA_1 = 1;
/* ..... */
/* Configure GPB-3 to input mode */
uint8_t u8PinValue;
DrvGPIO_Open (E_GPB, 3, E_IO_INPUT);
/* Get GPB-3 pin value */
u8PinValue = GPB_3;
```

5.5. Functions

DrvGPIO_Open

Prototype

```
int32_t DrvGPIO_Open (
        E_DRVGPIO_PORT  port,
        int32_t         i32Bit,
        E_DRVGPIO_IO    mode
        )
```

Description

Set the specified GPIO pin to the specified GPIO operation mode.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

mode [in]

E_DRVGPIO_IO, set the specified GPIO pin to be E_IO_INPUT, E_IO_OUTPUT, E_IO_OPENDRAIN or E_IO_QUASI mode.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Configure GPA-0 to GPIO output mode and GPA-1 to GPIO input mode*/
DrvGPIO_Open (E_GPA, 0, E_IO_OUTPUT);
DrvGPIO_Open (E_GPA, 1, E_IO_INPUT);
```

DrvGPIO_Close

Prototype

int32_t DrvGPIO_Close (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Close the specified GPIO pin function and set the pin to quasi-bidirectional mode.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Close GPA-0 function and set to default quasi-bidirectional mode */
DrvGPIO_Close (E_GPA, 0);
```

DrvGPIO_SetBit

Prototype

int32_t DrvGPIO_SetBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Set the specified GPIO pin to 1.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Configure GPA-0 as GPIO output mode*/
DrvGPIO_Open (E_GPA, 0, E_IO_OUTPUT);
/* Set GPA-0 to 1(high) */
DrvGPIO_SetBit (E_GPA, 0);
```

DrvGPIO_GetBit

Prototype

int32_t DrvGPIO_GetBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Get the pin value from the specified input GPIO pin.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

The specified input pin value: 0 / 1

E_DRVGPIO_ARGUMENT: ncorrect argument

Example

```
int32_t i32BitValue;
/* Configure GPA-1 as GPIO input mode*/
```

```
DrvGPIO_Open (E_GPA, 1, E_IO_INPUT);
i32BitValue = DrvGPIO_GetBit (E_GPA, 1);
if    (u32BitValue == 1)
{
      printf("GPA-1 pin status is high.\n");
}
else
{
      printf("GPA-1 pin status is low.\n");
}
```

DrvGPIO_ClrBit

Prototype

int32_t DrvGPIO_ClrBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Set the specified GPIO pin to 0.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect arguments

Example

```
/* Configure GPA-0 as GPIO output mode*/
DrvGPIO_Open (E_GPA, 0, E_IO_OUTPUT);
/* Set GPA-0 to 0(low) */
DrvGPIO_ClrBit (E_GPA, 0);
```

DrvGPIO_SetPortBits

Prototype

int32_tDrvGPIO_SetPortBits (E_DRVGPIO_PORT port, int32_t i32Data)

Description

Set the output port value to the specified GPIO port.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Data [in]

The data output value. It could be 0~0xFFFF.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Set the output value of GPA port to 0x1234 */
DrvGPIO_SetPortBits (E_GPA, 0x1234);
```

DrvGPIO_GetPortBits

Prototype

int32_t DrvGPIO_GetPortBits (E_DRVGPIO_PORT port)

Description

Get the input port value from the specified GPIO port.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

Include

Driver/DrvGPIO.h

Return Value

The specified input port value: 0 ~ 0xFFFF

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Get the GPA port input data value */
int32_t i32PortValue;
i32PortValue = DrvGPIO_GetPortBits (E_GPA);
```

DrvGPIO_GetDoutBit

Prototype

int32_t DrvGPIO_GetDoutBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Get the bit value from the specified Data Output Value Register. If the bit value is 1, it's meaning the pin is output data to high. Otherwise, it's output data to low.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified register: 0 / 1

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Get the GPA-1 data output value */
int32_t i32BitValue;
i32BitValue = DrvGPIO_GetDoutBit (E_GPA, 1);
```

DrvGPIO_GetPortDoutBits

Prototype

int32_t DrvGPIO_GetPortDoutBits (E_DRVGPIO_PORT port)

Description

Get the port value from the specified Data Output Value Register. If the corresponding bit of the return port value is 1, it means the corresponding bit is output data to high. Otherwise, it's output data to low.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

Include

Driver/DrvGPIO.h

Return Value

The portt value of the specified register: 0 ~ 0xFFFF

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Get the GPA port data output value */
int32_t i32PortValue;
i32PortValue = DrvGPIO_GetPortDoutBits (E_GPA);
```

DrvGPIO_SetBitMask

Prototype

int32_t DrvGPIO_SetBitMask (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

This function is used to protect the write data function of the corresponding GPIO pin. When set the bit mask, the write signal is masked and write data to the protect bit is ignored.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

```
/* Protect GPA-0 write data function */
DrvGPIO_SetBitMask (E_GPA, 0);
```

DrvGPIO_GetBitMask

Prototype

int32_t DrvGPIO_GetBitMask (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Get the bit value from the specified Data Output Write Mask Register. If the bit value is 1, it means the corresponding bit is protected. And write data to the bit is ignored.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified register: 0 / 1

Example

```
/* Get the bit value from GPA Data Output Write Mask Resister */
int32_t i32MaskValue;
i32MaskValue = DrvGPIO_GetBittMask (E_GPA, 0);
/* If (i32MaskValue = 1), its meaning GPA-0 is write protected */
```

DrvGPIO_ClrBitMask

Prototype

int32_t DrvGPIO_ClrBitMask (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

This function is used to remove the write protect function of the the corresponding GPIOpin. After remove the bit mask, write data to the corresponding bit is workable.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

```
/* Remove the GPA-0 write protect function */
DrvGPIO_ClrBitMask (E_GPA, 0);
```

DrvGPIO_SetPortMask

Prototype

int32_t DrvGPIO_SetPortMask (E_DRVGPIO_PORT port, int32_t i32MaskData)

Description

This function is used to protect the write data function of the corresponding GPIO pins.When set the bits are masked, write data to the protect bits are ignored.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32MaskData [in]

Specify pins of the GPIO port. It could be 0~0xFFFF.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Protect GPA-0/4 write data function */
DrvGPIO_SetPortMask (E_GPA, 0x11);
```

DrvGPIO_GetPortMask

Prototype

int32_t DrvGPIO_GetPortMask (E_DRVGPIO_PORT port)

Description

Get the port value from the specified Data Output Write Mask Register. If the corresponding bit of the return port value is 1, it's meaning the bits are protected. And write data to the bits are ignored.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

Include

Driver/DrvGPIO.h

Return Value

The portt value of the specified register: 0 ~ 0xFFFF

Example

```
/* Get the port value from GPA Data Output Write Mask Resister */
int32_t i32MaskValue;
i32MaskValue = DrvGPIO_GetPortMask (E_GPA);
/* If (i32MaskValue = 0x11), its meaning GPA-0/4 are protected */
```

DrvGPIO_ClrPortMask

Prototype

int32_t DrvGPIO_ClrPortMask (E_DRVGPIO_PORT port, int32_t i32MaskData)

Description

This function is used to remove the write protect function of the corresponding GPIO pins. After remove those bits mask, write data to the corresponding bits are workable.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32MaskData [in]

Specify pins of the GPIO port. It could be 0~0xFFFF.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

```
/* Remove the GPA-0/4 write protect function */
DrvGPIO_ClrPortMask (E_GPA, 0x11);
```

DrvGPIO_EnableDigitalInputBit

Prototype

```
int32_t DrvGPIO_EnableDigitalInputBit (
                E_DRVGPIO_PORT port,
                E_DRVGPIO_PIN i32Bit
            )
```

Description

Enable IO digital input path of the specified GPIO input pin.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port.
It could be E_GPA, E_GPB, E_GPC and E_GPD.

pin [in]

Specify pin of the GPIO port. It could be be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example:

```
/* Enable GPA.0 IO digital input path */
DrvGPIO_EnableDigitalInputBit (E_GPA, 0);
```

DrvGPIO_DisableDigitalInputBit

Prototype

```
int32_t DrvGPIO_DisableDigitalInputBit (
            E_DRVGPIO_PORT port,
            E_DRVGPIO_PIN i32Bit
        )
```

Description

Disable IO digital input path of the specified GPIO input pin.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port.
It could be E_GPA, E_GPB, E_GPC and E_GPD.

pin [in]

Specify pin of the GPIO port. It could be be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example:

```
/* Disable GPA.0 IO digital input path */
DrvGPIO_DisableDigitalInputBit (E_GPA, 0);
```

DrvGPIO_EnableDebounce

Prototype

int32_t DrvGPIO_EnableDebounce (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Enable the de-bounce function of the specified GPIO input pin.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

```
/* Enable GPA-0 interrupt de-bounce function */
DrvGPIO_EnableDebounce (E_GPA, 0);
```

DrvGPIO_DisableDebounce

Prototype

int32_t DrvGPIO_DisableDebounce (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Disable the de-bounce function of the specified GPIO input pin.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

```
/* Disable GPA-0 interrupt de-bounce function */
DrvGPIO_DisableDebounce (E_GPA, 0);
```

DrvGPIO_SetDebounceTime

Prototype

```
int32_t DrvGPIO_SetDebounceTime (
    uint32_t u32CycleSelection,
    E_DRVGPIO_DBCLKSRC ClockSource
)
```

Description

Set the interrupt de-bounce sampling time based on the de-bounce counter clock source. If the de-bounce clock source is from internal 10 KHz and sampling cycle selection is 4. The target de-bounce time is (2^4)*(1/(10*1000)) s = 16*0.0001 s = 1600 us, and system will sampling interrupt input once per 1600 us.

Parameter

u32CycleSelection [in]

The number of sampling cycle selection, the range of value is from 0 ~ 15. The target de-bounce time is (2^(u32CycleSelection))*(ClockSource) second.

ClockSource [in]

E_DRVGPIO_DBCLKSRC, it could be DBCLKSRC_HCLK or DBCLKSRC_10K.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Set de-bounce sampling time to 1600 us. (2^4)*(10 KHz) */
DrvGPIO_SetDebounceTime (4, E_DBCLKSRC_10K);
```

DrvGPIO_GetDebounceSampleCycle

Prototype

```
int32_t DrvGPIO_GetDebounceSampleCycle (void)
```

Description

This function is used to get the number of de-bounce sampling cycle selection.

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

Number of the sampling cycle selection: 0 ~ 15

Example

```
int32_t i32CycleSelection;
i32CycleSelection = DrvGPIO_GetDebounceSampleCycle ();
/* If i32CycleSelection is 4 and clock source from 10 KHz.          */
/* It's meaning to sample interrupt input once per 16*100us.    */
```

DrvGPIO_EnableInt

Prototype

```
int32_t DrvGPIO_EnableInt (
    E_DRVGPIO_PORT port,
    int32_t i32Bit,
    E_DRVGPIO_INT_TYPE TriggerType,
    E_DRVGPIO_INT_MODE Mode
)
```

Description

Enable the interrupt function of the specified GPIO pin. Excpet for GPB.14 and GPB.15 pins.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15. But the GPB.14/15 is only used for external interrupt 0/1.

TriggerType [in]

E_DRVGPIO_INT_TYPE, specify the interrupt trigger type. It could be E_IO_RISING, E_IO_FALLING or E_IO_BOTH_EDGE and it's meaning the interrupt function enable by rising edge/high level, falling edge/low level or both rising edge and falling edge. If the interrupt mode is E_MODE_LEVEL and interrupt type is E_BOTH_EDGE，then calling this API is ignored.

Mode [in]

E_DRVGPIO_INT_MODE, specify the interrupt mode. It could be E_MODE_EDGE or E_MODE_LEVEL to control the interrupt is by edge trigger or by level trigger. If the interrupt mode is E_MODE_LEVEL and interrupt type is E_BOTH_EDGE，then calling this API is ignored.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Enable GPB-13 interrupt function and its rising and edge trigger. */
DrvGPIO_EnableInt (E_GPB, 13, E_IO_RISING, E_MODE_EDGE);
```

DrvGPIO_DisableInt

Prototype

int32_t DrvGPIO_DisableInt (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Disable the interrupt function of the specified GPIO pin. Excpet for GPB.14 and GPB.15 pins.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

i32Bit [in]

Specify pin of the GPIO port. It could be 0~15. But the GPB.14/15 is only used for external interrupt 0/1.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

```
/* Disable GPB-13 interrupt function. */
DrvGPIO_DisableInt (E_GPB, 13);
```

DrvGPIO_SetIntCallback

Prototype

void DrvGPIO_SetIntCallback (

GPIO_GPAB_CALLBACK pfGPABCallback,

GPIO_GPCD_CALLBACK pfGPCDCallback

)

Description

Install the interrupt callback function for GPA/GPB port and GPC/GPD port. Excpet for GPB.14 and GPB.15 pins.

Parameter

pfGPABCallback [in], the function pointer of GPA/GPB callback function.
pfGPCDCallback [in], the function pointer of GPC/GPD callback function.

Include

Driver/DrvGPIO.h

Return Value

None

Example

```
/* Set GPA/B and GPC/D interrupt callback functions */
DrvGPIO_SetIntCallback (GPABCallback, GPCDCallback);
```

DrvGPIO_EnableEINT0

Prototype

void DrvGPIO_EnableEINT0 (

E_DRVGPIO_INT_TYPE TriggerType,

E_DRVGPIO_INT_MODE Mode,

GPIO_EINT0_CALLBACK pfEINT0Callback

)

Description

Enable the interrupt function for external GPIO interrupt from /INT0(GPB.14) pin.

Parameter

TriggerType [in]

E_DRVGPIO_INT_TYPE, specify the interrupt trigger type. It could be E_IO_RISING, E_IO_FALLING or E_IO_BOTH_EDGE and it's meaning the interrupt function enable by rising edge/high level, falling edge/low level or both rising edge and falling edge. If

the interrupt mode is E_MODE_LEVEL and interrupt type is E_BOTH_EDGE，then calling this API is ignored.

Mode [in]

E_DRVGPIO_INT_MODE, specify the interrupt mode. It could be E_MODE_EDGE or E_MODE_LEVEL to control the interrupt is by edge trigger or by level trigger. If the interrupt mode is E_MODE_LEVEL and interrupt type is E_BOTH_EDGE，then calling this API is ignored

pfEINT0Callback [in]

It's the function pointer of the external INT0 callback function.

Include

Driver/DrvGPIO.h

Return Value

None

Example

```
/* Enable external INT0 interrupt as falling and both-edge trigger. */
DrvGPIO_EnableEINT0 (E_IO_BOTH_EDGE, E_MODE_EDGE, EINT1Callback);
```

DrvGPIO_DisableEINT0

Prototype

void DrvGPIO_DisableEINT0 (void)

Description

Disable the interrupt function for external GPIO interrupt from /INT0 (GPB.14) pin.

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

None

Example

```
/* Disable external INT0 interrupt function. */
DrvGPIO_DisableEINT0 ();
```

DrvGPIO_EnableEINT1

Prototype

void DrvGPIO_EnableEINT1 (

E_DRVGPIO_INT_TYPE TriggerType,

E_DRVGPIO_INT_MODE Mode,

GPIO_EINT0_CALLBACK pfEINT0Callback

)

Description

Enable the interrupt function for external GPIO interrupt from /INT1(GPB.15) pin.

Parameter

TriggerType [in]

E_DRVGPIO_INT_TYPE, specify the interrupt trigger type. It could be E_IO_RISING, E_IO_FALLING or E_IO_BOTH_EDGE and it's meaning the interrupt function enable by rising edge/high level, falling edge/low level or both rising edge and falling edge. If the interrupt mode is E_MODE_LEVEL and interrupt type is E_BOTH_EDGE，then calling this API is ignored.

Mode [in]

E_DRVGPIO_INT_MODE, specify the interrupt mode. It could be E_MODE_EDGE or E_MODE_LEVEL L to control the interrupt is by edge trigger or by level trigger. If the interrupt mode is E_MODE_LEVEL and interrupt type is E_BOTH_EDGE，then calling this API is ignored

pfEINT1Callback [in]

It's the function pointer of the external INT1 callback function.

Include

Driver/DrvGPIO.h

Return Value

None

Example

```
/* Enable external INT1 interrupt as low level trigger. */
DrvGPIO_EnableEINT1 (E_IO_FALLING, E_MODE_LEVEL, EINT1Callback);
```

DrvGPIO_DisableEINT1

Prototype

void DrvGPIO_DisableEINT1 (void)

Description

Disable the interrupt function for external GPIO interrupt from /INT1(GPB.15) pin.

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

None

Example

```
/* Disable external INT1 interrupt function. */
DrvGPIO_DisableEINT1 ();
```

DrvGPIO_GetIntStatus

Prototype

uint32_t DrvGPIO_GetIntStatus (E_DRVGPIO_PORT port)

Description

Get the port value from the specified Interrupt Trigger Source Indicator Register.If the corresponding bit of the return port value is 1, it's meaning the interrupt occurred at the corresponding bit. Otherwise, no interrupt occurred at that bit.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port. It could be E_GPA, E_GPB, E_GPC and E_GPD.

Include

Driver/DrvGPIO.h

Return Value

The portt value of the specified register: 0 ~ 0xFFFF

Example

```
/* Get GPA interrupt status. */
int32_t i32INTStatus;
i32INTStatus = DrvGPIO_GetIntStatus (E_GPA);
```

DrvGPIO_InitFunction

Prototype

int32_t DrvGPIO_InitFunction (E_DRVGPIO_FUNC function)

Description

Initialize the specified function and configure the relative pins for specified function used.

Parameter

function [in]

DRVGPIO_FUNC, specified the relative GPIO pins as special function pins.
It could be:

E_FUNC_GPIO,
E_FUNC_I2C1,
E_FUNC_UART0,
E_FUNC_UART0_RX_TX,
E_FUNC_UART0_RTS_CTS,
E_FUNC_UART1,
E_FUNC_UART1_RX_TX,
E_FUNC_UART1_RTS_CTS,
E_FUNC_SPI0,
E_FUNC_SPI0_SS1,
E_FUNC_SPI1,
E_FUNC_SPI1_SS1_PB9,
E_FUNC_SPI1_SS1_PB4_QFN33,
E_FUNC_EXTINT0 / E_FUNC_EXTINT1,
E_FUNC_TMR0 / E_FUNC_TMR1 / E_FUNC_TMR2,
E_FUNC_PWM01 / E_FUNC_PWM23,
E_FUNC_PWM0 / E_FUNC_PWM1 / E_FUNC_PWM2 / E_FUNC_PWM3

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

```
/* Init UART0 RX, TX, RTS and CTS function */
DrvGPIO_InitFunction (E_FUNC_UART0);
```

DrvGPIO_GetVersion

Prototype

uint32_t DrvGPIO_GetVersion (void)

Description

This function is used to return the version number of GPIO driver.

Include

Driver/DrvGPIO.h

Return Value

The version number of GPIO driver:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

```
/* Get the current version of GPIO Driver */
int32_t i32GPIOVer;
i32GPIOVer = DrvGPIO_GetVersion ();
```

6. SPI Driver

6.1. SPI Introduction

The Serial Peripheral Interface (SPI) is a synchronous serial data communication protocol which operates in full duplex mode. Devices communicate in master/slave mode with 4-wire bi-direction interface. NuMicro™ NUC122 contains two sets of SPI controller performing a serial-to-parallel conversion on data received from a peripheral device, and a parallel-to-serial conversion on data transmitted to a peripheral device. Each SPI set can drive up to 2 external peripherals. It also can be driven as a slave device when the SLAVE bit (CNTRL[18]) is set.

Each controller can generate an individual interrupt signal when data transfer is finished and can be cleared by writing 1 to the respective interrupt flag. The active level of device/slave select signal can be programmed to low active or high active on SSR[SS_LVL] bit, which depends on the connected peripheral. Configure the DIVIDER register can program the frequency of serial clock output when it is as the master. If the VARCLK_EN bit in SPI_CNTRL[23] is enabled, the serial clock can be set as two programmable frequencies which are defined in DIVIDER and DIVIDER2. The format of the variable frequency is defined in VARCLK.

Each SPI controller contains two 32-bit transmission buffers (TX0 and TX1) and two reception buffers (RX0 and RX1), and can provide burst mode operation. It also supports variable length transfer.

In this document, we will introduce how to use the SPI driver.

6.2. SPI Feature

- Up to two sets of SPI controller.
- Support master/slave mode operation.
- Configurable data length of transfer word up to 32 bits.
- Variable output serial clock frequency in master mode.
- Provide burst mode operation, transmit/receive can be executed up to two times in one transfer.
- MSB or LSB first data transfer.
- 2 slave/device select lines in the master mode.
- Support Byte Reorder function.
- Compatible with Motorola SPI and National Semiconductor Microwire Bus.

6.3. Type Definition

E_DRVSPI_PORT

Enumeration Identifier	Value	Description
eDRVSPI_PORT0	0	SPI port 0
eDRVSPI_PORT1	1	SPI port 1

E_DRVSPI_MODE

Enumeration Identifier	Value	Description
eDRVSPI_MASTER	0	Master mode
eDRVSPI_SLAVE	1	Slave mode

E_DRVSPI_TRANS_TYPE

Enumeration Identifier	Value	Description
eDRVSPI_TYPE0	0	SPI transfer type 0
eDRVSPI_TYPE1	1	SPI transfer type 1
eDRVSPI_TYPE2	2	SPI transfer type 2
eDRVSPI_TYPE3	3	SPI transfer type 3
eDRVSPI_TYPE4	4	SPI transfer type 4
eDRVSPI_TYPE5	5	SPI transfer type 5
eDRVSPI_TYPE6	6	SPI transfer type 6
eDRVSPI_TYPE7	7	SPI transfer type 7

E_DRVSPI_ENDIAN

Enumeration Identifier	Value	Description
eDRVSPI_LSB_FIRST	0	Send LSB First
eDRVSPI_MSB_FIRST	1	Send MSB First

E_DRVSPI_BYTE_REORDER

Enumeration Identifier	Value	Description
eDRVSPI_BYTE_REORDER_SUSPEND_DISABLE	0	Both Byte Reorder function and Byte Suspend function are disabled
eDRVSPI_BYTE_REORDER_SUSPEND	1	Both Byte Reorder function and Byte Suspend function are enabled
eDRVSPI_BYTE_REORDER	2	Enable the Byte Reorder function
eDRVSPI_BYTE_SUSPEND	3	Enable the Byte Suspend function

E_DRVSPI_SSLTRIG

Enumeration Identifier	Value	Description
eDRVSPI_EDGE_TRIGGER	0	Edge trigger
eDRVSPI_LEVEL_TRIGGER	1	Level trigger

E_DRVSPI_SS_ACT_TYPE

Enumeration Identifier	Value	Description
eDRVSPI_ACTIVE_LOW_FALLING	0	Low-level/Falling-edge active
eDRVSPI_ACTIVE_HIGH_RISING	1	High-level/Rising-edge active

E_DRVSPI_SLAVE_SEL

Enumeration Identifier	Value	Description
eDRVSPI_NONE	0	No slave device was selected
eDRVSPI_SS0	1	Select the 1st slave select pin
eDRVSPI_SS1	2	Select the 2nd slave select pin
eDRVSPI_SS0_SS1	3	Both pins are selected

6.4. Functions

DrvSPI_Open

Prototype

```
int32_t DrvSPI_Open(
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_MODE eMode,
    E_DRVSPI_TRANS_TYPE eType,
    int32_t i32BitLength,
);
```

Description

This function is used to open SPI module. It decides the SPI to work in master or slave mode, SPI bus timing and bit length per transfer. The automatic slave select function will be enabled.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eMode [in]

To work in Master (eDRVSPI_MASTER) or Slave (eDRVSPI_SLAVE) mode

eType [in]

Transfer types, i.e. the bus timing. It could be eDRVSPI_TYPE0~eDRVSPI_TYPE7.

eDRVSPI_TYPE0: the clock idle state is low; drive data at rising-edge of serial clock; latch data at rising-edge of serial clock. Drive data and latch data at the same edge. Not recommend to use this transfer type.

eDRVSPI_TYPE1: the clock idle state is low; drive data at falling-edge of serial clock; latch data at rising-edge of serial clock.

eDRVSPI_TYPE2: the clock idle state is low; drive data at rising-edge of serial clock; latch data at falling-edge of serial clock.

eDRVSPI_TYPE3: the clock idle state is low; drive data at falling-edge of serial clock; latch data at falling-edge of serial clock. Drive data and latch data at the same edge. Not recommend to use this transfer type.

eDRVSPI_TYPE4: the clock idle state is high; drive data at rising-edge of serial clock; latch data at rising-edge of serial clock. Drive data and latch data at the same edge. Not recommend to use this transfer type.

eDRVSPI_TYPE5: the clock idle state is high; drive data at falling-edge of serial clock; latch data at rising-edge of serial clock.

eDRVSPI_TYPE6: the clock idle state is high; drive data at rising-edge of serial clock; latch data at falling-edge of serial clock.

eDRVSPI_TYPE7: the clock idle state is high; drive data at falling-edge of serial clock; latch data at falling-edge of serial clock. Drive data and latch data at the same edge. Not recommend to use this transfer type.

i32BitLength [in]

Bit length per transaction. The range is 1 ~32.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS	: Success.
E_DRVSPI_ERR_INIT	: The specified SPI port has been opened before.
E_DRVSPI_ERR_BIT_LENGTH	: The bit length is out of range.
E_DRVSPI_ERR_BUSY	: The specified SPI port is in busy status.

Example

```
/* Configure SPI0 as a master, 32-bit transaction */
DrvSPI_Open(eDRVSPI_PORT0, eDRVSPI_MASTER, eDRVSPI_TYPE1, 32);
```

DrvSPI_Close

Prototype

```
void DrvSPI_Close (
   E_DRVSPI_PORT eSpiPort
);
```

Description

Close the specified SPI module and disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Close SPI0 */

DrvSPI_Close(eDRVSPI_PORT0);

DrvSPI_SetEndian

Prototype

void DrvSPI_SetEndian (

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_ENDIAN eEndian

);

Description

This function is used to configure the bit order of each transaction.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eEndian [in]

Specify LSB first (eDRVSPI_LSB_FIRST) or MSB first (eDRVSPI_MSB_FIRST.)

Include

Driver/DrvSPI.h

Return Value

None

Example

/* The transfer order of SPI0 is LSB first */

DrvSPI_SetEndian(eDRVSPI_PORT0, eDRVSPI_LSB_FIRST);

DrvSPI_SetBitLength

Prototype

int32_t DrvSPI_SetBitLength(

E_DRVSPI_PORT eSpiPort,

int32_t i32BitLength

);

Description

This function is used to configure the bit length of SPI transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

i32BitLength [in]

Specify the bit length. The range is 1~32 bits.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BIT_LENGTH : The bit length is out of range.

Example

```
/* The transfer bit length of SPI0 is 8-bit */
DrvSPI_SetBitLength(eDRVSPI_PORT0, 8);
```

DrvSPI_SetByteReorder

Prototype

```
int32_t DrvSPI_SetByteReorder (
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_BYTE_REORDER eOption
);
```

Description

This function is used to enable/disable Byte Reorder function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eOption [in]

The options of Byte Reorder function and Byte Suspend function. The Byte Suspend function is only available in 32-bit transaction. The Byte Reorder function is only available in 16-/24-/32-bit transaction.

eDRVSPI_BYTE_REORDER_SUSPEND_DISABLE:

Both Byte Reorder function and Byte Suspend function are disabled.

eDRVSPI_BYTE_REORDER_SUSPEND:

Both Byte Reorder function and Byte Suspend function are enabled.

eDRVSPI_BYTE_REORDER:

Only enable the Byte Reorder function.

eDRVSPI_BYTE_SUSPEND:

Only enable the Byte Suspend function.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BIT_LENGTH : The bit length MUST be 16/24/32.

Example

```
/* The transfer bit length of SPI0 is 32-bit */
DrvSPI_SetBitLength(eDRVSPI_PORT0, 32);
/* Enable the Byte Reorder function of SPI0 */
DrvSPI_SetByteReorder(eDRVSPI_PORT0, eDRVSPI_BYTE_REORDER);
```

DrvSPI_SetSuspendCycle

Prototype

```
int32_t DrvSPI_SetSuspendCycle (
    E_DRVSPI_PORT eSpiPort,
    int32_t i32Interval
);
```

Description

Set the length of the suspend interval. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

i32Interval [in]

In burst transfer mode, this value specified the delay clock number between successive transactions. It could be 2~17. If the Byte Suspend function is enabled, it specified the delay clock number among each byte. It could be 2~17.

Ex:

i32Interval=2 … the suspend interval is 2 SPI clocks

i32Interval=17 … the suspend interval is 17 SPI clocks

In FIFO mode, this value specified the delay clock number between successive transactions. It could be 2~15 and 0. If i32Interval=0, the suspend interval is

35 system clocks + 1 SPI clock

If i32Interval=2~15, the expression of suspend interval is

(i32Interval+3) * system clock period + 1 SPI clock period

Ex:

i32Interval=2 … the suspend interval is 5 system clocks + 1 SPI clock

i32Interval=15 … the suspend interval is 18 system clocks + 1 SPI clock

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_SUSPEND_INTERVAL : The suspend interval setting is out of range.

Example

```
/* Enable the Byte Suspend function of SPI0 */
DrvSPI_SetByteReorder(eDRVSPI_PORT0, eDRVSPI_BYTE_SUSPEND)
/* The suspend interval is 10 SPI clock cycles */
DrvSPI_SetSuspendCycle (eDRVSPI_PORT0, 10);
```

DrvSPI_SetTriggerMode

Prototype

```
void DrvSPI_SetTriggerMode (
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_SSLTRIG eSSTriggerMode
);
```

Description

Set the trigger mode of slave select pin. In master mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eSSTriggerMode [in]

Specify the trigger mode.

eDRVSPI_EDGE_TRIGGER: edge trigger.

eDRVSPI_LEVEL_TRIGGER: level trigger.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Level trigger */
DrvSPI_SetTriggerMode(eDRVSPI_PORT0, eDRVSPI_LEVEL_TRIGGER);
```

DrvSPI_SetSlaveSelectActiveLevel

Prototype

```
void DrvSPI_SetSlaveSelectActiveLevel (
   E_DRVSPI_PORT eSpiPort,
   E_DRVSPI_SS_ACT_TYPE eSSActType
);
```

Description

Set the active level of slave select.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eSSActType [in]

Select the active type of slave select pin.

eDRVSPI_ACTIVE_LOW_FALLING:

Slave select pin is active low in level-trigger mode; or falling-edge trigger in edge-trigger mode.

eDRVSPI_ACTIVE_HIGH_RISING:

Slave select pin is active high in level-trigger mode; or rising-edge trigger in edge-trigger mode.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Configure the active level of SPI0 slave select pin */
DrvSPI_SetSlaveSelectActiveLevel(eDRVSPI_PORT0,
eDRVSPI_ACTIVE_LOW_FALLING);
```

DrvSPI_GetLevelTriggerStatus

Prototype

```
uint8_t DrvSPI_GetLevelTriggerStatus (
    E_DRVSPI_PORT eSpiPort
);
```

Description

This function is used to get the level-trigger transmission status of slave device.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TRUE: The transaction number and the transferred bit length met the specified requirements.

FALSE: The transaction number or the transferred bit length of one transaction doesn't meet the specified requirements.

Example

```
/* Level trigger */
```

```
DrvSPI_SetTriggerMode(eDRVSPI_PORT0, eDRVSPI_LEVEL_TRIGGER);
…
/* Check the level-trigger transmission status */
If( DrvSPI_GetLevelTriggerStatus(eDRVSPI_PORT0) )
        DrvSPI_DumpRxRegister(eDRVSPI_PORT0,
        &au32DestinationData[u32DataCount], 1); /* Read Rx buffer */
```

DrvSPI_EnableAutoSS

Prototype

```
void DrvSPI_EnableAutoSS (
   E_DRVSPI_PORT eSpiPort,
   E_DRVSPI_SLAVE_SEL eSlaveSel
);
```

Description

This function is used to enable the automatic slave select function and select the slave select pins. The automatic slave select means the SPI will set the slave select pin to active state when transferring data and set the slave select pin to inactive state when one transfer is finished. For some devices, the slave select pin may need to be kept at active state for many transfers. User should disable the automatic slave select function and control the slave select pin manually for these devices. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eSlaveSel [in]

Select the slave select pins which will be used.

eDRVSPI_NONE : no slave was selected.

eDRVSPI_SS0 : the SS0 was selected.

eDRVSPI_SS1 : the SS1 was selected.

eDRVSPI_SS0_SS1 : both SS0 and SS1 were selected.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable the automatic slave select function of SS0. */
DrvSPI_EnableAutoSS(eDRVSPI_PORT0, eDRVSPI_SS0);
```

DrvSPI_DisableAutoSS

Prototype

```
void DrvSPI_DisableAutoSS (
    E_DRVSPI_PORT eSpiPort
);
```

Description

This function is used to disable the automatic slave selection function. If user wants to keep the slave select signal at active state during multiple words data transfer, user can disable the automatic slave selection function and control the slave select signal manually. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Disable the automatic slave select function of SPI0 */
DrvSPI_DisableAutoSS(eDRVSPI_PORT0);
```

DrvSPI_SetSS

Prototype

```
void DrvSPI_SetSS(
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_SLAVE_SEL eSlaveSel
);
```

Description

Set the slave select pins. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eSlaveSel [in]

In automatic slave select operation, use this parameter to select the slave select pins which will be used.

In manual slave select operation, the specified slave select pins will be set to active state. It could be eDRVSPI_NONE, eDRVSPI_SS0, eDRVSPI_SS1 or eDRVSPI_SS0_SS1.

eDRVSPI_NONE : no slave was selected.

eDRVSPI_SS0 : the SS0 was selected.

eDRVSPI_SS1 : the SS1 was selected.

eDRVSPI_SS0_SS1 : both SS0 and SS1 were selected.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Disable the automatic slave select function of SPI0 */
DrvSPI_DisableAutoSS(eDRVSPI_PORT0);
/* Set the SS0 pin to active state */
DrvSPI_SetSS(eDRVSPI_PORT0, eDRVSPI_SS0);
```

DrvSPI_ClrSS

Prototype

```
void DrvSPI_ClrSS(
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_SLAVE_SEL eSlaveSel
);
```

Description

Set the specified slave select pins to inactive state. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eSlaveSel [in]

Specify slave select pins.

eDRVSPI_NONE : no slave was selected.

eDRVSPI_SS0 : the SS0 was selected.

eDRVSPI_SS1 : the SS1 was selected.

eDRVSPI_SS0_SS1 : both SS0 and SS1 were selected.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Disable the automatic slave select function of SPI0 */
DrvSPI_DisableAutoSS(eDRVSPI_PORT0);
/* Set the SS0 pin to inactive state */
DrvSPI_ClrSS(eDRVSPI_PORT0, eDRVSPI_SS0);
```

DrvSPI_IsBusy

Prototype

```
uint8_t DrvSPI_IsBusy(
    E_DRVSPI_PORT eSpiPort
);
```

Description

Check the busy status of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TURE: The SPI port is in busy.

FALSE: The SPI port is not in busy.

Example

```
/* set the GO_BUSY bit of SPI0 */
DrvSPI_SetGo(eDRVSPI_PORT0);
/* Check the busy status of SPI0 */
while( DrvSPI_IsBusy(eDRVSPI_PORT0) );
```

DrvSPI_BurstTransfer

Prototype

```
int32_t DrvSPI_BurstTransfer(
    E_DRVSPI_PORT eSpiPort,
    int32_t i32BurstCnt,
    int32_t i32Interval
);
```

Description

Configure the burst transfer settings. If i32BurstCnt is set to 2, it performs burst transfer. SPI controller will transfer two successive transactions. The suspend interval length between the two transactions is determined by the value of i32Interval. In slave mode, the setting of i32Interval is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

i32BurstCnt [in]

Specify the transaction number in one transfer. It could be 1 or 2.

i32Interval [in]

Suspend interval length. Specify the number of SPI clock cycle between successive transactions. The range of this setting value is 2~17.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BURST_CNT : The burst count is out of range.

E_DRVSPI_ERR_SUSPEND_INTERVAL : The interval is out of range.

Example

/* Configure the SPI0 burst transfer mode; two transactions in one transfer; 10 delay clocks between the transactions. */

DrvSPI_BurstTransfer(eDRVSPI_PORT0, 2, 10);

DrvSPI_SetClockFreq

Prototype

uint32_t

DrvSPI_SetClockFreq(

E_DRVSPI_PORT eSpiPort,

uint32_t u32Clock1,

uint32_t u32Clock2

);

Description

Configure the frequency of SPI clock. In master mode, the output frequency of serial clock is programmable. If the variable clock function is enabled, the output pattern of serial clock is defined in **VARCLK**. If the bit pattern of **VARCLK** is '0', the output frequency of SPICLK is equal to the frequency of variable clock 1. Otherwise, the output frequency is equal to the frequency of variable clock 2. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

u32Clock1 [in]

Specify the SPI clock rate in Hz. It's the clock rate of SPI engine clock and variable clock 1.

u32Clock2 [in]

Specify the SPI clock rate in Hz. It's the clock rate of variable clock 2.

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

The actual clock rate of SPI engine clock is returned. The actual clock may different to the target SPI clock due to hardware limitation.

Example

```
/* SPI0 clock rate of clock 1 is 2MHz; the clock rate of clock 2 is 1MHz */
DrvSPI_SetClockFreq(eDRVSPI_PORT0, 2000000, 1000000);
```

DrvSPI_GetClock1Freq

Prototype

```
uint32_t
DrvSPI_GetClock1Freq(
    E_DRVSPI_PORT eSpiPort
);
```

Description

Get the SPI engine clock rate in Hz. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

The frequency of SPI bus engine clock. The unit is Hz.

Example

```
/* Get the engine clock rate of SPI0 */
printf("SPI clock rate: %d Hz\n", DrvSPI_GetClock1Freq(eDRVSPI_PORT0));
```

DrvSPI_GetClock2Freq

Prototype

```
uint32_t
DrvSPI_GetClock2Freq(
    E_DRVSPI_PORT eSpiPort
);
```

Description

Get the clock rate of variable clock 2 in Hz. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

The frequency of variable clock 2. The unit is Hz.

Example

```
/* Get the clock rate of SPI0 variable clock 2 */
printf("SPI clock rate of variable clock 2: %d Hz\n",
DrvSPI_GetClock2Freq(eDRVSPI_PORT0));
```

DrvSPI_SetVariableClockFunction

Prototype

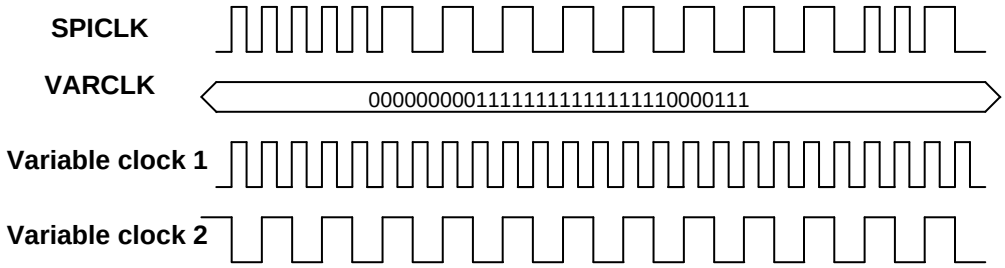
```
void
DrvSPI_SetVariableClockFunction (
   E_DRVSPI_PORT eSpiPort,
   uint8_t bEnable,
   uint32_t u32Pattern
);
```

Description

Set the variable clock function. The output pattern of serial clock is defined in **VARCLK** register. A two-bit combination in the **VARCLK** defines the pattern of one serial clock cycle. The bit field **VARCLK**[31:30] defines the first clock cycle of SPICLK. The bit field **VARCLK**[29:28] defines the second clock cycle of SPICLK and so on. The following figure is the timing relationship among the serial clock (SPICLK), the **VARCLK** register and the variable clock sources.

If the bit pattern of **VARCLK** is '0', the output frequency of SPICLK is equal to the frequency of variable clock 1.

If the bit pattern of **VARCLK** is '1', the output frequency of SPICLK is equal to the frequency of variable clock 2.



Note that when enable the variable clock function, the setting of transfer bit length must be programmed as 0x10 (16 bits mode) only.

In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

bEnable [in]

Enable (TRUE) / Disable (FALSE)

u32Pattern [in]

Specify the variable clock pattern. If **bEnable** is set to 0, this setting is functionless.

Include

Driver/DrvSPI.h

Return Value

None.

Example

```
/* Enable the SPI0 variable clock function and set the variable clock pattern */
DrvSPI_SetVariableClockFunction(eDRVSPI_PORT0, TRUE, 0x007FFF87);
```

DrvSPI_EnableInt

Prototype

```
void DrvSPI_EnableInt(
    E_DRVSPI_PORT eSpiPort,
    PFN_DRVSPI_CALLBACK pfnCallback,
    uint32_t u32UserData
);
```

Description

Enable the SPI interrupt of the specified SPI port and install the callback function.

Parameters

u16Port [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pfnCallback [in]

The callback function of the corresponding SPI interrupt.

u32UserData [in]

The parameter which will be passed to the callback function.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable the SPI0 interrupt and install the callback function. The parameter 0 will be passed
to the callback function. */
DrvSPI_EnableInt(eDRVSPI_PORT0, SPI0_Callback, 0);
```

DrvSPI_DisableInt

Prototype

```
void DrvSPI_DisableInt(
    E_DRVSPI_PORT eSpiPort
);
```

Description

Disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Disable the SPI0 interrupt */
DrvSPI_DisableInt(eDRVSPI_PORT0);
```

DrvSPI_GetIntFlag

Prototype

```
uint32_t DrvSPI_GetIntFlag (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Get the SPI interrupt flag.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

0: the SPI interrupt does not occur.

1: the SPI interrupt occurs.

Example

```
/* Get the SPI0 interrupt flag */
DrvSPI_GetIntFlag(eDRVSPI_PORT0);
```

DrvSPI_ClrIntFlag

Prototype

```
void DrvSPI_ClrIntFlag (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Clear the SPI interrupt flag.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None.

Example

```
/* Clear the SPI0 interrupt flag */
DrvSPI_ClrIntFlag(eDRVSPI_PORT0);
```

DrvSPI_SingleRead

Prototype

```
uint8_t DrvSPI_SingleRead(
    E_DRVSPI_PORT eSpiPort,
    uint32_t *pu32Data
);
```

Description

Read data from SPI RX registers and trigger SPI for next transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Data [out]

A buffer pointer. This buffer is used for storing the data got from the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data is valid.

FALSE: The data stored in pu32Data is invalid.

Example

/* Read the previous retrieved data and trigger next transfer. */

uint32_t u32DestinationData;

DrvSPI_SingleRead(eDRVSPI_PORT0, &u32DestinationData);

DrvSPI_SingleWrite

Prototype

uint8_t DrvSPI_SingleWrite (
E_DRVSPI_PORT eSpiPort,
uint32_t *pu32Data
);

Description

Write data to SPI TX0 register and trigger SPI to start transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Data [in]

A buffer pointer. The data stored in this buffer will be transmitted through the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data has been transferred.

FALSE: The SPI is in busy. The data stored in pu32Data has not been transferred.

Example

/* Write the data stored in u32SourceData to TX buffer of SPI0 and trigger SPI to start transfer. */

uint32_t u32SourceData;

DrvSPI_SingleWrite(eDRVSPI_PORT0, &u32SourceData);

DrvSPI_BurstRead

Prototype

uint8_t DrvSPI_BurstRead (
E_DRVSPI_PORT eSpiPort,

```
    uint32_t *pu32Buf
);
```

Description

Read two words of data from SPI RX registers and then trigger SPI for next transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [out]

A buffer pointer. This buffer is used for storing the data got from the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf is valid.

FALSE: The data stored in pu32Buf is invalid.

Example

```
/* Read two words of data from SPI0 RX registers to au32DestinationData[u32DataCount]
and au32DestinationData[u32DataCount+1]. And then trigger SPI for next transfer. */

DrvSPI_BurstRead(eDRVSPI_PORT0, &au32DestinationData[u32DataCount]);
```

DrvSPI_BurstWrite

Prototype

```
uint8_t DrvSPI_BurstWrite (
    E_DRVSPI_PORT eSpiPort,
    uint32_t *pu32Buf
);
```

Description

Write two words of data to SPI TX registers and then trigger SPI to start a transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [in]

A buffer pointer. The data stored in this buffer will be transmitted through the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf has been transferred.

FALSE: The SPI is in busy. The data stored in pu32Buf has not been transferred.

Example

```
/* Write two words of data stored in au32SourceData[u32DataCount] and
au32SourceData[u32DataCount+1] to SPI0 TX registers. And then trigger SPI for next
transfer. */

DrvSPI_BurstWrite(eDRVSPI_PORT0, &au32SourceData[u32DataCount]);
```

DrvSPI_DumpRxRegister

Prototype

```
uint32_t
DrvSPI_DumpRxRegister (
    E_DRVSPI_PORT eSpiPort,
    uint32_t *pu32Buf,
    uint32_t u32DataCount
);
```

Description

Read data from RX registers. This function will not trigger a SPI data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [out]

A buffer pointer. This buffer is used for storing the data got from the SPI RX registers.

u32DataCount [in]

The count of data read from RX registers. The maximum number is 2.

Include

Driver/DrvSPI.h

Return Value

The count of data actually read from Rx registers.

Example

```
/* Read one word of data from SPI0 RX buffer and store to
au32DestinationData[u32DataCount] */

DrvSPI_DumpRxRegister(eDRVSPI_PORT0, &au32DestinationData[u32DataCount], 1);
```

DrvSPI_SetTxRegister

Prototype

```
uint32_t
DrvSPI_SetTxRegister (
    E_DRVSPI_PORT eSpiPort,
    uint32_t *pu32Buf,
    uint32_t u32DataCount
);
```

Description

Write data to TX registers. This function will not trigger a SPI data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [in]

A buffer stores the data which will be written to TX registers.

u32DataCount [in]

The count of data written to TX registers.

Include

Driver/DrvSPI.h

Return Value

The count of data actually written to SPI TX registers.

Example

```
/* Write one word of data stored in u32Buffer to SPI0 TX register. */

DrvSPI_SetTxRegister(eDRVSPI_PORT0, &u32Buffer, 1);
```

DrvSPI_SetGo

Prototype

```
void DrvSPI_SetGo (
    E_DRVSPI_PORT eSpiPort
);
```

Description

In master mode, call this function can start a SPI data transfer. In slave mode, executing this function means that the slave is ready to communicate with a master.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Trigger a SPI data transfer */
DrvSPI_SetGo(eDRVSPI_PORT0);
```

DrvSPI_ClrGo

Prototype

```
void DrvSPI_ClrGo (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Stop a SPI data trasfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Stop a SPI data transfer */
DrvSPI_ClrGo(eDRVSPI_PORT0);
```

DrvSPI_SetFIFOMode

Prototype

void

DrvSPI_SetFIFOMode (

E_DRVSPI_PORT eSpiPort,

uint8_t bEnable,

int32_t i32Interval

);

Description

Enable/disable FIFO mode. If the caller enables FIFO mode, check the setting of suspend interval.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

bEnable [in]

Enable (TRUE) / Disable (FALSE)

i32Interval [in]

In FIFO mode, it could be 2~15 and 0. 0 indicates the maximum suspend interval; 2 indicates the minimum suspend interval. Please refer to NUC122 TRM for the actual suspend interval.

Include

Driver/DrvSPI.h

Return Value

None.

Example

```
/* Enable the SPI0 FIFO mode */
DrvSPI_SetFIFOMode(eDRVSPI_PORT0, TRUE, 0);
```

DrvSPI_IsRxEmpty

Prototype

uint8_t DrvSPI_IsRxEmpty(

E_DRVSPI_PORT eSpiPort

);

Description

Check the status of the Rx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TURE: Rx buffer is empty.

FALSE: Rx buffer is not empty.

Example

```
/* Enable the SPI0 FIFO mode */
DrvSPI_SetFIFOMode(eDRVSPI_PORT0, TRUE, 0);
/* Check the status of SPI0 Rx buffer */
while( DrvSPI_IsRxEmpty(eDRVSPI_PORT0) )
{
…
}
```

DrvSPI_IsRxFull

Prototype

```
uint8_t DrvSPI_IsRxFull(
   E_DRVSPI_PORT eSpiPort
);
```

Description

Check the status of the Rx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TURE: Rx buffer is full.

FALSE: Rx buffer is not full.

Example

```
/* Enable the SPI0 FIFO mode */
DrvSPI_SetFIFOMode(eDRVSPI_PORT0, TRUE, 0);
/* Check the status of SPI0 Rx buffer */
while( DrvSPI_IsRxFull(eDRVSPI_PORT0) )
{
…
}
```

DrvSPI_IsTxEmpty

Prototype

```
uint8_t DrvSPI_IsTxEmpty(
   E_DRVSPI_PORT eSpiPort
);
```

Description

Check the status of the Tx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TURE: Tx buffer is empty.

FALSE: Tx buffer is not empty.

Example

```
/* Enable the SPI0 FIFO mode */
```

```
DrvSPI_SetFIFOMode(eDRVSPI_PORT0, TRUE, 0);
/* Check the status of SPI0 Tx buffer */
while( DrvSPI_IsTxEmpty(eDRVSPI_PORT0) )
{
…
}
```

DrvSPI_IsTxFull

Prototype

```
uint8_t DrvSPI_IsTxFull(
   E_DRVSPI_PORT eSpiPort
);
```

Description

Check the status of the Tx buffer of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TURE: Tx buffer is full.

FALSE: Tx buffer is not full.

Example

```
/* Enable the SPI0 FIFO mode */
DrvSPI_SetFIFOMode(eDRVSPI_PORT0, TRUE, 0);
/* Check the status of SPI0 Tx buffer */
while( DrvSPI_IsTxFull(eDRVSPI_PORT0) )
{
…
}
```

DrvSPI_GetVersion

Prototype

uint32_t

DrvSPI_GetVersion (void);

Description

Get the version number of SPI driver.

Include

Driver/DrvSPI.h

Return Value

Version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

printf("Driver version: %x\n", DrvSPI_GetVersion());

7. I2C Driver

7.1. I2C Introduction

I2C is bi-directional serial bus with two wires that provides a simple and efficient method of data exchange between devices. The I2C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously. Serial, 8-bit oriented bi-directional data transfers can be made up 1.0 Mbps.

For NuMicro™ NUC122 Series, I2C device could act as master or slave and I2C driver can help user to use I2C functions easily.

7.2. I2C Feature

The I2C includes following features:

- Support master and slave mode up to 1Mbps.
- Built-in a 14-bit time-out counter will request the I2C interrupt if the I2C bus hangs up and time-out counter overflows.
- Support 7-bit addressing mode.
- Support multiple address recognition. (four slave address with mask option)

7.3. Type Definition

E_I2C_CALLBACK_TYPE

Enumeration identifier	Value	Description
I2CFUNC	0	For I2C Normal condition
ARBITLOSS	1	For Arbitration Loss condition when I2C operates as master mode.
BUSERROR	2	For I2C Bus Error condition
TIMEOUT	3	For I2C 14-bit time-out counter time out

7.4. Functions

DrvI2C_Open

Prototype

int32_t DrvI2C_Open (uint32_t u32BusClock);

Description

To open the I2C hardware and configure the I2C bus clock. The maximum of I2C bus clock is 1 MHz.

Parameter

u32BusClock [in]

To configure I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
/* Enable I2C and set I2C bus clock 100KHz */
DrvI2C_Open (100000);
```

DrvI2C_Close

Prototype

int32_t DrvI2C_Close (void);

Description

To close the I2C hardware.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
DrvI2C_Close ( );              /* Disable I2C */
```

DrvI2C_SetClockFreq

Prototype

int32_t DrvI2C_SetClockFreq (uint32_t u32BusClock);

Description

To configure the I2C bus clock. *I2C bus clock = I2C source clock / (4 x (I2CCLK_DIV+1)).* The maximum of I2C bus clock is 1 MHz.

Parameter

u32BusClock [in]

To configure I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
/* Set I2C bus clock 200 KHz */
DrvI2C_SetClockFreq (200000);
```

DrvI2C_GetClockFreq

Prototype

uint32_t DrvI2C_GetClockFreq (void);

Description

To get the I2C bus clock. *I2C bus clock = I2C source clock / (4 x (I2CCLK_DIV+1))*

Parameter

None

Include

Driver/DrvI2C.h

Return Value

I2C bus clock

Example

```
uint32_t    u32clock;
```

```
u32clock = DrvI2C_GetClockFreq ( );      /* Get I2C bus clock */
```

DrvI2C_SetAddress

Prototype

int32_t DrvI2C_SetAddress (uint8_t slaveNo, uint8_t slave_addr, uint8_t GC_Flag);

Description

To set 7-bit physical slave address to the specified I2C slave address. Four slave addresses supported. The setting takes effect when I2C operates as slave mode.

Parameter

slaveNo [in]

To select slave address. The slaveNo is 0 ~ 3.

slave_addr [in]

To set 7-bit physical slave address for selected slave address.

GC_Flag [in]

To enable or disable general call function. (1: enable, 0: disable)

Include

Driver/DrvI2C.h

Return Value

0	Succeed
<0	Failed

Example

```
DrvI2C_SetAddress(0, 0x15, 0);    /* Set I2C 1st slave address 0x15 */
DrvI2C_SetAddress(1, 0x35, 0);    /* Set I2C 2nd slave address 0x35 */
DrvI2C_SetAddress(2, 0x55, 0);    /* Set I2C 3rd slave address 0x55 */
DrvI2C_SetAddress(3, 0x75, 0);    /* Set I2C 4th slave address 0x75 */
```

DrvI2C_SetAddressMask

Prototype

int32_t DrvI2C_SetAddressMask (uint8_t slaveNo, uint8_t slaveAddrMask);

Description

To set 7-bit physical slave address mask to the specified I2C salve address mask. Four slave address masks supported. The setting takes effect when I2C operates as slave mode.

Parameter

slaveNo [in]

To select slave address mask. The value is 0 ~ 3.

slaveAddrMask [in]

To set 7-bit physical slave address mask for selected slave address mask. The corresponding address bit is “Don’t care”.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

<0 Failed

Example

```
DrvI2C_SetAddress (0, 0x15, 0);      /* Set I2C 1st slave address 0x15 */
DrvI2C_SetAddress (1, 0x35, 0);      /* Set I2C 2nd slave address 0x35 */
/* Set I2C 1st slave address mask 0x01, slave address 0x15 and 0x14 would be addressed */
DrvI2C_SetAddressMask (0, 0x01);
/* Set I2C 2nd slave address mask 0x04, slave address 0x35 and 0x31 would be addressed */
DrvI2C_SetAddressMask (1, 0x04);
```

DrvI2C_GetStatus

Prototype

uint32_t DrvI2C_GetStatus (void);

Description

To get the I2C status code. There are 26 status codes. Please refer to Data Transfer Flow in I2C Section of TRM in details.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

I2C status code

Example

```
uint32_t    u32status;
u32status = DrvI2C_GetStatus ( );      /* Get I2C current status code */
```

DrvI2C_WriteData

Prototype

void DrvI2C_WriteData (uint8_t u8data);

Description

To set a byte of data to be sent.

Parameter

u8data [in]

Byte data.

Include

Driver/DrvI2C.h

Return Value

None

Example

DrvI2C_WriteData (0x55); /* Set byte data 0x55 into I2C data register */

DrvI2C_ReadData

Prototype

uint8_t DrvI2C_ReadData (void);

Description

To read the last data from I2C bus.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Last byte data

Example

```
uint8_t  u8data;
u8data = DrvI2C_ReadData ( ); /* Read out byte data from I2C data register */
```

DrvI2C_Ctrl

Prototype

void DrvI2C_Ctrl (uint8_t start, uint8_t stop, uint8_t intFlag, uint8_t ack);

Description

To set I2C control bit include STA, STO, AA, SI in control register.

Parameter

start [in]

To set STA bit or not. (1: set, 0: don't set)

If the STA bit is set, a START or repeat START signal will be generated when I2C bus is free.

stop [in]

To set STO bit or not. (1: set, 0: don't set)

If the STO bit is set, a STOP signal will be generated. When a STOP condition is detected, this bit will be cleared by hardward automatically.

intFlag [in]

To clear SI flag (I2C interrupt flag). (1: clear, 0: don't work)

ack [in]

To enable AA bit (Assert Acknowledge control bit) or not. (1: enable, 0: disable)

Include

Driver/DrvI2C.h

Return Value

None

Example

DrvI2C_Ctrl (0, 0, 1, 0); /* Set I2C SI bit to clear SI flag */

DrvI2C_Ctrl (1, 0, 0, 0); /* Set I2C STA bit to send START signal */

DrvI2C_GetIntFlag

Prototype

uint8_t DrvI2C_GetIntFlag (void);

Description

To get I2C interrupt flag status.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Interrupt status (1 or 0)

Example

```
uint8_t    u8flagStatus;
u8flagStatus = DrvI2C_GetIntFlag ( ); /* Get the status of I2C interrupt flag */
```

DrvI2C_ClearIntFlag

Prototype

```
void      DrvI2C_ClearIntFlag (void);
```

Description

To clear I2C interrupt flag if the flag is set 1.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

Example

```
DrvI2C_ClearIntFlag ( ); /* Clear I2C interrupt flag (SI) */
```

DrvI2C_EnableInt

Prototype

```
int32_t    DrvI2C_EnableInt (void);
```

Description

To enable I2C interrupt function.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

DrvI2C_EnableInt ( ); /* Enable I2C interrupt */

DrvI2C_DisableInt

Prototype

int32_t DrvI2C_DisableInt (void);

Description

To disable I2C interrupt function.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

DrvI2C_DisableInt ( ); /* Disable I2C interrupt */

DrvI2C_InstallCallBack

Prototype

int32_t DrvI2C_InstallCallBack (E_I2C_CALLBACK_TYPE Type, I2C_CALLBACK callbackfn);

Description

To install I2C call back function in I2C interrupt handler.

Parameter

Type [in]

There are four types for call back function. (I2CFUNC / ARBITLOSS / BUSERROR / TIMEOUT)

I2CFUNC: For normal I2C condition

ARBITLOSS: For master mode when arbitration loss occurs. The status code is 0x38.

BUSERROR: For bus error condition. The status code is 0x00.

TIMEOUT: For 14-bit time-out counter overflow.

callbackfn [in]

Call back function name for specified interrupt event.

Include

Driver/DrvI2C.h

Return Value

0	Succeed
<0	Failed

Example

```
/* Install I2C call back function 'I2C_Callback_Normal' for I2C normal condition */
DrvI2C_InstallCallback (I2CFUNC, I2C_Callback_Normal);
/* Install I2C call back function 'I2C_Callback_BusErr' for Bus Error condition */
DrvI2C_InstallCallback (BUSERROR, I2C_Callback_BusErr);
```

DrvI2C_UninstallCallBack

Prototype

int32_t DrvI2C_UninstallCallBack (E_I2C_CALLBACK_TYPE Type);

Description

To uninstall I2C call back function in I2C interrupt handler.

Parameter

Type [in]

There are four types for call back function. (I2CFUNC / ARBITLOSS / BUSERROR / TIMEOUT)

I2CFUNC: For normal I2C condition

ARBITLOSS: For master mode when arbitration loss occurs. The status code is 0x38.

BUSERROR: For bus error condition. The status code is 0x00.

TIMEOUT: For 14-bit time-out counter overflow.

Include

Driver/DrvI2C.h

Return Value

0	Succeed
<0	Failed

Example

```
/* Uninstall I2C call back function for I2C normal condition */
DrvI2C_UninstallCallBack (I2CFUNC);
/* Uninstall I2C call back function for Bus Error condition */
DrvI2C_UninstallCallBack (BUSERROR);
```

DrvI2C_SetTimeoutCounter

Prototype

int32_t DrvI2C_SetTimeoutCounter (int32_t i32enable, uint8_t u8div4);

Description

To configure 14-bit time-out counter.

Parameter

i32enable [in]

To enable or disable 14-bit time-out counter. (1: enable, 0: disable)

u8div4 [in]

1: Enable DIV4 function. The source clock of the time-out counter is equal to HCLK / 4 when the time-out counter is enabled.

0: Disable DIV4 function. The source clock of the time-out counter is from HCLK when the time-out counter is enabled.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
/* Enable I2C 14-bit timeout counter and disable its DIV4 function */
DrvI2C_EnableTimeoutCount (1, 0);
```

DrvI2C_ClearTimeoutFlag

Prototype

void DrvI2C_ClearTimeoutFlag (void);

Description

To clear I2C TIF flag if the flag is set 1.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

Example

```
DrvI2C_ClearTimeoutFlag ( ); /* Clear I2C TIF flag */
```

DrvI2C_GetVersion

Prototype

uint32_t DrvI2C_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

8. RTC Driver

8.1. RTC Introduction

Real Time Clock (RTC) unit provides user the real time and calendar message .The RTC uses a 32.768 KHz external crystal. A built in RTC is designed to generate the periodic interrupt signal. The period can be 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second. There is RTC overflow counter and it can be adjusted by software.

8.2. RTC Features

- There is a time counter for user to check time.
- Support periodic time tick interrupt with 8 period options 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second.
- Support RTC Time Tick and Alarm Match interrupt
- Support wake up CPU from sleep or power-down mode.

8.3. Constant Definition

Constant Name	Value	Description
DRVRTC_INIT_KEY	0xa5eb1357	A key number to make RTC leaving reset state
DRVRTC_WRITE_KEY	0xA965	A key number to unlock RTC protected regiser
DRVRTC_CLOCK_12	0	12-Hour mode
DRVRTC_CLOCK_24	1	24-Hour mode
DRVRTC_AM	1	a.m.
DRVRTC_PM	2	p.m.
DRVRTC_YEAR2000	2000	Set the year is 2000.
DRVRTC_FCR_REFERENCE	32761	A reference value to compensate 32 kHz

8.4. Type Definition

E_DRVRTC_INT_SOURCE

Enumeration identifier	Value	Description
DRVRTC_ALARM_INT	1	Set alarm interrupt
DRVRTC_TICK_INT	2	Set tick interrupt
DRVRTC_ALL_INT	3	Set alarm and tick interrupt

E_DRVRTC_TICK

Enumeration identifier	Value	Description
DRVRTC_TICK_1_SEC	0	Set tick period 1 tick per second
DRVRTC_TICK_1_2_SEC	1	Set tick period 2 tick per second
DRVRTC_TICK_1_4_SEC	2	Set tick period 4 tick per second
DRVRTC_TICK_1_8_SEC	3	Set tick period 8 tick per second
DRVRTC_TICK_1_16_SEC	4	Set tick period 16 tick per second
DRVRTC_TICK_1_32_SEC	5	Set tick period 32 tick per second
DRVRTC_TICK_1_64_SEC	6	Set tick period 64 tick per second
DRVRTC_TICK_1_128_SEC	7	Set tick period 128 tick per second

E_DRVRTC_TIME_SELECT

Enumeration identifier	Value	Description

DRVRTC_CURRENT_TIME	0	Select current time option
DRVRTC_ALARM_TIME	1	Select alarm time option

E_DRVRTC_DWR_PARAMETER

Enumeration identifier	Value	Description
DRVRTC_SUNDAY	0	Day of Week: Sunday
DRVRTC_MONDAY	1	Day of Week: Monday
DRVRTC_TUESDAY	2	Day of Week: Tuesday
DRVRTC_WEDNESDAY	3	Day of Week: Wednesday
DRVRTC_THURSDAY	4	Day of Week: Thursday
DRVRTC_FRIDAY	5	Day of Week: Friday
DRVRTC_SATURDAY	6	Day of Week: Saturday

8.5. Functions

DrvRTC_SetFrequencyCompensation

Prototype

```
int32_t
DrvRTC_SetFrequencyCompensation (
    int32_t i32FrequencyX100
    );
```

Description

Set Frequency Compensation Data

Parameter

i32FrequencyX100 [in]

Specify the RTC clock X100, ex: 3277365 means 32773.65.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FCR_VALUE: Wrong Compensation value

Example

```
/* If the measured RTC crystal frequency is 32773.65Hz.*/
DrvRTC_SetFrequencyCompensation (3277365) ;
```

DrvRTC_IsLeapYear

Prototype

```
int32_t
DrvRTC_IsLeapYear (void);
```

Description

According to current time , return this year is leap year or not.

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

1: This year is a leap year.

0: This year is not a leap year.

Example

```
If (DrvRTC_IsLeapYear())
   printf("This is Leap year!");
else
   printf("This is not Leap year!");
```

DrvRTC_GetIntTick

Prototype

```
int32_t DrvRTC_GetIntTick (void);
```

Description

The function is used to get current Software tick count after enable tick interrupt.

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

Interrupt tick count number

Example

```
/* Polling the tick count to wait 3 sec.*/
DrvRTC_SetTickMode(DRVRTC_TICK_1_2_SEC) ;    /* 1 tick is 0.5 sec.*/
DrvRTC_EnableInt(DRVRTC_TICK_INT,NULL);
while(DrvRTC_GetTick()<6);
printf("Pass though 3 sec\n")
```

DrvRTC_ResetIntTick

Prototype

void DrvRTC_ResetTick (void);

Description

The function is used to reset the tick count counting in interrupt.

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

None

Example

```
DrvRTC_ResetTick () ;
```

DrvRTC_WriteEnable

Prototype

int32_t

DrvRTC_WriteEnable (void);

Description

Access Password to AER to make access other register enable

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FAILED : Failed.

Note

After write a password to AER register, FCR / TAR / CAR / TTR register can be written or read. And after 512 RTC clocks(about 15ms), access enable wiil auto-clear.

Example

```
/* Before you want to set the value in FCR / TAR / CAR / TTR register, using the function to open access account. */
DrvRTC_WriteEnable (void) ;
```

DrvRTC_Init

Prototype

```
int32_t DrvRTC_Init (void);
```

Description

Initial RTC. It consists of clear callback function pointer, enable 32K clock and RTC clock and write initial key to let RTC start count.

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

Example

```
/*In the beginning, call the function to initial RTC */
DrvRTC_Init() ;
```

DrvRTC_SetTickMode

Prototype

int32_t DrvRTC_SetTickMode(uint8_t ucMode);

Description

The function is used to set time tick period for periodic time tick Interrupt.

Parameter

ucMode [in] the structure of DRVRTC_TICK. It is ued to set the RTC time tick period for Periodic Time Tick Interrupt request. It consists of

DRVRTC_TICK_1_SEC	: Time tick is 1 second
DRVRTC_TICK_1_2_SEC	: Time tick is 1/2 second
DRVRTC_TICK_1_4_SEC	: Time tick is 1/4 second
DRVRTC_TICK_1_8_SEC	: Time tick is 1/8 second
DRVRTC_TICK_1_16_SEC	: Time tick is 1/16 second
DRVRTC_TICK_1_32_SEC	: Time tick is 1/32 second
DRVRTC_TICK_1_64_SEC	: Time tick is 1/64 second
DRVRTC_TICK_1_128_SEC	: Time tick is 1/128 second

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Access Enable failed

E_DRVRTC_ERR_ENOTTY: Parameter is wrong

Example

```
/* Set Tick interrupt is 128 tick/sec */
DrvRTC_SetTickMode (DRVRTC_TICK_1_128_SEC) ;
```

DrvRTC_EnableInt

Prototype

```
int32_t DrvRTC_EnableInt (
    DRVRTC_INT_SOURCE str_IntSrc,
    PFN_DRVRTC_CALLBACK pfncallback);
```

Description

The function is used to enable specified interrupt and install callback function..

Parameter

str_IntSrc [in] the structure of interrupt source. It consists of

DRVRTC_ALARM_INT	: Alarm interrupt
DRVRTC_TICK_INT	: Tick interrupt
DRVRTC_ALL_INT	: Alarm and tick interrupt

pfncallback [in] Callback function pointer

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_ENOTTY: Parameter is wrong

Example

```
/* Enable tick interrupt and install callback function "RTC_TickCallBackfn".*/
DrvRTC_EnableInt(DRVRTC_TICK_INT, RTC_TickCallBackfn);
```

DrvRTC_DisableInt

Prototype

```
int32_t
DrvRTC_DisableInt (
   DRVRTC_INT_SOURCE str_IntSrc);
```

Description

The function is used to disable specified interrupt and clear callback function pointer.

Parameter

str_IntSrc [in] the structure of interrupt source. It consists of

DRVRTC_ALARM_INT	: Alarm interrupt
DRVRTC_TICK_INT	: Tick interrupt
DRVRTC_ALL_INT	: Alarm and tick interrupt

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_ENOTTY: Parameter is wrong

Example

/* Disable tick and alarm interrupt*/

DrvRTC_DisableInt(DRVRTC_ALL_INT);

DrvRTC_Open

Prototype

int32_t

DrvRTC_Open (

S_DRVRTC_TIME_DATA_T *sPt

);

Description

Set Current time (Year/Month/Day, Hour/Minute/Sec and day of week)

Parameter

sPt [in]

Specify the time property and current time. It includes

u8cClockDisplay	: DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24
u8cAmPm	: DRVRTC_AM / DRVRTC_PM
u32cSecond	: Second value
u32cMinute	: Minute value
u32cHour	: Hour value
u32cDayOfWeek	: Day of week
u32cDay	: Day value
u32cMonth	: Month value
u32Year	: Year value
u8IsEnableWakeUp	: Enable or not Wakeup function when time alarm happen

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Initial RTC Failed.

Example

/* Start RTC count from 2009.Jan.19, 13:20:00 . */
S_DRVRTC_TIME_DATA_T sInitTime;

```
sInitTime.u32Year       = 2009;
sInitTime.u32cMonth     = 1;
sInitTime.u32cDay       = 19;
sInitTime.u32cHour      = 13;
sInitTime.u32cMinute    = 20;
sInitTime.u32cSecond    = 0;
sInitTime.u32cDayOfWeek = DRVRTC_MONDAY;
sInitTime.u8cClockDisplay = DRVRTC_CLOCK_24;
if (DrvRTC_Open(&sInitTime) !=E_SUCCESS)
{
    printf("RTC Open Fail!!\n");
}
```

DrvRTC_Read

Prototype

```
int32_t
DrvRTC_Read (
    E_DRVRTC_TIME_SELECT eTime,
    S_DRVRTC_TIME_DATA_T *sPt
);
```

Description

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

Parameter

eTime [in]

Specify the current/alarm time to be read.

DRVRTC_CURRENT_TIME	: Current time
DRVRTC_ALARM_TIME	: Alarm time

***sPt [in]**

Specify the buffer to store the data read from RTC. It includes:

u8cClockDisplay	: DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24
u8cAmPm	: DRVRTC_AM / DRVRTC_PM
u32cSecond	: Second value
u32cMinute	: Minute value
u32cHour	: Hour value
u32cDayOfWeek	: Day of week
u32cDay	: Day value
u32cMonth	: Month value
u32Year	: Year value

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Initial RTC Failed.

Example

```
/* Condition: You want to get current RTC calendar and time */
S_DRVRTC_TIME_DATA_T sCurTime;

DrvRTC_Read(DRVRTC_CURRENT_TIME, &sCurTime);

printf("Current Time:%d/%02d/%02d    %02d:%02d:%02d\n",
   sCurTime.u32Year,sCurTime.u32cMonth,sCurTime.u32cDay,sCurTime.u32cHour,sCurTi
   me.u32cMinute,sCurTime.u32cSecond);
```

DrvRTC_Write

Prototype

```
int32_t
DrvRTC_Write (
   E_DRVRTC_TIME_SELECT eTime,
   S_DRVRTC_TIME_DATA_T *sPt
);
```

Description

Set current date/time or alarm date/time to RTC

Parameter

eTime [in]

Specify the current/alarm time to be written.

DRVRTC_CURRENT_TIME : Current time

DRVRTC_ALARM_TIME : Alarm time

***sPt [in]**

Specify the data to write to RTC. It includes:

u8cClockDisplay	DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24
u8cAmPm	DRVRTC_AM / DRVRTC_PM
u32cSecond	Second value
u32cMinute	Minute value
u32cHour	Hour value
u32cDayOfWeek	Day of week
u32cDay	Day value
u32cMonth	Month value
u32Year	Year value

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Initial RTC Failed.

Example

```
/* Condition: Update current the second of time to zero */
S_DRVRTC_TIME_DATA_T sCurTime;

DrvRTC_Read(DRVRTC_ALARM_TIME, &sCurTime);

sCurTime.u32cSecond = 0;

DrvRTC_Write(DRVRTC_ALARM_TIME, &sCurTime);
```

DrvRTC_Close

Prototype

```
int32_t
DrvRTC_Close (void);
```

Description

Disable NVIC channel of RTC and both tick and alarm interrupt..

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

Example

```
DrvRTC_Close();
```

DrvRTC_GetVersion

Prototype

```
int32_t
DrvRTC_GetVersion (void);
```

Description

Return the current version number of driver.

Include

Driver/DrvRTC.h

Return Value

Version number :

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

9. PWM Driver

9.1. PWM Introduction

The basic components in a PWM set is pre-scaler, clock divider, 16-bit counter, 16-bit comparator, inverter, dead-zone generator. They are all driven by engine clock source. There are four engine clock sources, included 12 MHz crystal clock, 32 KHz crystal clock, HCLK, and internal 22 MHZ clock. Clock divider provides the channel with 5 clock sources (1, 1/2, 1/4, 1/8, 1/16). Each PWM-timer receives its own clock signal from clock divider which receives clock from 8-bit pre-scaler. The 16-bit counter in each channel receive clock signal from clock selector and can be used to handle one PWM period. The 16-bit comparator compares number in counter with threshold number in register loaded previously to generate PWM duty cycle.

To prevent PWM driving output pin with unsteady waveform, 16-bit counter and 16-bit comparator are implemented with double buffering feature. User can feel free to write data to counter buffer register and comparator buffer register without generating glitch.

When 16-bit down counter reaches zero, the interrupt request is generated to inform CPU that time is up. When counter reaches zero, if counter is set as auto-reload mode, it is reloaded automatically and start to generate next cycle. User can set counter as one-shot mode instead of auto-reload mode. If counter is set as one-shot mode, counter will stop and generate one interrupt request when it reaches zero.

9.2. PWM Features

The PWM controller includes following features:

- Up to one PWM group (PWMA). Please refer to NuMicro™ NUC122 Series Products Selection Guide of Appendix to know the number of PWM group.
- Each PWM group has two PWM generators. Each PWM generator supports one 8-bit prescaler, one clock divider, two PWM-timers (down counter), one dead-zone generator and two PWM outputs.
- One-shot or Auto-reload PWM mode.
- Up to four capture input channels.
- Each capture input channel supports rising/falling latch register and capture interrupt flag.

9.3. Constant Definition

Constant Name	Value	Description
DRVPWM_TIMER0	0x00	PWM Timer 0
DRVPWM_TIMER1	0x01	PWM Timer 1
DRVPWM_TIMER2	0x02	PWM Timer 2
DRVPWM_TIMER3	0x03	PWM Timer 3
DRVPWM_CAP0	0x10	PWM Capture 0
DRVPWM_CAP1	0x11	PWM Capture 1
DRVPWM_CAP2	0x12	PWM Capture 2
DRVPWM_CAP3	0x13	PWM Capture 3
DRVPWM_CAP_ALL_INT	3	PWM Capture Rising and Falling Interrupt
DRVPWM_CAP_RISING_INT	1	PWM Capture Rising Interrupt
DRVPWM_CAP_FALLING_INT	2	PWM Capture Falling Interrupt
DRVPWM_CAP_RISING_FLAG	6	Capture rising interrupt flag
DRVPWM_CAP_FALLING_FLAG	7	Capture falling interrupt flag
DRVPWM_CLOCK_DIV_1	4	Input clock divided by 1
DRVPWM_CLOCK_DIV_2	0	Input clock divided by 2
DRVPWM_CLOCK_DIV_4	1	Input clock divided by 4
DRVPWM_CLOCK_DIV_8	2	Input clock divided by 8
DRVPWM_CLOCK_DIV_16	3	Input clock divided by 16
DRVPWM_AUTO_RELOAD_MODE	1	PWM Timer auto-reload mode
DRVPWM_ONE_SHOT_MODE	0	PWM Timer One-shot mode

9.4. Functions

DrvPWM_IsTimerEnabled

Prototype

int32_t DrvPWM_IsTimerEnabled(uint8_t u8Timer);

Description

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

Parameter

u8Timer [in]

Specify the timer.
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.

DRVPWM_TIMER3: PWM timer 3.

Include

Driver/DrvPWM.h

Return Value

1: The specified timer is enabled.

0: The specified timer is disabled.

Example

```
int32_t    i32state ;
/* Check if PWM timer 3 is enabled or not */
if(DrvPWM_IsTimerEnabled (DRVPWM_TIMER3)==1)
printf("PWM timer 3 is enabled!\n");
else if(DrvPWM_IsTimerEnabled (DRVPWM_TIMER3)==0)
 printf("PWM timer 3 is disabled!\n");
```

DrvPWM_SetTimerCounter

Prototype

```
void      DrvPWM_SetTimerCounter(uint8_t u8Timer, uint16_t u16Counter);
```

Description

This function is used to set the PWM specified timer counter.

Parameter

u8Timer [in]

Specify the timer.
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

u16Counter [in]

Specify the timer value. (0~65535). If the counter is set to 0, the timer will stop.

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Set 10000 to PWM timer 3 counter register. When the PWM timer 3 start to count down,
PWM timer 3 will count down from 10000 to 0. If PWM timer 3 is set to auto-reload mode,
the PWM timer 3 will reload 10000 to PWM timer 3 counter register after PWM timer 3
count down to 0 and PWM timer 3 will continue to count down from 10000 to 0 again. */

DrvPWM_SetTimerCounter(DRVPWM_TIMER3, 10000);
```

DrvPWM_GetTimerCounter

Prototype

uint32_t DrvPWM_GetTimerCounter(uint8_t u8Timer);

Description

This function is used to get the PWM specified timer counter value

Parameter

u8Timer [in]

Specify the timer.
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

Include

Driver/DrvPWM.h

Return Value

The specified timer counter value.

Example

```
/* Get PWM timer 0 counter value. */

uint32_t u32RetValTimer0CounterValue;

u32RetValTimer0CounterValue = DrvPWM_GetTimerCounter(DRVPWM_TIMER0);
```

DrvPWM_EnableInt

Prototype

void DrvPWM_EnableInt(uint8_t u8Timer, uint8_t u8Int, PFN_DRVPWM_CALLBACK pfncallback);

Description

This function is used to enable the PWM timer/capture interrupt and install the call back function.

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

u8Int [in]

Specify the capture interrupt type (The parameter is valid only when capture function)
DRVPWM_CAP_RISING_INT : The capture rising interrupt.
DRVPWM_CAP_FALLING_INT : The capture falling interrupt.
DRVPWM_CAP_ALL_INT : All capture interrupt.

pfncallback [in]

The pointer of the callback function for specified timer / capture.

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Enable PWM capture 0 falling edge interrupt and install DRVPWM_CapIRQHandler() as it's interrupt callback function.*/

DrvPWM_EnableInt(DRVPWM_CAP0, DRVPWM_CAP_FALLING_INT, DRVPWM_CapIRQHandler);
```

DrvPWM_DisableInt

Prototype

```
void DrvPWM_DisableInt(uint8_t u8Timer);
```

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Disable PWM capture 0 interrupts including rising and falling interrupt source and also uninstall PWM capture 0 rising and falling interrupt callback functions. */

DrvPWM_DisableInt(DRVPWM_CAP0);

/* Disable PWM timer 0 interrupt and uninstall PWM timer 0 callback function.*/

DrvPWM_DisableInt(DRVPWM_TIMER0);
```

DrvPWM_ClearInt

Prototype

```
void      DrvPWM_ClearInt(uint8_t u8Timer);
```

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Clear PWM timer 1 interrupt flag.*/
DrvPWM_ClearInt(DRVPWM_TIMER1);
/* Clear PWM capture 0 interrupt flag. */
DrvPWM_ClearInt(DRVPWM_CAP0);
```

DrvPWM_GetIntFlag

Prototype

```
int32_t     DrvPWM_GetIntFlag(uint8_t u8Timer);
```

Description

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

Parameter

u8Timer [in]

Specify the timer
- DRVPWM_TIMER0: PWM timer 0.
- DRVPWM_TIMER1: PWM timer 1.
- DRVPWM_TIMER2: PWM timer 2.
- DRVPWM_TIMER3: PWM timer 3.

or the capture.
- DRVPWM_CAP0: PWM capture 0.
- DRVPWM_CAP1: PWM capture 1.
- DRVPWM_CAP2: PWM capture 2.
- DRVPWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

1: The specified interrupt occurs.

0: The specified interrupt doesn't occur.

Example

```
/* Get PWM timer 2 interrupt flag.*/
if(DrvPWM_GetIntFlag(DRVPWM_TIMER2)==1)
printf("PWM timer 2 interrupt occurs!\n);
else if(DrvPWM_GetIntFlag(DRVPWM_TIMER2)==0)
printf("PWM timer 2 interrupt dosen't occur!\n);
```

DrvPWM_GetRisingCounter

Prototype

uint16_t DrvPWM_GetRisingCounter(uint8_t u8Capture);

Description

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

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

The value was latched from PWM capture current counter when there's a rising transition.

Example

```
/* Get PWM capture 3 rising latch register value. */

uint16_t u16RetValTimer3RisingLatchValue;

u16RetValTimer3RisingLatchValue = DrvPWM_GetRisingCounter (DRVPWM_CAP3);
```

DrvPWM_GetFallingCounter

Prototype

uint16_t DrvPWM_GetFallingCounter(uint8_t u8Capture);

Description

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

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

The value was latched from PWM capture current counter when there's a falling transition.

Example

/* Get PWM capture 3 falling latch register value.*/

uint16_t u16RetValTimer3FallingLatchValue;

u16RetValTimer3FallingLatchValue = DrvPWM_GetFallingCounter (DRVPWM_CAP3);

DrvPWM_GetCaptureIntStatus

Prototype

int32_t DrvPWM_GetCaptureIntStatus(uint8_t u8Capture, uint8_t u8IntType);

Description

Check if there's a rising / falling transition

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

u8IntType [in]

Specify the Capture Latched Indicator.

DRVPWM_CAP_RISING_FLAG : The capture rising indicator flag.

DRVPWM_CAP_FALLING_FLAG : The capture falling indicator flag.

Include

Driver/DrvPWM.h

Return Value

TRUE: The specified transition occurs.

FALSE: The specified transition doesn't occur.

Example

```
/* Get PWM capture 1 rising transition flag.*/
if(DrvPWM_GetCaptureIntStatus(DRVPWM_CAP1, DRVPWM_CAP_RISING_FLAG)==TRUE)
printf("PWM capture 1 rising transition occurs!\n");
else if(DrvPWM_GetCaptureIntStatus(DRVPWM_CAP1, DRVPWM_CAP_RISING_FLAG)==FALSE)
printf("PWM capture 1 rising transition doesn't occur!\n");
```

DrvPWM_ClearCaptureIntStatus

Prototype

void DrvPWM_ClearCaptureIntStatus(uint8_t u8Capture, uint8_t u8IntType);

Description

Clear the rising / falling transition indicator flag

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

u8IntType [in]

Specify the Capture Latched Indicator.

DRVPWM_CAP_RISING_FLAG : The capture rising indicator flag.

DRVPWM_CAP_FALLING_FLAG : The capture falling indicator flag.

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Clear PWM capture 1 falling transition flag.*/
DrvPWM_ClearCaptureIntStatus(DRVPWM_CAP1, DRVPWM_CAP_FALLING_FLAG);
```

DrvPWM_Open

Prototype

void DrvPWM_Open(void);

Description

Enable PWM engine clock and reset PWM.

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Enable PWM engine clock and reset PWM engine. */

DrvPWM_Open();

DrvPWM_Close

Prototype

void DrvPWM_Close(void);

Description

Disable PWM engine clock and the Capture Input / PWM Output Enable function.

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Disable PWM timer 0~3 output, PWM capture 0~3 input and disable PWM engine clock.*/

DrvPWM_Close ( );

DrvPWM_EnableDeadZone

Prototype

void DrvPWM_EnableDeadZone(uint8_t u8Timer, uint8_t u8Length, int32_t i32EnableDeadZone);

Description

This function is used to set the dead zone length and enable/disable Dead Zone function.

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0 or DRVPWM_TIMER1: PWM timer 0 & PWM timer 1.
DRVPWM_TIMER2 or DRVPWM_TIMER3: PWM timer 2 & PWM timer 3.

u8Length [in]

Specify Dead Zone Length: 0~255. The unit is one period of PWM clock.

i32EnableDeadZone [in]

Enable DeadZone (1) / Diasble DeadZone (0)

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Enable PWM timer 0 and time 1 Dead-Zone function. PWM timer 0 and PWM timer 1 became a complementary pair. Set Dead-Zone time length to 100 and the unit time of Dead-Zone length which is the same as the unit of received PWM timer clock.*/

```
uint8_t u8DeadZoneLength = 100;

DrvPWM_EnableDeadZone (DRVPWM_TIMER0, u8DeadZoneLength, 1);
```

Sample code

/* Enable Timer0 and Timer1 Dead-Zone function and set Dead-Zone interval to 5us. Dead zone interval = [1 / (PWM0 engine clock source / sPt.u8PreScale / sPt.u8ClockSelector)]* u8DeadZoneLength = unit time * u8DeadZoneLength = [1/(12000000 / 6 / 1)] * 10 = 5us */

```
uint8_t u8DeadZoneLength = 10; // Set dead zone length to 10 unit time

/* PWM Timer property */

sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;

sPt.u8HighPulseRatio = 30;       /* High Pulse period : Total Pulse period = 30 : 100 */

sPt.i32Inverter = 0;

sPt.u32Duty = 1000;

sPt.u8ClockSelector = DRVPWM_CLOCK_DIV_1;

sPt.u8PreScale = 6;

u8Timer = DRVPWM_TIMER0;

/* Select PWM engine clock source */

DrvPWM_SelectClockSource(u8Timer, DRVPWM_EXT_12M);

/* Set PWM Timer0 Configuration */

DrvPWM_SetTimerClk(u8Timer, &sPt);

/* Enable Output for PWM Timer0 */

DrvPWM_SetTimerIO(u8Timer, 1);

/* Enable Output for PWM Timer1 */

DrvPWM_SetTimerIO(DRVPWM_TIMER1, 1);

/* Enable Timer0 and Time1 dead zone function and Set dead zone length to 10 */

DrvPWM_EnableDeadZone(u8Timer, u8DeadZoneLength, 1);

/* Enable the PWM Timer 0 */

DrvPWM_Enable(u8Timer, 1);
```

DrvPWM_Enable

Prototype

void DrvPWM_Enable(uint8_t u8Timer, int32_t i32Enable);

Description

This function is used to enable PWM timer / capture function

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

i32Enable [in]

Enable (1) / Disable (0)

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Enable PWM timer 0 function. */
DrvPWM_Enable(DRVPWM_TIMER0, 1);
/* Enable PWM capture 1 function.*/
DrvPWM_Enable(DRVPWM_CAP1, 1);
```

DrvPWM_SetTimerClk

Prototype

uint32_t DrvPWM_SetTimerClk(uint8_t u8Timer, S_DRVPWM_TIME_DATA_T *sPt);

Description

This function is used to configure the frequency/pulse/mode/inverter function. The function will set the frequency property automatically when user set a nonzero frequency value by *u32Frequency*. When the setting of frequency value (*u32Frequency*) is not specified, i.e set

to 0, user needs to provide the setting of clock selector, prescale and duty to generate desired frequency.

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

***sPt [in]**

It includes the following parameter

Parameters	Description
u32Frequency	The timer/capture frequency (Hz)
u8HighPulseRatio	High pulse ratio (1~100)
u8Mode	DRVPWM_ONE_SHOT_MODE / DRVPWM_AUTO_RELOAD_MODE
bInverter	Inverter Enable (1) / Inverter Disable (0)
u8ClockSelector	Clock Selector DRVPWM_CLOCK_DIV_1: PWM input clock is divided by 1 DRVPWM_CLOCK_DIV_2: PWM input clock is divided by 2 DRVPWM_CLOCK_DIV_4: PWM input clock is divided by 4 DRVPWM_CLOCK_DIV_8: PWM input clock is divided by 8 DRVPWM_CLOCK_DIV_16: PWM input clock is divided by 16 (The parameter takes effect when *u32Frequency* = 0)
u8PreScale	Prescale (1 ~ 255). If the u8PreScale is set to 0, the timer will stop The PWM input clock = PWM source clock / (*u8PreScale* + 1) (The parameter takes effect when *u32Frequency* = 0)
u32Duty	Pulse duty (0x1 ~ 0x10000) (The parameter takes effect when *u32Frequency* = 0 or *u8Timer* = DRVPWM_CAP0/DRVPWM_CAP1/DRVPWM_CAP2/DRVPWM_CAP3)

Include

Driver/DrvPWM.h

Return Value

The actual specified PWM frequency (Hz).

Example

/* PWM timer 0 output 1KHz waveform and duty cycle of waveform is 20% */

Method 1:

Fill sPt.u32Frequency = 1000 to determine the waveform frequency and

DrvPWM_SetTimerClk() will set the frequency property automatically.

```
/* PWM Timer property */
sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;
sPt.u8HighPulseRatio = 20; /* High Pulse peroid : Total Pulse peroid = 20 : 100 */
sPt.i32Inverter = 0;
sPt.u32Frequency = 1000; // Set 1KHz to PWM timer output frequency
u8Timer = DRVPWM_TIMER0;
/* Select PWM engine clock */
DrvPWM_SelectClockSource(u8Timer, DRVPWM_HCLK);
/* Set PWM Timer0 Configuration */
DrvPWM_SetTimerClk(u8Timer, &sPt);
/* Enable Output for PWM Timer0 */
DrvPWM_SetTimerIO(u8Timer, 1);
/* Enable Interrupt Sources of PWM Timer 0 and install call back function */
DrvPWM_EnableInt(u8Timer, 0, DRVPWM_PwmIRQHandler);
/* Enable the PWM Timer 0 */
DrvPWM_Enable(u8Timer, 1);
```

Method 2:

Fill sPt.u8ClockSelector, sPt.u8PreScale and sPt.u32Duty to determine the output waveform frequency.

Assume HCLK frequency is 22MHz.

Output frequency = HCLK freq / sPt.u8ClockSelector / sPt.u8PreScale / sPt.u32Duty = 22MHz / 1 / 22 / 1000 = 1KHz

```
/* PWM Timer property */
sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;
sPt.u8HighPulseRatio = 20; /* High Pulse peroid : Total Pulse peroid = 20 : 100 */
sPt.i32Inverter = 0;
sPt.u8ClockSelector = DRVPWM_CLOCK_DIV_1;
sPt.u8PreScale = 22;
sPt.u32Duty = 1000;
u8Timer = DRVPWM_TIMER0;

/* Select PWM engine clock and user must know the HCLK frequency*/
DrvPWM_SelectClockSource(u8Timer, DRVPWM_HCLK);
/* Set PWM Timer0 Configuration */
```

```
DrvPWM_SetTimerClk(u8Timer, &sPt);
/* Enable Output for PWM Timer0 */
DrvPWM_SetTimerIO(u8Timer, 1);
/* Enable Interrupt Sources of PWM Timer0 and install call back function */
DrvPWM_EnableInt(u8Timer, 0, DRVPWM_PwmIRQHandler);
/* Enable the PWM Timer 0 */
DrvPWM_Enable(u8Timer, 1);
```

DrvPWM_SetTimerIO

Prototype

void DrvPWM_SetTimerIO(uint8_t u8Timer, int32_t i32Enable);

Description

This function is used to enable/disable PWM timer/capture I/O function

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.

i32Enable [in]

Enable (1) / Disable (0)

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Enable PWM timer 0 output.*/
DrvPWM_SetTimerIO(DRVPWM_TIMER0, 1);
/* Disable PWM timer 0 output. */
DrvPWM_SetTimerIO(DRVPWM_TIMER0, 0);
/* Enable PWM capture 3 input. */
```

DrvPWM_SetTimerIO(DRVPWM_CAP3, 1);

/* Disable PWM capture timer 3 input

DrvPWM_SetTimerIO(DRVPWM_CAP3, 0);

DrvPWM_SelectClockSource

Prototype

void DrvPWM_SelectClockSource(uint8_t u8Timer, uint8_t u8ClockSourceSelector);

Description

This function is used to select PWM0&PWM1 and PWM2&PWM3 engine clock source. It means PWM0/1 use one clock source. PWM2/3 can use another clock source and so on. In other words, if user change PWM timer 0 clock source from external 12MHz to internal 22MHz, the clock source of PWM timer 1 will also be changed from external 12MHz to internal 22MHz. Furthermore, it is possible to set the clock source of PWM1 to be external 12MHz and set the clock source of PWM2 to be external 32.768Hz.

Parameter

u8Timer [in]

Specify the timer

DRVPWM_TIMER0 or DRVPWM_TIMER1: PWM timer 0 & PWM timer 1.

DRVPWM_TIMER2 or DRVPWM_TIMER3: PWM timer 2 & PWM timer 3.

u8ClockSourceSelector [in]

To set the clock source of specified PWM timer. it could be DRVPWM_EXT_12M / DRVPWM_EXT_32K / DRVPWM_HCLK / DRVPWM_INTERNAL_22M. where DRVPWM_EXT_12M is external crystal clock. DRVPWM_EXT_32K is external 32.768 Hz crystal clock. DRVPWM_HCLK is HCLK. DRVPWM_INTERNAL_22M is internal 22.1184 MHz crystal clock

Include

Driver/DrvPWM.h

Return Value

None

Example

Select PWM timer 0 and PWM timer 1 engine clock source from HCLK.

DrvPWM_SelectClockSource(DRVPWM_TIMER0, DRVPWM_HCLK);

Select PWM timer 2 and PWM timer 3 engine clock source from external 12MHz.

DrvPWM_SelectClockSource(DRVPWM_TIMER3, DRVPWM_EXT_12M);

DrvPWM_GetVersion

Prototype

uint32_t DrvPWM_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvPWM.h

Return Value

PWM driver current version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

```
/* Get PWM driver current version number */

int32_t i32PWMVersionNum ;

i32PWMVersionNum = DrvPWM_GetVersion();
```

10. PS2 Driver

10.1. PS2 Introduction

PS/2 device controller provides basic timing control for PS/2 communication. All communication between the device and the host is managed through the CLK and DATA pins. The device controller generates the CLK signal after receiving a request to send, but host has ultimate control over communication. DATA sent from the host to the device is read on the rising edge and DATA sent from device to the host is change after rising edge. One 16 bytes Tx FIFO is used to reduce CPU intervention, but no RX FIFO. Software can select 1 to 16 bytes Tx FIFO depth for a continuous transmission.

Because PS/2 device controller is very simple, we recommend using macro as much as possible for speed consideration. Because no Rx FIFO, so DrvPS2_Read only read one byte; but DrvPS2_Write can write any length bytes to host

Default PS/2 interrupt handler has been implemented, it's PS2_IRQHandler. User can issue DrvPS2_EnableInt () function to install interrupt call back function and issue DrvPS2_DisableInt () to uninstall interrupt call back function.

10.2. PS2 Feature

The PS/2 device controller includes following features:

- APB interface compatible.
- Host communication inhibit and request to send detection.
- Reception frame error detection
- Programmable 1 to 16 bytes TX FIFO to reduce CPU intervention. But no Rx FIFO
- Double buffer for RX.
- Software override bus.

10.3. Constant Defination

Constant Name	Value	Description
DRVPS2_RXINT	0x00000001	PS2 RX interrupt
DRVPS2_TXINT	0x00000002	PS2 TX interrupt
DRVPS2_TXFIFODEPTH	16	TX FIFO depth

10.4. Macros

_DRVPS2_OVERRIDE

Prototype

void _DRVPS2_OVERRIDE(bool state);

Description

This macro is used to enable/disable software to control DATA/CLK line.

Parameter

state **[in]**

Specify software override or not. 1 means to enable software override PS/2 CLK/DATA pin state, 0 means to disable it.

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable Software to control DATA/CLK pin */
_DRVPS2_OVERRIDE(1)
/* Disable Software to control DATA/CLK pin */
_DRVPS2_OVERRIDE(0)
```

_DRVPS2_PS2CLK

Prototype

void _DRVPS2_PS2CLK(bool state);

Description

This macro can force PS2CLK high or low regardless of the internal state of the device controller if _DRVPS2_OVERRIDE called. 1 means high, 0 means low

Parameter

state **[in]**

Specify PS2CLK line high or low

Include

Driver/DrvPS2.h

Return Value

None.

Note

The macro is meaningful only when DRVPS2_OVERRIDE has been called.

Example

```
/* Force PS2CLK pin high. */
_DRVPS2_PS2CLK(1);
/* Force PS2CLK pin low. */
_DRVPS2_PS2CLK(0);
```

_DRVPS2_PS2DATA

Prototype

void _DRVPS2_PS2DATA(bool state);

Description

This macro can force PS2DATA high or low regardless of the internal state of the device controller if _DRVPS2_OVERRIDE called. 1 means high, 0 means low.

Parameter

u16Port [in]

Specify PS2DATA line high or low

Include

Driver/DrvPS2.h

Return Value

None.

Note

The macro is meaningful only when _DRVPS2_OVERRIDE has been called.

Example

```
/* Force PS2DATApin high. */
_DRVPS2_PS2DATA (1);
/* Force PS2DATA pin low. */
_DRVPS2_PS2DATA (0);
```

_DRVPS2_CLRFIFO

Prototype

void DRVPS2_CLRFIFO();

Description

The macro is used to clear TX FIFO.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Clear TX FIFO. */
_DRVPS2_CLRFIFO();
```

_DRVPS2_ACKNOTALWAYS

Prototype

void _DRVPS2_ACKNOTALWAYS();

Description

The macro is used to enable ack not always.. If parity error or stop bit is not received correctly, acknowledge bit will not be sent to host at 12th clock.,

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable ackknowlwde NOT always. */
_DRVPS2_ACKNOTALWAYS()
```

_DRVPS2_ACKALWAYS

Prototype

void _DRVPS2_ACKALWAYS();

Description

The macro is used to enable ack always.. If parity error or stop bit is not received correctly, acknowledge bit will always send acknowledge to host at 12th clock for host to device communication

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable ackknowlwde always. */
_DRVPS2_ACKALWAYS()
```

_DRVPS2_RXINTENABLE

Prototype

void _DRVPS2_RXINTENABLE();

Description

The macro is used to enable Rx interrupt. When acknowledge bit is sent from host to device, RX interrupt will happen

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable RX interrupt. */
_DRVPS2_RXINTENABLE();
```

_DRVPS2_RXINTDISABLE

Prototype

void _DRVPS2_RXINTDISABLE();

Description

The macro is used to disable Rx interrupt.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Disable RX interrupt. */
_DRVPS2_RXINTDISABLE ();
```

_DRVPS2_TXINTENABLE

Prototype

void _DRVPS2_TXINTENABLE();

Description

The macro is used to enable TX interrupt. When STOP bit is transmitted, TX interrupt will happen.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable TX interrupt. */
_DRVPS2_TXINTENABLE();
```

_DRVPS2_TXINTDISABLE

Prototype

void _DRVPS2_TXINTDISABLE ();

Description

The macro is used to disable TX interrupt.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Disable TX interrupt. */
_DRVPS2_TXINTDISABLE();
```

_DRVPS2_PS2ENABLE

Prototype

void _RVPS2_PS2ENABLE ();

Description

The macro is used to enable PS/2 device controller.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable PS/2 device controller. */
_DRVPS2_PS2ENABLE ();
```

_DRVPS2_PS2DISABLE

Prototype

void _DRVPS2_PS2DISABLE ();

Description

The macro is used to disable PS/2 device controller.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Disable PS/2 device controller. */
_DRVPS2_PS2DISABLE ();
```

_DRVPS2_TXFIFO

Prototype

void _DRVPS2_TXFIFO(depth);

Description

The macro is used to set TX FIFO depth. The range of TX FIFO is [1,16]

Parameter

data [in] : Specify TX FIFO depth(1~16).

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Set TX FIFO depth to 16 bytes. */
_DRVPS2_TXFIFO(16);
/* Set TX FIFO depth to 1 bytes. */
_DRVPS2_TXFIFO(1);
```

_DRVPS2_SWOVERRIDE

Prototype

void _DRVPS2_SWOVERRIDE(bool data, bool clk);

Description

The macro is used to set PS2DATA and PS2CLK line by software override. It's equal to these macos:

_DRVPS2_PS2DATA(data);

```
_DRVPS2_PS2CLK(clk);
_DRVPS2_OVERRIDE(1);
```

Parameter

data [in]

Specify PS2DATA line high or low

clk [in]

Specify PS2CLK line high or low

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Set PS2DATA to high and set PS2CLK to low. */
_DRVPS2_SWOVERRIDE(1, 0);
/* Set PS2DATA to low and set PS2CLK to high. */
_DRVPS2_SWOVERRIDE(0, 1);
```

_DRVPS2_INTCLR

Prototype

```
void _DRVPS2_INTCLR(uint8_t intclr) ;
```

Description

The macro is used to clear interrup status.

Parameter

intclr [in]

Specify to clear TX or RX interrupt. Intclr=0x1 for clear RX interrupt; Intclr=0x2 for clear TX interrupt; Intclr=0x3 for clear RX and TX interrupt

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Clear RX interrupt. */
_DRVPS2_INTCLR(1);
```

```
/* Clear TX interrupt. */
_DRVPS2_INTCLR(2);
/* Clear TX and RX interrupt. */
_DRVPS2_INTCLR(3);
```

_DRVPS2_RXDATA

Prototype

uint8_t _DRVPS2_RXDATA();

Description

Reads 1 byte from the receive register.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

One byte data received.

Example

```
/* Read one byte from PS/2 receive data register. */
uint8_t u8ReceiveData;
u8ReceiveData = _DRVPS2_RXDATA();
```

_DRVPS2_TXDATAWAIT

Prototype

void _DRVPS2_TXDATAWAIT(uint32_t data, uint32_t len);

Description

The macro is used to wait TX FIFO EMPTY, set TX FIFO depth(length-1) and fill TX FIFO0-3(Register PS2TXDATA0). Data is sent immediately if bus is in IDLE state. The range of length is from 1 to 16 bytes. If the transfer size is more than 4 bytes, user should call DRVPS2_TXDATA1~3() after calling _DRVPS2_TXDATAWAIT() to transfer remind data.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1

Parameter

data [in]

Specify the data sent

len [in]

Specify the length of the data sent. Unit is byte. Range is [1, 16]

Include

Driver/DrvPS2.h

Return Value

None

Example

/* Wait TX FIFO empty and then write 16 bytes to TX FIFO. The sixteen bytes consist of 0x01 to 0x16. */

_DRVPS2_TXDATAWAIT(0x04030201, 16);

_DRVPS2_TXDATA1(0x08070605);

_DRVPS2_TXDATA2(0x0C0B0A09);

_DRVPS2_TXDATA3(0x100F0E0D);

/* Wait TX FIFO empty and then write 5 bytes to TX FIFO. The six bytes consist of 0x01 to 0x05. */

_DRVPS2_TXDATAWAIT(0x04030201, 5);

_DRVPS2_TXDATA1(0x05);

/* Wait TX FIFO empty and then write 3 bytes to TX FIFO. The three bytes consist of 0x01 to 0x03. */

_DRVPS2_TXDATAWAIT(0x030201, 3);

_DRVPS2_TXDATA

Prototype

void _DRVPS2_TXDATA(uint32_t data, uint32_t len);

Description

The macro is used to set TX FIFO depth and fill TX FIFO0-3. But not wait TX FIFO EMPTY. Data is sent if bus is in IDLE state immediately. The range of len is [1, 16]

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data sent

len [in]

Specify the length of the data sent. Unit is byte. Range is [1, 16]

Include

Driver/DrvPS2.h

Return Value

None

Note

If the transfer size is more than 4 bytes, user should issue _DRVPS2_TXDATA1~3() after issuing _DRVPS2_TXDATA();

Example

```
/*Write 16 bytes to TX FIFO. The sixteen bytes consist of 0x01 to 0x16. */
_DRVPS2_TXDATA(0x04030201, 16);
_DRVPS2_TXDATA1(0x08070605);
_DRVPS2_TXDATA2(0x0C0B0A09);
_DRVPS2_TXDATA3(0x100F0E0D);
/* Write 5 bytes to TX FIFO. The six bytes consist of 0x01 to 0x05. */
_DRVPS2_TXDATA(0x04030201, 5);
_DRVPS2_TXDATA1(0x05);
/* Write 3 bytes to TX FIFO. The three bytes consist of 0x01 to 0x03. */
_DRVPS2_TXDATA(0x030201, 3);
```

_DRVPS2_TXDATA0

Prototype

```
void _DRVPS2_TXDATA0(uint32_t data);
```

Description

The macro is used to fill TX FIFO0-3. But not wait TX FIFO EMPTY and not set TX FIFO depth. Data is sent if bus is in IDLE state immediately.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Write 16 bytes to TX FIFO. The sixteen bytes consist of 0x01 to 0x16. */
while(_DRVPS2_ISTXEMPTY()==0);
_DRVPS2_TXFIFO(16);
```

```
_DRVPS2_TXDATA0(0x04030201);
_DRVPS2_TXDATA1(0x08070605);
_DRVPS2_TXDATA2(0x0C0B0A09);
_DRVPS2_TXDATA3(0x100F0E0D);
```

_DRVPS2_TXDATA1

Prototype

void _DRVPS2_TXDATA1(uint32_t data);

Description

The macro is used to fill TX FIFO4-7. But not wait TX FIFO EMPTY and not set TX FIFO depth.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent

Include

Driver/DrvPS2.h

Return Value

None

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_TXDATA2

Prototype

void _DRVPS2_TXDATA2(uint32_t data);

Description

The macro is used to fill TX FIFO8-11. But not wait TX FIFO EMPTY and not set TX FIFO depth.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent

Include

Driver/DrvPS2.h

Return Value

None

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_TXDATA3

Prototype

void _DRVPS2_TXDATA3(uint32_t data);

Description

The macro is used to fill TX FIFO12-15. But not wait TX FIFO EMPTY and not set TX FIFO depth.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent.

Include

Driver/DrvPS2.h

Return Value

None

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_ISTXEMPTY

Prototype

uint8_t _DRVPS2_ISTXEMPTY();

Description

The macro is used to check TX FIFO whether or not empty

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

TX FIFO empty status.

0: TX FIFO is empty.

1: TX FIFO is not empty.

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_ISFRAMEERR

Prototype

uint8_t _DRVPS2_ISFRAMEERR();

Description

The macro is used to check whether or not frame error happen. For host to device communication, if STOP bit is not received it is a frame error. If frame error occurs, DATA line may keep at low state after 12th clock. At this moment, software override PS2CLK to send clock till PS2DATA release to high state. After that, device sends a “Resend” command to host

Parameter

None

Include

Driver/DrvPS2.h

Return Value

Frame error status.

0: Not frame error.

1: Frame error.

Example

```
/* Check Frame error and print the result. */
if(_DRVPS2_ISFRAMEERR()==1)
    printf(“Frame error happen!!\n”);
else
    printf(“Frame error not happen!!\n”);
```

_DRVPS2_ISRXBUSY

Prototype

uint8_t _DRVPS2_ISRXBUSY();

Description

The macro is used to check whether or not Rx busy. If busy it indicates that PS/2 device is currently receiving data

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

RX busy flag.

0: RX is not busy,

1: RX is busy.

Example

```
/* Check RX is busy or not. */
if(_DRVPS2_ISRXBUSY()==1)
        printf("RX is busy!\n");
else
        printf("RX is not busy!\n");
```

10.5. Functions

DrvPS2_Open

Prototype

int32_t DrvPS2_Open();

Description

This function is used to init PS/2 IP. It includes enable PS2 clock, enable PS/2 controller, clear FIFO, set TX FIFO depth to default value zero.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS.

Example

```
/* Initialize PS/2 IP. */
DrvPS2_Open();
```

DrvPS2_Close

Prototype

void DrvPS2_Close();

Description

This function is used to disable PS2 controller, disable PS/2 clock and set TX FIFO depth to default value zero

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None

Example

```
/* Close PS2 IP. */
DrvPS2_Close ();
```

DrvPS2_EnableInt

Prototype

```
int32_t DrvPS2_EnableInt (
    uint32_t    u32InterruptFlag,
    PFN_DRVPS2_CALLBACK pfncallback
);
```

Description

This function is used to enable TX/RX interrupt and install interrupt call back function.

Parameter

u32InterruptFlag [in]

Specify TX/RX interrupt flag that will be enable. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT

pfncallback [in]

Specify the interrupt call back function. When PS2 interrupt happen, this function will be called

Include

Driver/ DrvPS2.h

Return Value

E_SUCCESS

Example

```
/* Enable TX/RX interrupt, install TX/RX call back function: PS2Mouse_IRQHandler(); */
DrvPS2_EnableInt(DRVPS2_TXINT| DRVPS2_RXINT, PS2Mouse_IRQHandler);
```

DrvPS2_DisableInt

Prototype

```
void DrvPS2_DisableInt(uint32_t        u32InterruptFlag);
```

Description

This function is used to disable Tx/Rx interrupt and uninstall interrupt call back function..

Parameter

u32InterruptFlag [in]

Specify TX/RX interrupt flag that will be disabled. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT.

Include

Driver/ DrvPS2.h

Return Value

None

Example

```
/* Disable TX/RX interrupt and uninstall TX and RX call back function. */
DrvPS2_DisableInt(DRVPS2_TXINT| DRVPS2_RXINT);
```

DrvPS2_IsIntEnabled

Prototype

```
uint32_t DrvPS2_IsIntEnabled(uint32_t        u32InterruptFlag);
```

Description

This function is used to check whether or not interrupt be enabled.

Parameter

u32InterruptFlag [in]

Specify TX/RX interrupt flag that will be checked. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT.

Include

Driver/DrvPS2.h

Return Value

0 : No interrupt be enable.

2 : TX interrupt be enable

4 : RX interrupt be enable

6 : TX and RX interrupt be enable.

Example

```
/* Check TX and RX interrupt enable or not enable. */
uint32_u32TXRXIntEnable
u32TXRXIntEnable = DrvPS2_IsIntEnabled(DRVPS2_TXINT| DRVPS2_RXINT)
if(u32TXRXIntEnable ==0)
printf("No interrupt be enable!!\n");
else if(u32TXRXIntEnable ==2)
printf("TX interrupt be enable!!\n");
else if(u32TXRXIntEnable ==4)
printf("RX interrupt be enable!!\n");
else if(u32TXRXIntEnable ==6)
printf("TX and RX interrupt be enable!!\n");
```

DrvPS2_ClearIn

Prototype

uint32_t DrvPS2_ClearInt(uint32_t u32InterruptFlag);

Description

This function is used to clear interrupt status.

Parameter

U32InterruptFlag [in]

Specify Tx/Rx interrupt flag that will be cleared. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS: Success.

Example

```
/* Clear TX interrupt. */
DrvPS2_ClearInt(DRVPS2_TXINT);
/* Clear RX interrupt. */
DrvPS2_ClearInt(DRVPS2_RXINT);
/* Clear TX and RX interrupt. */
DrvPS2_ClearInt(DRVPS2_TXINT| DRVPS2_RXINT);
```

DrvPS2_GetIntStatus

Prototype

```
int8_t DrvPS2_GetIntStatus(uint32_t  u32InterruptFlag);
```

Description

This function is used to check interrupt status. If interrupt that be checked happens it will return TRUE

Parameter

U32InterruptFlag [in]

Specify TX/RX interrupt flag that will be checked. It can be DRVPS2_TXINT or DRVPS2_RXINT

Include

Driver/DrvPS2.h

Return Value

TRUE: interrupt that be checked happens

FALSE: interrupt that be checked doesn’t happen.

Example

```
/* Check TX interrupt status */
int8_t i8InterruptStatus;
i8InterruptStatus = DrvPS2_GetIntStatus(DRVPS2_TXINT);
if(i8InterruptStatus==TRUE)
   printf(“TX interrupt that be checked happens”\n);
else
   printf(“TX interrupt doesn’t happen”\n);
```

DrvPS2_SetTxFIFODepth

Prototype

void DrvPS2_SetTxFIFODepth(uint16_t u16TxFIFODepth);

Description

This function is used to set TX FIFO depth. The function will call macro DRVPS2_TXFIFO to set TX FIFO depth

Parameter

u16TxFIFODepth [in]

Specify TX FIFO depth. The range can be [1, 16]

Include

Driver/DrvPS2.h

Return Value

None

Example

```
/* Set TX FIFO depth to 16 bytes. */
DrvPS2_SetTxFIFODepth(16);
/* Set TX FIFO depth to 1 byte. */
DrvPS2_SetTxFIFODepth(1);
```

DrvPS2_Read

Prototype

int32_t DrvPS2_Read(uint8_t *pu8RxBuf);

Description

The function is used to read one byte to the buffer of pu8RxBuf. The function will call macro DRVPS2_RXDATA to receive data

Parameter

pu8RxBuf [out]

the buffer is used to contain byte received. The size of buffer needs one byte only

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS: Success.

Example

```
/* Read RX data and print it. */
uint8_t u8RXData;
DrvPS2_Read(&u8RXData);
printf("RX data is %x\n", u8RXData);
```

DrvPS2_Write

Prototype

```
int32_t
DrvPS2_Write(
    uint32_t      *pu32TxBuf,
    uint32_t      u32WriteBytes
);
```

Description

The function is used to write the buffer of pu32TxBuf and the length of u32WriteBytes to host. If data count sent is less than 16 bytes, please use macro DRVPS2_TXDATAxxx for speed

Parameter

pu32TxBuf [in]

the data that will be sent to host.

u32WriteBytes [in]

the length of data that will be sent to host.

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS: Success.

Example

```
/* Write 64 bytes to TX buffer and TX buffer will send the 64 bytes out. */
uint32_t au32TXData[64];
DrvPS2_Write(au32TXData, 64);
```

DrvPS2_GetVersion

Prototype

```
int32_t DrvPS2_GetVersion(void);
```

Description

Return the current version number of driver.

Include

Driver/ DrvPS2.h

Return Value

PS2 driver current version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

```
/* Get PS/2 driver current version number */

int32_t i32Ps2VersionNum;

i32Ps2VersionNum = DrvPS2_GetVersion ();
```

11. FMC Driver

11.1. FMC Introduction

NuMicro™ NUC122 series equips with 64/32k bytes on chip embedded flash for application program memory (APROM), 4k bytes for ISP loader program memory (LDROM), and user configuration (Config0). User configuration block provides several bytes to control system logic, like flash security lock, boot select, brown out voltage level, ..., and so on. NuMicro™ NUC122 series also provide additional 4k bytes data flash for user to store some application depended data before chip power off.

11.2. FMC Feature

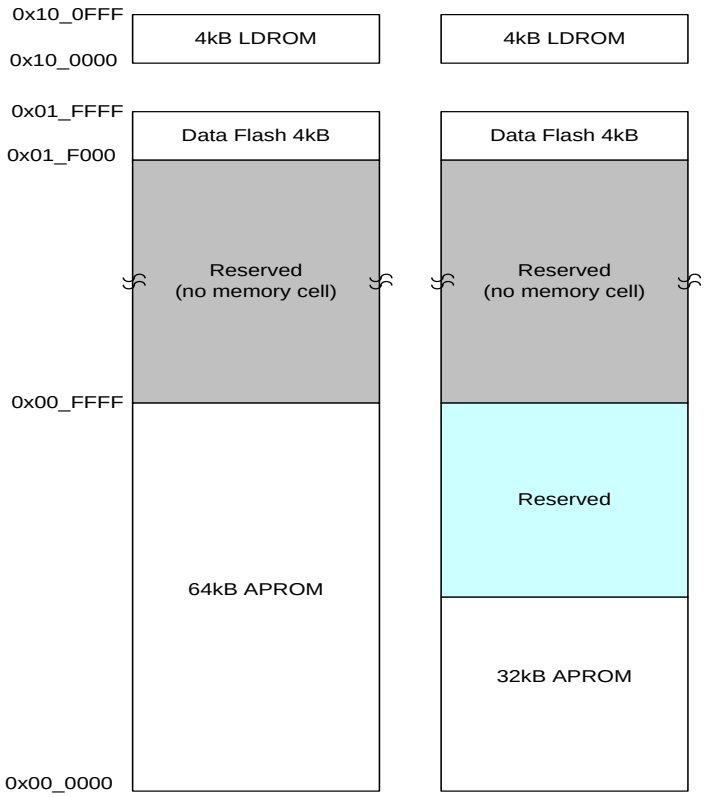
The FMC includes following features:

- 64/32kB application program memory (APROM).
- 4kB in system programming loader program memory (LDROM).
- 4kB data flash with 512 bytes page erase unit for user to store data
- Provide user configuration to control system logic.
- APROM cannot be updated when the MCU is running in APROM; LDROM can not be updated when the MCU is running in LDROM

Memory Address Map

Block Name	Size	Start Address	End Address
AP ROM	32 KB 64 KB	0x00000000	0x00007FFF 0x0000FFFF
Data Flash	4 KB	0x0001F000	0x0001FFFF
LD ROM	4KB	0x00100000	0x00100FFF
User Configuration	1 word	0x00300000	0x00300000

Flash Memory Structure



32/64kB Flash Memory Structure

11.3. Type Definition

E_FMC_BOOTSELECT

Enumeration identifier	Value	Description
E_FMC_APROM	0	Boot from APROM
E_FMC_LDROM	1	Boot from LDROMl

11.4. Functions

DrvFMC_EnableISP

Prototype

void DrvFMC_EnableISP (void);

Description

To enable ISP function.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Enable ISP function */
DrvFMC_EnableISP ( );
```

DrvFMC_DisableISP

Prototype

```
void      DrvFMC_DisableISP (void);
```

Description

To disable ISP function.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Disable ISP function */
```

DrvFMC_DisableISP ( );

DrvFMC_BootSelect

Prototype

void DrvFMC_BootSelect(E_FMC_BOOTSELECT boot);

Description

To select next booting from APROM or LDROM.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

boot [in]

Specify E_FMC_APROM or E_FMC_LDROM.

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_BootSelect (E_FMC_LDROM);      /* Next booting from LDROM */
DrvFMC_BootSelect (E_FMC_APROM);      /* Next booting from APROM */
```

DrvFMC_GetBootSelect

Prototype

E_FMC_BOOTSELECT DrvFMC_GetBootSelect(void);

Description

To get current boot select setting.

Parameter

None.

Include

Driver/DrvFMC.h

Return Value

E_FMC_APROM The current boot select setting is in APROM

E_FMC_LDROM The current boot select setting is in LDROM

Example

E_FMC_BOOTSELECT e_bootSelect

/* Check this booting is from APROM or LDROM */

e_bootSelect = DrvFMC_GetBootSelect ( );

DrvFMC_EnableLDUpdate

Prototype

void DrvFMC_EnableLDUpdate (void);

Description

To enable LDROM update function. LDROM can be updated if LDROM update function is enabled when the MCU runs in APROM.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

/* Enable LDROM update function */

DrvFMC_EnableLDUpdate ( );

DrvFMC_DisableLDUpdate

Prototype

void DrvFMC_DisableLDUpdate (void);

Description

To disable LDROM update function.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Disable LDROM update function */
DrvFMC_DisableLDUpdate ( );
```

DrvFMC_EnableConfigUpdate

Prototype

```
void      DrvFMC_EnableConfigUpdate (void);
```

Description

To enable Config update function. If Congif update function is enabled, the user configuration can be update regardless of MCU is running in APROM or LDROM.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Enable Config update function */
DrvFMC_EnableConfigUpdate ( );
```

DrvFMC_DisableConfigUpdate

Prototype

void DrvFMC_DisableConfigUpdate (void);

Description

To disable Config update function.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Disable Config update function */
DrvFMC_DisableConfigUpdate ( );
```

DrvFMC_Write

Prototype

int32_t DrvFMC_Write (uint32_t u32addr, uint32_t u32data);

Description

To write word data into APROM, LDROM, Data Flash or Config. The Memory Map of APROM is depended on the product of NuMicro™ NUC122 series. Please refer to NuMicro™ NUC122 Series Products Selection Guide of Appendix for Flash size. The corresponding function in Config0 are described in FMC Section of TRM in details.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u32addr [in]

Word address of APROM, LDROM, Data Flash or Config.

u32data [in]

Word data to be programmed into APROM, LDROM, Data Flash or Config.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

```
/* Program word data 0x12345678 into address 0x1F000 */
DrvFMC_Write (0x1F000, 0x12345678);
```

DrvFMC_Read

Prototype

```
int32_t    DrvFMC_Read (uint32_t u32addr, uint32_t * u32data);
```

Description

To read data from APROM, LDROM, Data Flash or Config. The Memory Map of APROM is depended on the product of NuMicro™ NUC122 series. Please refer to NuMicro™ NUC122 Series Products Selection Guide of Appendix for Flash size.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u32addr [in]

Word address of APROM, LDROM, Data Flash or Config.

u32data [out]

The word data to store data from APROM, LDROM, Data Flash or Config.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

```
uint32_t u32Data;
```

```
/* Read word data from address 0x1F000, and read data is stored to u32Data */
DrvFMC_Read (0x1F000, &u32Data);
```

DrvFMC_Erase

Prototype

```
int32_t    DrvFMC_Erase (uint32_t u32addr);
```

Description

To page erase APROM, LDROM, Data Flash or Config. The flash page erase unit is 512 bytes. The Memory Map of APROM is depended on the product of NuMicro™ NUC122 series. Please refer to NuMicro™ NUC122 Series Products Selection Guide of Appendix for Flash size.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u32addr [in]

Flash page base address of APROM, LDROM and Data Flash, or Config0 addrsss.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

```
/* Page Erase from 0x1F000 to 0x1F1FF */
DrvFMC_Erase (0x1F000);
```

DrvFMC_WriteConfig

Prototype

```
int32_t    DrvFMC_WriteConfig(uint32_t u32data0);
```

Description

To erase Config0 and write data into Config0. The corresponding functions in Config0 are described in FMC Section of TRM in details.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

u32data0 [in]

Word data to be programmed into Config0.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

```
/* Program word data 0xFFFFFFFE into Config0 */
DrvFMC_WriteConfig (0xFFFFFFFE);
```

DrvFMC_ReadDataFlashBaseAddr

Prototype

uint32_t DrvFMC_ReadDataFlashBaseAddr (void);

Description

To read data flash base address. The base address is fixed at 0x1F000 for NUC122 series.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

Data Flash base address

Example

```
uint32_t u32Data;
/* Read Data Flash base address */
u32Data = DrvFMC_ReadDataFlashBaseAddr ( );
```

DrvFMC_EnableLowFreqOptMode

Prototype

void DrvFMC_EnableLowFreqOptMode (void);

Description

To enable flash access low frequency optimization mode. It can improve flash access performance when CPU runs at low frequency.

Note 1

Set this bit only when HCLK ≤ 20 MHz. If HCLK > 20 MHz, CPU will fetch wrong code and cause fail result.

Note 2

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Enable flash access low frequency optimization mode */
DrvFMC_EnableLowFreqOptMode ( );
```

DrvFMC_DisableLowFreqOptMode

Prototype

void DrvFMC_DisableLowFreqOptMode (void);

Description

To disable flash access low frequency optimization mode.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

/* Disable flash access low frequency optimization mode */

DrvFMC_DisableLowFreqOptMode ( );

DrvFMC_EnableMiddleFreqOptMode

Prototype

void DrvFMC_EnableMiddleFreqOptMode (void);

Description

To enable flash access middle frequency optimization mode. It can improve flash access performance when CPU runs at middle frequency.

Note 1

Set this bit only when 20 MHz ≤ HCLK ≤ 40 MHz. If HCLK > 40 MHz, CPU will fetch wrong code and cause fail result.

Note 2

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

/* Enable flash access middle frequency optimization mode */

DrvFMC_EnableMiddleFreqOptMode ( );

DrvFMC_DisableMiddleFreqOptMode

Prototype

void DrvFMC_DisableMiddleFreqOptMode (void);

Description

To disable flash access middle frequency optimization mode.

Note

Please make sure that the Register Write-Protection function has been unlocked before using this API. User can check the status of the Register Write-Protection function with DrvSYS_IsProtectedRegLocked ( ).

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Disable flash access middle frequency optimization mode */
DrvFMC_DisableMiddleFreqOptMode ( );
```

DrvFMC_GetVersion

Prototype

uint32_t DrvFMC_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

Version number:

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

12. USB Driver

12.1. Introduction

This article is provided for manufacturers who are using USB Device controller to complete their USB applications. It is assumed that the reader is familiar with the Universal Serial Bus Specification, Revision 1.1.

12.2. Feature

- Conform to USB2.0 Full speed, 12Mbps.
- Provide 1 interrupt source with 4 interrupt events.
- Support Control, Bulk, Interrupt, and Isochronous transfers.
- Suspend when no bus signaling for 3 ms.
- Provide 6 endpoints for configuration.
- Include 512 bytes internal SRAM as USB buffer.
- Provide remote wake-up capability.

12.3. USB Framework

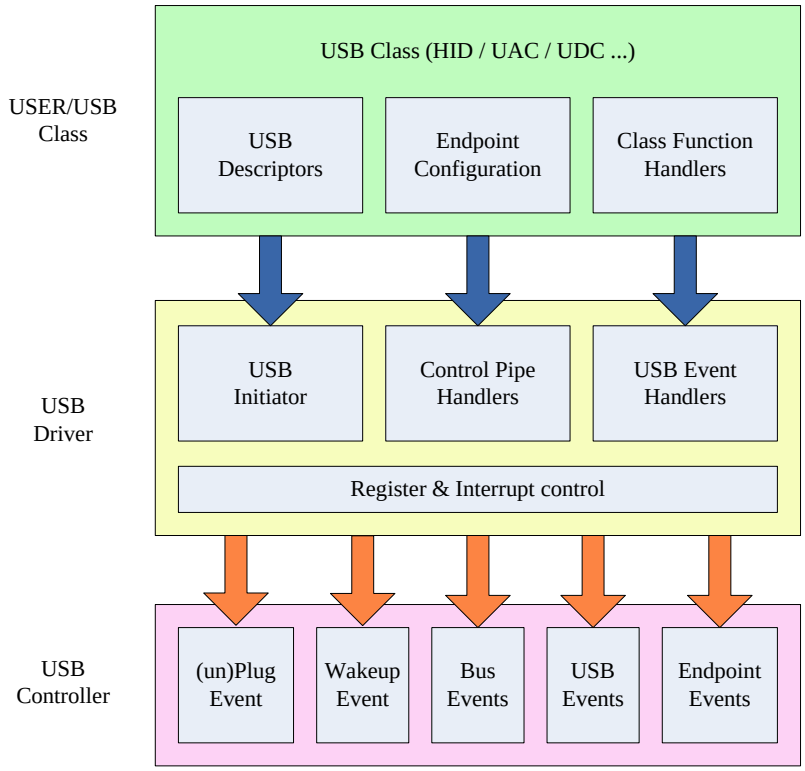


Figure 14-1: USB Framework

Above figure shows the framework of USB device library. The lowest layer is USB controller. The USB controller will raise different interrupt events according to USB, BUS and floating detection status. All the events are handled by USB driver by relative event handlers. USB driver also take care the basic handler of control pipe of USB protocol. Most function dependent handlers and USB descriptors must be provided by user applications or USB class definitions.

12.4. Call Flow

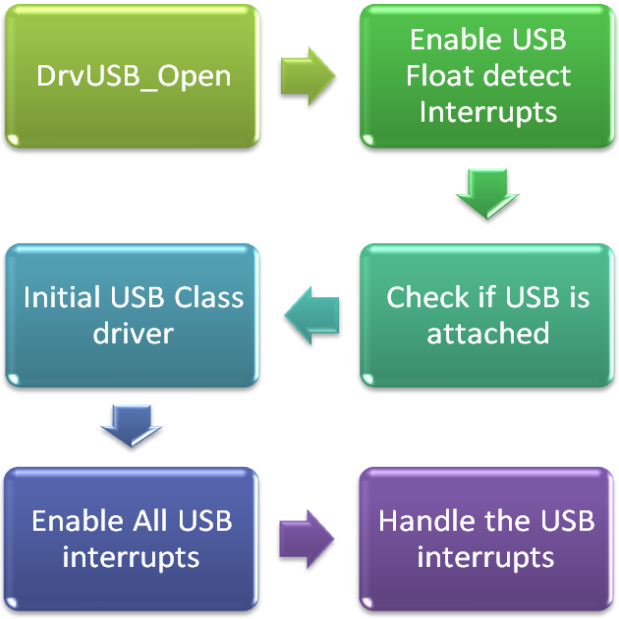


Figure 14-2: USB Driver Call Flow

The above figure shows the call flow of USB driver. The DrvUSB_Open is used to initial the USB device controller. Then USB floating detection is enabled to detect USB plug/un-plug events. If USB attached, it need to call the USB class driver to initial USB class specified descriptions, event handlers. Finally, all related USB interrupts are enabled to handle the USB events.

12.5. Constant Definition

USB Register Address

Constant Name	Value	Description
USBD_INTEN	0x40060000	USB Interrupt Enable Register Address
USBD_INTSTS	0x40060004	USB Interrupt Event Status Register Address
USBD_FADDR	0x40060008	USB Device Function Address Register Address
USBD_EPSTS	0x4006000C	USB Endpoint Status Register Address Address
USBD_ATTR	0x40060010	USB Bus Status and Attribution Register Address
USBD_FLDETB	0x40060014	USB Floating Detected Register Address
USBD_BUFSEG	0x40060018	Setup Token Buffer Segmentation Register Address
USBD_BUFSEG0	0x40060020	Endpoint 0 Buffer Segmentation Register Address
USBD_MXPLD0	0x40060024	Endpoint 0 Maximal Payload Register Address

Constant Name	Value	Description
USBD_CFG0	0x40060028	Endpoint 0 Configuration Register Address
USBD_CFGP0	0x4006002C	Endpoint 0 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG1	0x40060030	Endpoint 1 Buffer Segmentation Register Address
USBD_MXPLD1	0x40060034	Endpoint 1 Maximal Payload Register Address
USBD_CFG1	0x40060038	Endpoint 1 Configuration Register Address
USBD_CFGP1	0x4006003C	Endpoint 1 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG2	0x40060040	Endpoint 2 Buffer Segmentation Register Address
USBD_MXPLD2	0x40060044	Endpoint 2 Maximal Payload Register Address
USBD_CFG2	0x40060048	Endpoint 2 Configuration Register Address
USBD_CFGP2	0x4006004C	Endpoint 2 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG3	0x40060050	Endpoint 3 Buffer Segmentation Register Address
USBD_MXPLD3	0x40060054	Endpoint 3 Maximal Payload Register Address
USBD_CFG3	0x40060058	Endpoint 3 Configuration Register Address
USBD_CFGP3	0x4006005C	Endpoint 3 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG4	0x40060060	Endpoint 4 Buffer Segmentation Register Address
USBD_MXPLD4	0x40060064	Endpoint 4 Maximal Payload Register Address
USBD_CFG4	0x40060068	Endpoint 4 Configuration Register Address
USBD_CFGP4	0x4006006C	Endpoint 4 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG5	0x40060070	Endpoint 5 Buffer Segmentation Register Address
USBD_MXPLD5	0x40060074	Endpoint 5 Maximal Payload Register Address
USBD_CFG5	0x40060078	Endpoint 5 Configuration Register Address
USBD_CFGP5	0x4006007C	Endpoint 5 Set Stall and Clear In/Out Ready Control Register Address
USBD_DRVSE0	0x40060090	USB Drive SE0 Control Register Address
USB_SRAM_BASE	0x40060100	USB Buffer Base Address

INTEN Register Bit Definition

Constant Name	Value	Description
INTEN_INNAK	0x00008000	Active NAK interrupt function and its status flag for IN token
INTEN_WAKEUP_EN	0x00000100	Wake Up Function Enable
INTEN_WAKEUP_IE	0x00000008	USB Wake Up Interrupt Enable
INTEN_FLDET_IE	0x00000004	Floating Detect Interrupt Enable
INTEN_USB_IE	0x00000002	USB Event Interrupt Enable

Constant Name	Value	Description
INTEN_BUS_IE	0x00000001	Bus Event Interrupt Enable

INTSTS Register Bit Definition

Constant Name	Value	Description
INTSTS_SETUP	0x80000000	Setup Event Status
INTSTS_EPEVT5	0x00200000	Endpoint 5's USB Event Status
INTSTS_EPEVT4	0x00100000	Endpoint 4's USB Event Status
INTSTS_ EPEVT 3	0x00080000	Endpoint 3's USB Event Status
INTSTS_ EPEVT 2	0x00040000	Endpoint 2's USB Event Status
INTSTS_ EPEVT 1	0x00020000	Endpoint 1's USB Event Status
INTSTS_ EPEVT 0	0x00010000	Endpoint 0's USB Event Status
INTSTS_WAKEUP_STS	0x00000008	Wakeup Interrupt Status
INTSTS_FLDET_STS	0x00000004	Floating Detected Interrupt Status
INTSTS_USB_STS	0x00000002	USB event Interrupt Status
INTSTS_BUS_STS	0x00000001	BUS Interrupt Status

ATTR Register Bit Definition

Constant Name	Value	Description
ATTR_BYTEM	0x00000400	CPU access USB RAM Size Mode Select
ATTR_PWRDN	0x00000200	Power down PHY, low active
ATTR_DPPU_EN	0x00000100	Pull-up resistor on D+ enable
ATTR_USB_EN	0x00000080	USB Controller Enable
ATTR_RWAKEUP	0x00000020	Remote Wake Up
ATTR_PHY_EN	0x00000010	PHY Function Enable
ATTR_TIMEOUT	0x00000008	Time Out Status
ATTR_RESUME	0x00000004	Resume Status
ATTR_SUSPEND	0x00000002	Suspend Status
ATTR_USBRST	0x00000001	USB Reset Status

Confiuration Register Bit Definition

Constant Name	Value	Description
CFG_CSTALL	0x00000200	Clear STALL Response

Constant Name	Value	Description
CFG_DSQ_SYNC	0x00000080	Data Sequence Synchronization
CFG_STATE	0x00000060	Endpoint STATE
CFG_EPT_IN	0x00000040	IN endpoint
CFG_EPT_OUT	0x00000020	Out endpoint
CFG_ISOCH	0x00000010	Isochronous Endpoint
CFG_EP_NUM	0x0000000F	Endpoint Number

Extera-Confiuration Register Bit Definition

Constant Name	Value	Description
CFGP_SSTALL	0x00000002	Set the device to respond STALL
CFGP_CLRRDY	0x00000001	Clear Ready

12.6. Macro

_DRVUSB_ENABLE_MISC_INT

Prototype

```
void _DRVUSB_ENABLE_MISC_INT (
    uint32_t      u32Flags
);
```

Description

Enable/Disable miscellaneous interrupts including USB event, Wakeup event, Float-detection event and bus event.

Parameter

u32Flags [in]

USB interrupt events. It can be following flags.

IEF_WAKEUP: Wakeup interrupt flag.

IEF_FLD: Float-detection interrupts flag.

IEF_USB: USB event interrupt flag.

IEF_BUS: Bus event interrupt flag.

u32Flag = 0 will disable all USB interrupts.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
_DRVUSB_ENABLE_MISC_INT(0);    /* Disable All USB-related interrupts. */
_DRVUSB_ENABLE_MISC_INT(IEF_WAKEUP | IEF_WAKEUPEN | IEF_FLD |
IEF_USB | IEF_BUS); /* Enable wakeup, float-detection, USB and bus interrupts */
```

_DRVUSB_ENABLE_WAKEUP

Prototype

void _DRVUSB_ENABLE_WAKEUP (void);

Description

Enable USB wakeup function. If USB wakeup function is enabled, any activity of USB bus could be used to wakeup CPU from power down.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

```
_DRVUSB_ENABLE_WAKEUP(); /* To enable the USB wakeup function */
```

_DRVUSB_DISABLE_WAKEUP

Prototype

void _DRVUSB_DISABLE_WAKEUP (void);

Description

Disable USB wakeup function. If USB wakeup function is disable, USB can't used to wakeup up CPU from power down.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_DISABLE_WAKEUP(); /* To avoid wakeup CPU by USB */

_DRVUSB_ENABLE_WAKEUP_INT

Prototype

void _DRVUSB_ENABLE_WAKEUP_INT (void);

Description

Enable wakeup interrupt. USB will raise a wakeup event interrupt when wakeup interrupt is enabled.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

DRVUSB_ENABLE_WAKEUP_INT() /* To enable wakeup event interrupt */

_DRVUSB_DISABLE_WAKEUP_INT

Prototype

void _DRVUSB_DISABLE_WAKEUP_INT (void);

Description

Disable wakeup interrupt to avoid USB raise an interrupt when wakeup from power down.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

DRVUSB_DISABLE_WAKEUP_INT () /* To disable wakeup event interrupt */

_DRVUSB_ENABLE_FLDET_INT

Prototype

void _DRVUSB_ENABLE_FLDET_INT (void);

Description

Enable float-detection interrupt to raise an interrupt when USB plug-in or un-plug.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_FLDET_INT() /* To enable float-detection interrupt */

_DRVUSB_DISABLE_FLDET_INT

Prototype

void _DRVUSB_DISABLE_FLDET_INT (void);

Description

Disable float-detection interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_DISABLE_FLDET_INT() /* To disable float-detection interrupt */

_DRVUSB_ENABLE_USB_INT

Prototype

void _DRVUSB_ENABLE_USB_INT (void);

Description

Enable USB interrupt. It could be used to control USB interrupt only and _DRVUSB_ENABLE_MISC_INT() can be used to control all USB related interrupts at the same time.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_USB_INT () /* To enable USB interrupt */

_DRVUSB_DISABLE_USB_INT

Prototype

void _DRVUSB_DISABLE_USB_INT (void);

Description

Disable USB interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

DRVUSB DISABLE _USB_INT () /* To disable USB interrupt */

_DRVUSB_ENABLE_BUS_INT

Prototype

void _DRVUSB_ENABLE_BUS_INT (void);

Description

Enable USB bus interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_BUS_INT () /* To enable USB bus interrupt */

_DRVUSB_DISABLE_BUS_INT

Prototype

void _DRVUSB_DISABLE_BUS_INT (void);

Description

Disable bus interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_DISABLE_BUS_INT () /* To disable USB bus interrupt */

_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL

Prototype

void _DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL (

uint32_t u32EPId

);

Description

Clear specified USB endpoint hardware In/Out Ready and respond STALL,

Parameter

u32EPId[in]

EP Identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL(3) /* To clear ready flag of USB endpoint identity 3 and let it to response STALL. */

Notes

Here, EP (endpoint) identity means number of USB device hardware, not USB endpoint number defined by USB standard.

_DRVUSB_CLEAR_EP_READY

Prototype

void _DRVUSB_CLEAR_EP_READY (

uint32_t u32EPId

);

Description

Clear EP In/Out Ready.

Parameter

u32EPId[in]

EP Identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_CLEAR_EP_READY(1) /* To clear ready flag of USB endpoint identity 1. */

_DRVUSB_SET_SETUP_BUF

Prototype

void _DRVUSB_SET_SETUP_BUF (

uint32_t u32BufAddr

);

Description

Specify buffer address for Setup transaction. This buffer is used to store setup token data and its size is fixed to be 8 bytes according to USB standard. Therefore, the buffer address must be 8 bytes alignment.

Parameter

u32BufAddr [in]

Buffer address for setup token. It could be USB_BA+0x100 ~ USB_BA+0x2F8 where USB_BA is 0x40060000.

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_SET_SETUP_BUF(0x400602F8) /* Set the setup packet address to 0x400602F8 */

_DRVUSB_SET_EP_BUF

Prototype

void _DRVUSB_SET_EP_BUF (

uint32_t u32EPId,

uint32_t u32BufAddr

);

Description

Specify buffer address for specified hardware endpoint identity and it must be 8 bytes alignment. This buffer would be used to buffer the data of IN/OUT USB transaction. The buffer size used by IN/OUT USB transaction is dependent on maximum payload of related endpoint identity.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

u32BufAddr [in]

Used to set buffer address and valid address is from 0x40060100 ~ 0x400602F8. Furthermore, buffer address + maximum payload size must less than 0x400602FF.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
_DRVUSB_SET_EP_BUF(1, 0x40060100)    /* Set the buffer address of endpoint identity 1 to 0x40060100 */
```

_DRVUSB_TRIG_EP

Prototype

```
void _DRVUSB_TRIG_EP (
    uint32_t      u32EPId,
    uint32_t      u32TrigSize
);
```

Description

Trigger next transaction for specified endpoint identity and the transaction size is also defined at the same time.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5) for trigger Data In or Out transaction.

u32TrigSize [in]

For Data Out transaction, it means maximum data size transferred from Host; for Data In transaction, it means how many data transferred to Host.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Trigger the transaction of endpoint identity 1 and the transaction payload size is 64 bytes */
_DRVUSB_TRIG_EP (1, 64)
```

_DRVUSB_GET_EP_DATA_SIZE

Prototype

```
uint32_t
_DRVUSB_GET_EP_DATA_SIZE (
    uint32_t      u32EPId
);
```

Description

Length of data transmitted to or received from Host for specified endpoint identity.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

For IN endpoint: length of data transmitting to host in bytes.

For OUT endpoint: Actual length of data receiving from host in bytes.

Example

```
/* To get the size of received data of endpoint identity 1. */
size = _DRVUSB_GET_EP_DATA_SIZE(1);
```

_DRVUSB_SET_EP_TOG_BIT

Prototype

```
void      _DRVUSB_SET_EP_TOG_BIT (
    uint32_t      u32EPId,
    int32_t       bData0
)
```

Description

Specify Data0 or Data1 for specified endpoint identity. This bit will toggle automatically after Host ACK the IN token.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

bData0 [in]

Specify DATA0 or DATA1 for IN transaction. TRUE is for DATA0, FALSE is for DATA1.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To set the toggle bit as DATA0 for endpoint identity 1 */
_DRVUSB_SET_EP_TOG_BIT(1, TRUE);
```

_DRVUSB_SET_EVENT_FLAG

Prototype

```
void _DRVUSB_SET_EVENT_FLAG (
    uint32_t      u32Data
);
```

Description

Set Interrupt Event Flag to clear them. The interrupt event flags are write one clear.

Parameter

u32Data [in]

Specify the event to be clear. It could be

Events	Value	Description
EVF_SETUP	0x80000000	Got a setup token event
EVF_EPTF5	0x00200000	Got USB event from endpoint identity 5
EVF_EPTF4	0x00100000	Got USB event from endpoint identity 4
EVF_EPTF3	0x00080000	Got USB event from endpoint identity 3
EVF_EPTF2	0x00040000	Got USB event from endpoint identity 2
EVF_EPTF1	0x00020000	Got USB event from endpoint identity 1
EVF_EPTF0	0x00010000	Got USB event from endpoint identity 0
EVF_WAKEUP	0x00000008	Got a wakeup event
EVF_FLD	0x00000004	Got float-detection event
EVF_USB	0x00000002	Got USB event include endpoint events or setup event
EVF_BUS	0x00000001	Got USB bus event

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_SET_EVENT_FLAG(EVF_BUS); /* Clear USB bus event */

_DRVUSB_SET_EVENT_FLAG(EVF_BUS | EVF_FLD); /* Clear USB bus event and float-detection event */

_DRVUSB_GET_EVENT_FLAG

Prototype

uint32_t

_DRVUSB_GET_EVENT_FLAG (void);

Description

Get Interrupt Event Flags

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return EVF register value. Please refer to _DRVUSB_SET_EVENT_FLAG() for detail event information.

Example

u32Events = _DRVUSB_GET_EVF(); /* Get events */

_DRVUSB_CLEAR_EP_STALL

Prototype

void _DRVUSB_CLEAR_EP_STALL (

uint32_t u32EPId

);

Description

Stop to force specified endpoint identity to respond STALL to host.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_CLEAR_EP_STALL(1);/* Clear the STALL of endpoint identity 1 */

_DRVUSB_TRIG_EP_STALL

Prototype

```
void _DRVUSB_TRIG_EP_STALL (
    uint32_t      u32EPId
);
```

Description

Force EPx (x = 0 ~ 5) to response STALL

Parameter

u32EPId[in]

EP identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_TRIG_EP_STALL (1); /* Force to STALL endpoint identity 1 */

_DRVUSB_CLEAR_EP_DSQ_SYNC

Prototype

```
void _DRVUSB_CLEAR_EP_DSQ_SYNC (
uint32_t      u32EPId
```

);

Description

Clear the endpoint toggle bit to DATA0,i.e force the toggle bit to be DATA0. This bit will toggle automatically after IN token ack from host.

Parameter

u32EPId [in]

EP Identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Force the toggle bit of endpoint identity 2 to be DATA0 */
_DRVUSB_CLEAR_EP_DSQ_SYNC (2);
```

_DRVUSB_SET_CFG

Prototype

```
void _DRVUSB_SET_CFG (
        uint32_t    u32CFGNum,
        uint32_t    u32Data
);
```

Description

This macro is used to set USB CFG register.

Parameter

u32CFGNum [in]

CFG number (valid value: 0 ~ 5).

u32Data [in]

Specify the setting for CFG register.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Set USB CFG2 control register as 0x3 */
_DRVUSB_SET_CFG (2, 0x3);
```

_DRVUSB_GET_CFG

Prototype

```
uint32_t
_DRVUSB_GET_CFG (
    uint32_t      u32CFGNum
);
```

Description

Get current setting of USB CFG register.

Parameter

u32CFGNum [in]

CFG number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

Return specified CFG register value

Example

```
/* Get the setting of USB CFG2 control register */
u32Cfg = _DRVUSB_GET_CFG (3);
```

_DRVUSB_SET_FADDR

Prototype

```
void _DRVUSB_SET_FADDR (
    uint32_t      u32Addr
)
```

Description

To set USB device address. The valid address is from 0 ~ 127.

Parameter

u32Addr [in]

The USB device address and it could be 0 ~ 127.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Set the USB devcie address as 3 */
_DRVUSB_SET_FADDR (3);
```

_DRVUSB_GET_FADDR

Prototype

```
uint32_t
_DRVUSB_GET_FADDR (void)
```

Description

To get USB device address.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return USB device address.

Example

```
/* Get USB devcie address */
u32Addr = _DRVUSB_GET_FADDR ();
```

_DRVUSB_GET_EPSTS

Prototype

```
uint32_t
_DRVUSB_GET_EPSTS (void)
```

Description

Get USB endpoint states register (EPSTS) value. The states register could be used to idendity the detail information of USB event. For detail information of EPSTS, please refere to NuMicro™ Technical Reference Manual.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return STS register value

Example

```
/* Get USB STS register value */
u32Reg = _DRVUSB_GET_STS();
```

_DRVUSB_SET_CFGP

Prototype

```
void _DRVUSB_SET_CFGP(
    uint8_t     u8CFGPNum,
    uint32_t    u32Data
);
```

Description

To set extra configuration register (CFGP). The CFGP register could be used to STALL the endpoint and clear endpoint ready flag.

CFGP[1]: STALL control bit. Set '1' to force the endpoint to response STALL to host.

CFGP[0]: Ready flag and it is write one clear.

Parameter

u8CFGPNum[in]

CFGP register number (valid value: 0 ~ 5).

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* To STALL the endpoint identity 1. */

_DRVUSB_SET_CFGP(1, 0x2);

_DRVUSB_GET_CFGP

Prototype

uint32_t

_DRVUSB_GET_CFGP (

uint32_t u32CFGPNum

);

Description

Get the value of extra configuration register (CFGP)

Parameter

u32CFGPNum[in]

CFGP register number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

Return CFGP register value

Example

/* Get the register value of CFG1 */

_DRVUSB_GET_CFGP(1);

_DRVUSB_ENABLE_USB

Prototype

void _DRVUSB_ENABLE_USB (void)

Description

Enable USB, PHY and use remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Enable USB, PHY and remote wakeup. */

_DRVUSB_ENABLE_USB();

_DRVUSB_DISABLE_USB

Prototype

void _DRVUSB_DISABLE_USB (void)

Description

Disable USB, PHY but still enable remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Disable USB, PHY but still enable remote wakeup. */

_DRVUSB_DISABLE_USB();

_DRVUSB_DISABLE_PHY

Prototype

void _DRVUSB_DISABLE_PHY (void)

Description

Disable PHY and remote wake-up.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Disable PHY and remote wakeup. */

_DRVUSB_DISABLE_PHY();

_DRVUSB_ENABLE_SE0

Prototype

void _DRVUSB_ENABLE_SE0 (void)

Description

Force USB to drive SE0 to bus. It can be used to simulate unplug event to let host re-connect to device. For more information about SE0, please refer to USB standard.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Force bus to be SE0 state */

_DRVUSB_ENABLE_SE0();

_DRVUSB_DISABLE_SE0

Prototype

void _DRVUSB_DISABLE_SE0 (void)

Description

Stop to drive SE0 to USB bus.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Stop to drive SE0 state to USB bus */
_DRVUSB_DISABLE_SE0();
```

_DRVUSB_SET_CFG_EP0

Prototype

```
void _DRVUSB_SET_CFG_EP0 (
    uint32_t      u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 0. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 0 */
_DRVUSB_SET_CFG_EP0(0x2);
```

_DRVUSB_SET_CFG_EP1

Prototype

void _DRVUSB_SET_CFG_EP1 (

uint32_t u32Data

)

Description

Stall control and clear In/out ready flag of endpoint identity 1. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* To STALL endpoint identity 1 */

_DRVUSB_SET_CFG_EP1(0x2);

_DRVUSB_SET_CFG_EP2

Prototype

void _DRVUSB_SET_CFG_EP2 (

uint32_t u32Data

)

Description

Stall control and clear In/out ready flag of endpoint identity 2. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 2 */
_DRVUSB_SET_CFG_EP2(0x2);
```

_DRVUSB_SET_CFGP3

Prototype

```
void _DRVUSB_SET_CFG_EP3 (
    uint32_t      u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 3. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 3 */
_DRVUSB_SET_CFG_EP3(0x2);
```

_DRVUSB_SET_CFGP4

Prototype

```
void _DRVUSB_SET_CFG_EP4 (
    uint32_t      u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 4. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 4 */
_DRVUSB_SET_CFG_EP4(0x2);
```

_DRVUSB_SET_CFGP5

Prototype

```
void _DRVUSB_SET_CFG_EP5 (
   uint32_t      u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 5. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 5 */
_DRVUSB_SET_CFG_EP5(0x2);
```

12.7. Functions

DrvUSB_GetVersion

Prototype

uint32_t

DrvUSB_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Version number :

31:24	23:16	15:8	7:0
00000000	MAJOR_NUM	MINOR_NUM	BUILD_NUM

Example

```
/* To get module's version */
u32Version = DrvUSB_GetVersion();
```

DrvUSB_Open

Prototype

```
int32_t
DrvUsb_Open (
    void *        pVoid
)
```

Description

This function is used to reset USB controller, initial the USB endpoints, interrupts, and USB driver structures. It also used to call the relative handler when the USB is attached before USB driver opened. The user must provide the materials before they can call DrvUSB_Open, including sEpDescription, g_sBusOps.

sEpDescription:

The structure type of sEpDescription is as follows:

```
typedef struct
{
    //bit7 is directory bit, 1: input; 0: output
    uint32_t    u32EPAddr;
    uint32_t    u32MaxPacketSize;
    uint8_t * u8SramBuffer;
}S_DRVUSB_EP_CTRL;
```

This structure is used to set the endpoint number, maximum packet size, and buffer of specified endpoint hardware. There are 6 endpoints hardware available in NUC122 series USB controller.

g_sBusOps:

The structure type of g_sBusOps is as follows:

```
typedef struct
{
    PFN_DRVUSB_CALLBACK         apfnCallback;
    void *                      apCallbackArgu;
}S_DRVUSB_EVENT_PROCESS
```

It is used to install the USB bus event handler, such as follows:

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
    {NULL, NULL},                                   /* attach event callback */
    {NULL, NULL},                                   /* detach event callback */
    {DrvUSB_BusResetCallback, &g_HID_sDevice},      /* bus reset event callback */
    {NULL, NULL},                                   /* bus suspend event callback */
    {NULL, NULL},                                   /* bus resume event callback */
    {DrvUSB_CtrlSetupAck, &g_HID_sDevice},          /* setup event callback */
};
```

Parameter

pVoid

NULL	None
Callback function	If the pVoid is not NULL, it will be the callback function of USB

interrupt and it is called after DrvUSB_PreDispatchEvent in USB interrupt handler

Include

Driver/DrvUsb.h

Return Value

E_SUCCESS: Succeed

Example

```
/* To open USB device */
i32Ret = DrvUSB_Open(0);
if(i32Ret != E_SUCCESS)
    return i32Ret;
```

DrvUSB_Close

Prototype

void DrvUSB_Close (void);

Description

Close USB controller and disable USB interrupt.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* To close USB device */
DrvUSB_Close();
```

DrvUSB_PreDispatchEvent

Prototype

void DrvUSB_PreDispatchEvent(void);

Description

Pre-dispatch event base on EVF register.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To pre dispatch USB device events at IRQ handler */
USBD_IRQHandler()
{
        DrvUSB_PreDispatchEvent();
}
```

DrvUSB_DispatchEvent

Prototype

void DrvUSB_DispatchEvent(void)

Description

Dispatch misc and endpoint event. Misc event include attach/detach/bus reset/bus suspend/bus resume and setup ACK, Misc event's handler is defined by g_sBusOps[]. The user must provide g_sBusOps[] before using USB driver.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* To dispatch USB events to handle them by related callback funcitons. */
DrvUSB_DispatchEvent();
```

DrvUSB_IsData0

Prototype

int32_t DrvUSB_IsData0(uint32_t u32EpId)

Description

To check if the current DATA is DATA0. If it is false, then it should be DATA1.

Parameter

u32EpId The hardware endpoint id. The id could be 0~5.

Include

Driver/DrvUSB.h

Return Value

TRUE	The current data packet is DATA0
FALSE	The current data packet is DATA1

Example

```
/* Get toggle bit of endpoint identity 2 */
if(DrvUSB_IsData0(2) )
{
   /* The toggle bit of endpoint identity 2 is DATA0 */
}
```

DrvUSB_GetUsbState

Prototype

E_DRVUSB_STATE DrvUSB_GetUsbState(void)

Description

Get current USB state E_DRVUSB_STATE. The status list as follows:

USB Status	Description
eDRVUSB_DETACHED	The USB has been detached.
eDRVUSB_ATTACHED	The USB has been attached.
eDRVUSB_POWERED	The USB is powered.
eDRVUSB_DEFAULT	The USB is in normal state.
eDRVUSB_ADDRESS	The USB is in ADDRESS state.
eDRVUSB_CONFIGURED	The USB is in CONFIGURATION state.
eDRVUSB_SUSPENDED	The USB is suspended.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

To return the current USB state.

Example

```
/* Get current USB state */
eUsbState = DrvUSB_GetUsbState();
```

```
if (eUsbState == eDRVUSB_DETACHED)
{
        /* USB unplug */
}
```

DrvUSB_SetUsbState

Prototype

void DrvUSB_SetUsbState(E_DRVUSB_STATE eUsbState)

Description

To change current USB state. Please refer to DrvUSB_GetUsbState for available states.

Parameter

eUsbState The USB state.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Set current USB state */
DrvUSB_SetUsbState(eDRVUSB_DETACHED);
```

DrvUSB_GetEpIdentity

Prototype

uint32_t DrvUSB_GetEpIdentity(uint32_t u32EpNum, uint32_t u32EpAttr)

Description

To get endpoint index base on endpoint number and direction. The endpoint id is used to identify the hardware endpoint resource. The range of endpoint index could be 0 ~ 5. The endpoint number is assigned by software and it could be 0 ~ 15 according to USB standard. Host will access the device through relative endpoint number.

Parameter

u32EpNum	The endpoint number (0 ~ 15)
u32EpAttr	The endpoint number attribute. It could be EP_INPUT or EP_OUTPUT

Include

Driver/DrvUSB.h

Return Value

0~5	The endpoint id of specified endpoint address.
otherwise	Can't get relative endpoint id according to the input endpoint address.

Example

```
/* Get the hardware endpoint identity of USB OUT endpoint 3 */
u32EpId = DrvUSB_GetEpIdentity(3, EP_OUTPUT);
```

DrvUSB_GetEpId

Prototype

uint32_t DrvUSB_GetEpId(uint32_t u32EpNum)

Description

Get endpoint index base on endpoint address. This argument “u32EpNum” is different from DrvUSB_GetEPIdentity's because its argument includes direction bit (bit 7). eg: 0x81. If the bit 7 is high, it indicates this is EP_INPUT, otherwise it is EP_OUTPUT.

Parameter

u32EpNum	The endpoint address with direction information at bit 7.

Include

Driver/DrvUSB.h

Return Value

0~5	The endpoint id of specified endpoint address.
otherwise	Can't get relative endpoint id according to the input endpoint address.

Example

```
/* Get the hardware endpoint identity of USB IN endpoint 4 */
u32EpId = DrvUSB_GetEpIdentity(0x84);
```

DrvUSB_DataOutTrigger

Prototype

int32_t DrvUSB_DataOutTrigger(uint32_t u32EpNum, uint32_t u32Size)

Description

Trigger data out ready flag by write MXPLD register. It indicates the relative endpoint buffer is ready to receive data out packet.

Parameter

u32EpNum	The endpoint number (0~15)
u32Size	Maximum size want to receive from USB

Include

Driver/DrvUSB.h

Return Value

0	Succeed
<0	Can't get relative endpoint id according to the input endpoint address.

Example

```
/* Trigger endpoint number 2 to receive OUT packet of host and the maximum packet size is 64 bytes */
DrvUSB_DataOutTrigger(2, 64);
```

DrvUSB_GetOutData

Prototype

uint8_t * DrvUSB_GetOutData(uint32_t u32EpNum, uint32_t *u32Size)

Description

This function will return the buffer pointer of u32EpNum 's out USB SRAM buffer. User can use this pointer to get the data payload of current data out packet.

Parameter

u32EpNum	The endpoint number (0~15)
u32Size	Data size received from USB

Include

Driver/DrvUSB.h

Return Value

To return USB SRAM address.

Example

```
/* Get the buffer address and size of received data of endpoint number 2 */
pu8EpBuf = DrvUSB_GetOutData(2, &u32Size);
```

DrvUSB_DataIn

Prototype

int32_t DrvUSB_DataIn(uint32_t u32EpNum, const uint8_t * u8Buffer, uint32_t u32Size)

Description

Trigger ready flag for sending data after receive IN token from host, USB will send the data. if u8Buffer == NULL && u32Size == 0 then send DATA1 always else DATA0 and DATA1 by turns.

Parameter

u32EpNum	The endpoint number (0~15)
u8Buffer	The data buffer for DATA IN token
u32Size	The size of data buffer

Include

Driver/DrvUSB.h

Return Value

0	Successful
E_DRVUSB_SIZE_TOO_LONG	The size is larger than maximum packet size

Example

```
/* Prepare 2 bytes data for endpoint number 0 IN transaction. */
DrvUSB_DataIn(0, au8Data, 2);
```

DrvUSB_BusResetCallback

Prototype

void DrvUSB_BusResetCallback(void * pVoid)

Description

Bus reset handler. After receiving bus reset event, this handler will be called. It will reset USB address, accept SETUP packet and initial the endpoints.

Parameter

pVoid	Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
   {NULL, NULL},                                    /* attach event callback */
   {NULL, NULL},                                    /* detach event callback */
   {DrvUSB_BusResetCallback, &g_HID_sDevice},    /* bus reset event callback */
   {NULL, NULL},                                    /* bus suspend event callback */
   {NULL, NULL},                                    /* bus resume event callback */
   {DrvUSB_CtrlSetupAck, &g_HID_sDevice},         /* setup event callback */
};
```

DrvUSB_InstallClassDevice

Prototype

void * DrvUSB_InstallClassDevice(S_DRVUSB_CLASS *sUsbClass)

Description

Register USB class device to USB driver.

Parameter

sUsbClass USB class structure pointer.

Include

Driver/DrvUSB.h

Return Value

Return USB driver pointer

Example

```
/* Register USB class device to USB driver. */
g_HID_sDevice.device = (void *)DrvUSB_InstallClassDevice(&sHidUsbClass);
```

DrvUSB_InstallCtrlHandler

Prototype

```
int32_t DrvUSB_InstallCtrlHandler(
    void *    *device,
    S_DRVUSB_CTRL_CALLBACK_ENTRY    *psCtrlCallbackEntry,
    uint32_t    u32RegCnt
)
```

Description

Register ctrl pipe handler including SETUP ACK , IN ACK, OUT ACK handle for Standard/Vendor/Class command.

Parameter

device	USB driver device pointer.
psCtrlCallbackEntry	Handler structure pointer.
u32RegCnt	Handler structure size.

Include

Driver/DrvUSB.h

Return Value

0	Success
E_DRVUSB_NULL_POINTER	Null function pointer

Example

```
/* Register ctrl pipe handler. */
i32Ret = DrvUSB_InstallCtrlHandler(g_HID_sDevice.device, g_asCtrlCallbackEntry,
sizeof(g_asCtrlCallbackEntry) / sizeof(g_asCtrlCallbackEntry[0]));
```

DrvUSB_CtrlSetupAck

Prototype

void DrvUSB_CtrlSetupAck(void * pArgu)

Description

When SETUP ack interrupt happen, this function will be called. It will call SETUP handler that DrvUSB_InstallCtrlHandler registered base on command category and command.

Parameter

pArgu	Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
    {NULL, NULL},                                   /* attach event callback */
    {NULL, NULL},                                   /* detach event callback */
```

```
    {DrvUSB_BusResetCallback, &g_HID_sDevice},    /* bus reset event callback */
    {NULL, NULL},                                 /* bus suspend event callback */
    {NULL, NULL},                                 /* bus resume event callback */
    {DrvUSB_CtrlSetupAck, &g_HID_sDevice},        /* setup event callback */
};
```

DrvUSB_CtrlDataInAck

Prototype

void DrvUSB_CtrlDataInAck(void * pArgu)

Description

When IN ack interrupt happen, this function will be called. It will call IN ACK handler that DrvUSB_InstallCtrlHandler registered base on command category and command.

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* USB event call back */
S_DRVUSB_EVENT_PROCESS g_sUsbOps[12] =
{
    {DrvUSB_CtrlDataInAck, &g_HID_sDevice},/* ctrl pipe0 (EP address 0) In ACK callback */
    {DrvUSB_CtrlDataOutAck, &g_HID_sDevice},/* ctrl pipe0 (EP address 0) Out ACK callback */
    {HID_IntInCallback, &g_HID_sDevice},/* EP address 1 In ACK callback */
    {NULL, NULL},                                 /* EP address 1 Out ACK callback */
    {NULL, NULL},                                 /* EP address 2 In ACK callback */
    {HID_IntOutCallback, &g_HID_sDevice},/* EP address 2 Out ACK callback */
    {NULL, NULL},                                 /* EP address 3 In ACK callback */
    {NULL, NULL},                                 /* EP address 3 Out ACK callback */
    {NULL, NULL},                                 /* EP address 4 In ACK callback */
    {NULL, NULL},                                 /* EP address 4 Out ACK callback */
    {NULL, NULL},                                 /* EP address 5 In ACK callback */
    {NULL, NULL},                                 /* EP address 5 Out ACK callback */
};
```

DrvUSB_CtrlDataOutAck

Prototype

void DrvUSB_CtrlDataOutAck(void * pArgu)

Description

When OUT ack interrupt happen, this function will be called. It will call OUT handler that DrvUSB_RegisterCtrl registered base on command category and command.

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* USB event call back */
S_DRVUSB_EVENT_PROCESS g_sUsbOps[12] =
{
    {DrvUSB_CtrlDataInAck     , &g_HID_sDevice},/* ctrl pipe0 (EP address 0) In ACK callback */
    {DrvUSB_CtrlDataOutAck    , &g_HID_sDevice},/* ctrl pipe0 (EP address 0) Out ACK callback */
    {HID_IntInCallback          , &g_HID_sDevice},/* EP address 1 In ACK callback */
    {NULL, NULL},                                 /* EP address 1 Out ACK callback */
    {NULL, NULL},                                 /* EP address 2 In ACK callback */
    {HID_IntOutCallback           , &g_HID_sDevice},/* EP address 2 Out ACK callback */
    {NULL, NULL},                                 /* EP address 3 In ACK callback */
    {NULL, NULL},                                 /* EP address 3 Out ACK callback */
    {NULL, NULL},                                 /* EP address 4 In ACK callback */
    {NULL, NULL},                                 /* EP address 4 Out ACK callback */
    {NULL, NULL},                                 /* EP address 5 In ACK callback */
    {NULL, NULL},                                 /* EP address 5 Out ACK callback */
};
```

DrvUSB_CtrlDataInDefault

Prototype

void DrvUSB_CtrlDataInDefault(void * pVoid)

Description

IN ACK default handler. It is used to return ACK for next OUT token.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* If no control data IN callback installed, just use default one */
if (psEntry->pfnCtrlDataInCallback == NULL)
    psEntry->pfnCtrlDataInCallback = DrvUSB_CtrlDataInDefault;
```

DrvUSB_CtrlDataOutDefault

Prototype

void DrvUSB_CtrlDataOutDefault(void * pVoid)

Description

OUT ACK default handler. It is used to return zero data length packet when next IN token.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* If no control data OUT callback installed, just use default one */
if (psEntry->pfnCtrlDataOutCallback == NULL)
   psEntry->pfnCtrlDataOutCallback = DrvUSB_CtrlDataOutDefault;
```

DrvUSB_Reset

Prototype

void DrvUSB_Reset(uint32_t u32EpNum)

Description

Restore the specified CFGx and CFGPx registers according the endpoint number.

Parameter

u32EpNum The endpoint number to reset

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Reset endpoint number 2 */
DrvUSB_Reset(2);
```

DrvUSB_ClrCtrlReady

Prototype

void DrvUSB_ClrCtrlReady(void)

Description

Clear ctrl pipe ready flag that was set by MXPLD.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear control endpoint ready flag */
DrvUSB_ClrCtrlReady();
```

DrvUSB_ClrCtrlReadyAndTrigStall

Prototype

void DrvUSB_ClrCtrlReadyAndTrigStall(void);

Description

Clear control pipe ready flag that was set by MXPLD and send STALL.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear control pipe ready flag that was set by MXPLD and send STALL. */
DrvUSB_ClrCtrlReadyAndTrigStall();
```

DrvUSB_GetSetupBuffer

Prototype

uint32_t DrvUSB_GetSetupBuffer(void)

Description

Get setup buffer address of USB SRAM to read the received setup packet data.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Setup buffer address

Example

```
/* Get setup buffer address of USB SRAM. */
SetupBuffer = (uint8_t *)DrvUSB_GetSetupBuffer();
```

DrvUSB_GetFreeSRAM

Prototype

uint32_t DrvUSB_GetFreeSRAM(void)

Description

Get free USB SRAM buffer address after EP assign base on sEpDescription[i].u32MaxPacketSize in DrvUSB_Open. User can get this for dual buffer.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Free USB SRAM address

Example

```
/* Get the base address of free USB SRAM */
u32BaseAddr = DrvUSB_GetFreeSRAM();
```

DrvUSB_EnableSelfPower

Prototype

void DrvUSB_EnableSelfPower(void)

Description

Enable self-power attribution of USB device.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Set a flag to note the USB device is self-power */
DrvUSB_EnableSelfPower();
```

DrvUSB_DisableSelfPower

Prototype

void DrvUSB_DisableSelfPower(void)

Description

Disable self-power attribution of USB device.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear the flag to note the USB device is not self-power */
DrvUSB_ DisableSelfPower ();
```

DrvUSB_IsSelfPowerEnabled

Prototype

int32_t DrvUSB_IsSelfPowerEnabled(void)

Description

Self-power is enable or disable.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

TRUE	The device is self-powered.
FALSE	The device is bus-powered.

Example

```
/* Check if the USB device is self-power */
if(DrvUSB_IsSelfPowerEnabled())
{
   /* The USB device is self-power */
}
```

DrvUSB_EnableRemoteWakeup

Prototype

void DrvUSB_EnableRemoteWakeup(void)

Description

Enable remote wakeup attribution of USB device.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Set the flag to note the USB device supports remote wakeup */
```

DrvUSB_EnableRemoteWakeup();

DrvUSB_DisableRemoteWakeup

Prototype

void DrvUSB_DisableRemoteWakeup(void)

Description

Disable remote wakeup attribution.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Clear the flag to note the USB device doesn't support remote wakeup */

DrvUSB_DisableRemoteWakeup();

DrvUSB_IsRemoteWakeupEnabled

Prototype

int32_t DrvUSB_IsRemoteWakeupEnabled (int32_t * pbVoid)

Description

Return remote wakeup is enabling or disable.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

TRUE	Support remote wakeup
FALSE	Not support remote wakeup

Example

/* Check if the USB device supports remote wakeup. */

```
if(DrvUSB_ IsRemoteWakeupEnabled ())
{
   /* Remote wakeup enable flag is set */
}
```

DrvUSB_SetMaxPower

Prototype

int32_t DrvUSB_SetMaxPower(uint32_t u32MaxPower)

Description

Configure max power. The unit is 2mA. Maximum MaxPower 0xFA (500mA), default is 0x32 (100mA)

Parameter

u32MaxPower Maximum power value

Include

Driver/DrvUSB.h

Return Value

0: Successful

<0: Wrong maximum value

Example

```
/* Set the maximum power is 150mA */
DrvUSB_SetMaxPower(75);
```

DrvUSB_GetMaxPower

Prototype

int32_t DrvUSB_GetMaxPower(void)

Description

Get current max power. The unit is in 2mA, i.e. 0x32 is 100mA.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Return the maximum power. (2mA unit)

Example

```
/* Get the maximum power */
i32Power = DrvUSB_GetMaxPower();
```

DrvUSB_EnableUSB

Prototype

void DrvUSB_EnableUSB(S_DRVUSB_DEVICE *psDevice)

Description

Enable USB, PHY and remote wakeup.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Enable USB, PHY and remote wakeup function. */
DrvUSB_EnableUSB(psDevice);
```

DrvUSB_DisableUSB

Prototype

void DrvUSB_DisableUSB(S_DRVUSB_DEVICE * psDevice)

Description

Disable USB, PHY but keep remote wakeup function on.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Enable USB, PHY and remote wakeup function. */
DrvUSB_DisableUSB(psDevice);
```

DrvUSB_PreDispatchWakeupEvent

Prototype

void DrvUSB_PreDispatchWakeupEvent(S_DRVUSB_DEVICE *psDevice)

Description

Pre-dispatch wakeup event. This function does nothing and reserves for further usage

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

N/A

DrvUSB_PreDispatchFDTEvent

Prototype

void DrvUSB_PreDispatchFDTEvent(S_DRVUSB_DEVICE * psDevice)

Description

Pre-dispatch plug-in and plug-out event

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Pre-dispatch float-detection event. */
DrvUSB_PreDispatchFDTEvent(&gsUsbDevice);
```

DrvUSB_PreDispatchBusEvent

Prototype

void DrvUSB_PreDispatchBusEvent(S_DRVUSB_DEVICE *psDevice)

Description

Pre-dispatch BUS event

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Pre-dispatch bus event. */
DrvUSB_PreDispatchBusEvent(&gsUsbDevice);
```

DrvUSB_PreDispatchEPEvent

Prototype

void DrvUSB_PreDispatchEPEvent(S_DRVUSB_DEVICE * psDevice)

Description

Pre-dispatch EP event including IN ACK/IN NAK/OUT ACK/ISO end. This function is used to recognize endpoint events and record them for further processing of DrvUSB_DispatchEPEvent(). All EP event handlers are defined at g_sUsbOps[].

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Clear USB events individually instead of in total. Otherwise, incoming USB events may be cleared mistakenly. Pre-dispatch USB event. */

DrvUSB_PreDispatchEPEvent(&gsUsbDevice);

DrvUSB_DispatchWakeupEvent

Prototype

void DrvUSB_DispatchWakeupEvent(S_DRVUSB_DEVICE *psDevice)

Description

Dispatch wakeup event. This function does nothing and reserves for further usage.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

N/A

DrvUSB_DispatchMiscEvent

Prototype

void DrvUSB_DispatchMiscEvent(S_DRVUSB_DEVICE * psDevice)

Description

Dispatch Misc event. The event is set by attach/detach/bus reset/bus suspend/bus resume and setup ACK. Misc event's handler is defined at g_sBusOps[].

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Parsing the MISC events and call relative handles */

DrvUSB_DispatchMiscEvent(&gsUsbDevice);

DrvUSB_DispatchEPEvent

Prototype

void DrvUSB_DispatchEPEvent(S_DRVUSB_DEVICE * psDevice)

Description

Dispatch EP event, the event is set by DrvUSB_PreDispatchEPEvent() including IN ACK/IN NAK/OUT ACK/ISO end. The EP event's handler is defined at g_sUsbOps[].

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Parsing the endpoint events and call relative handlers */

DrvUSB_ DispatchEPEvent (&gsUsbDevice);

DrvUSB_CtrlSetupSetAddress

Prototype

void DrvUSB_CtrlSetupSetAddress(void * pVoid)

Description

Setup ACK handler for set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

/*ctrl pipe call back.*/

```
/*it will be call by DrvUSB_CtrlSetupAck, DrvUSB_CtrlDataInAck and DrvUSB_CtrlDataOutAck*/
/*if in ack handler and out ack handler is 0, default handler will be called */
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
   {REQ_STANDARD, SET_ADDRESS, DrvUSB_CtrlSetupSetAddress,
DrvUSB_CtrlDataInSetAddress, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupClearSetFeature

Prototype

void DrvUSB_CtrlSetupClearSetFeature(void * pVoid)

Description

Setup ACK handler for Clear feature command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
   {REQ_STANDARD, CLEAR_FEATURE, DrvUSB_CtrlSetupClearSetFeature, 0, 0,
&g_HID_sDevice}
};
```

DrvUSB_CtrlSetupGetConfiguration

Prototype

void DrvUSB_CtrlSetupGetConfiguration(void * pVoid)

Description

Setup ACK handler for Get configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
   {REQ_STANDARD, GET_CONFIGURATION, DrvUSB_CtrlSetupGetConfiguration, 0, 0,
&g_HID_sDevice}
};
```

DrvUSB_CtrlSetupGetStatus

Prototype

void DrvUSB_CtrlSetupGetStatus(void * pVoid)

Description

Setup ACK handler for Get status command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
   {REQ_STANDARD, GET_STATUS, DrvUSB_CtrlSetupGetStatus, 0, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupGetInterface

Prototype

void DrvUSB_CtrlSetupGetInterface(void * pVoid)

Description

Setup ACK handler for Get interface command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, GET_INTERFACE, DrvUSB_CtrlSetupGetInterface, 0, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupSetInterface

Prototype

void DrvUSB_CtrlSetupSetInterface(void * pVoid)

Description

Setup ACK handler for Set interface command.

Parameter

pVoid	Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, SET_INTERFACE, DrvUSB_CtrlSetupSetInterface, 0, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupSetConfiguration

Prototype

void DrvUSB_CtrlSetupSetConfiguration(void * pVoid)

Description

Setup ACK handler for Set configuration command.

Parameter

pVoid	Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
   {REQ_STANDARD, SET_CONFIGURATION, DrvUSB_CtrlSetupSetConfiguration, 0, 0,
&g_HID_sDevice}
};
```

DrvUSB_CtrlDataInSetAddress

Prototype

void DrvUSB_CtrlDataInSetAddress(void * pVoid)

Description

Setup ACK handler for Set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
   {REQ_STANDARD, SET_ADDRESS, DrvUSB_CtrlSetupSetAddress,
DrvUSB_CtrlDataInSetAddress, 0, &g_HID_sDevice}
};
```

DrvUSB_memcpy

Prototype

void DrvUSB_memcpy(uint8_t *pi8Dest, uint8_t *pi8Src, uint32_t u32Size)

Description

The USB buffer is recommended to be byte access thus this function is implemented by byte access.

Parameter

pi8Dest: Destination pointer

pi8Src: Source pointer

u32Size: Data size. The unit is byte.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Copy 64 bytes data from USB SRAM to SRAM */
DrvUSB_memcpy(0x20000800, 0x40060100, 64);
```

13. Appendix

13.1. NuMicro™ NUC122 Series Products Selection Guide

Part number	Flash (KB)	ISP ROM (KB)	SRAM (KB)	I/O	Timer	Connectivity						I^2S	Comp.	PWM	ADC	RTC	ISP ICP	Package
						UART	SPI	I^2C	USB	LIN	PS2							
NUC122ZD2AN	64 KB	4KB	8 KB	up to 18	4x32-bit	1	2	1	1	1	-	-	-	-	-	-	v	QFN33
NUC122ZC1AN	32 KB	4KB	4 KB	up to 18	4x32-bit	1	2	1	1	1	-	-	-	-	-	-	v	QFN33
NUC122LD2AN	64 KB	4KB	8 KB	up to 30	4x32-bit	2	2	1	1	2	1	-	-	4	-	v	v	LQFP48
NUC122LC1AN	32 KB	4KB	4 KB	up to 30	4x32-bit	2	2	1	1	2	1	-	-	4	-	v	v	LQFP48
NUC122RD2AN	64 KB	4KB	8 KB	up to 41	4x32-bit	2	2	1	1	2	1	-	-	4	-	v	v	LQFP64
NUC122RC1AN	32 KB	4KB	4 KB	up to 41	4x32-bit	2	2	1	1	2	1	-	-	4	-	v	v	LQFP64

13.2. PDID Table

Part number	PDID
NUC122ZD2AN	0x00012231
NUC122ZC1AN	0x00012235
NUC122LD2AN	0x00012204
NUC122LC1AN	0x00012208
NUC122RD2AN	0x00012213
NUC122RC1AN	0x00012217

14. Revision History

Version	Date	Description
V1.00.001	May. 5, 2011	● Created
V1.00.002	July. 8, 2011	● Add new DrvGPIO functions

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.