Nuvoton 501 Series Driver Reference Guide

V1.01.001

Publication Release Date: Nov. 2009

Support Chips:

Nuvoton 501 Series

Support Platforms:

ADS 1.2

RVDS 3.0

Keil RealView MDK 3.6

IAR EWARM 5.2

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. DrvADC Introduction

1.1. ADC Introduction

The 10-bit Analog to Digital Converter (ADC) is a successive approximation type ADC with 8-channel inputs. It takes 34 cycles to convert one sample, and the maximum input clock to ADC is 25MHz. The ADC can operate in two modes, one is normal ADC mode and the other one is audio recording mode. The two modes can't work at the same time.

Beside the 10-bit ADC, an 8-levels voltage detector is included in all chips. The detecting result is independent of power supply, and it can give the system a warning signal when battery voltage is lower than an absolute reference voltage.

1.2. ADC Feature

The Analog to Digital Converter includes following features:

- Maximum conversion rate: 300K sample per second
- Power supply voltage: 3.3V
- Analog input voltage range: 0 ~ 3.3 volts
- 8-level voltage detector
- The APIs includes setting conditions and getting data for ADC application about Voltage Detector

2. DrvADC APIs Specification

2.1. Macros

_DRVADC_CONV

Prototype

VOID _DRVADC_CONV (VOID);

Description

Inform ADC to start converting the input voltage to digital value.

Include

Driver/DrvADC.h

_DRVADC_CHECK_CONV

Prototype

VOID _DRVADC_CHECK_CONV (VOID);

Description

Check if the conversion is finished. After this API return, the conversion is finished and the status flag is cleared (status flag will be set if the conversion is finished and should be cleared).

Include

Driver/DrvADC.h

_DRVADC_GET_ADC_DATA

Prototype

UINT32

_DRVADC_GET_ADC_DATA (VOID);

Description

After conversion is done, normal ADC channel conversion value can be obtained by this API.

Include

Driver/DrvADC.h

Return Value

10-bit ADC value

_DRVADC_RECORD_MODE_SELECT

Prototype

UINT32

_DRVADC_RECORD_MODE_SELECT(u8RecordMode);

Description

Select the record mode between 0 ~ 3.

Parameters

u8RecordMode [in]

0: One audio sample generates one AUDIO interrupt flag.

1: Two audio samples generate one AUDIO interrupt flag.

2: Four audio samples generate one AUDIO interrupt flag.

3: Eight audio samples generate one AUDIO interrupt flag.

Include

Driver/DrvADC.h

_DRVADC_GETAUDIOINTFLAG

Prototype

BOOL

_DRVADC_GETAUDIOINTFLAG(VOID);

Description

Get audio interrupt flag.

Include

Driver/DrvADC.h

Return Value

Audio interrupt flag status

_ DRVADC_CLEARAUDIOINTFLAG

Prototype

VOID

_ DRVADC_CLEARAUDIOINTFLAG (VOID);

Description

Clear audio interrupt flag.

Include

Driver/DrvADC.h

Return Value

None

2.2. Functions

DrvADC_IsLowVoltage

Prototype

BOOL

DrvAdc_IsLowVoltage (VOID);

Description

Check if the voltage is lower than the reference voltage.

Include

Driver/DrvADC.h

Return Value

TRUE: Low voltage.

FALSE: Voltage is not lower than the reference value.

DrvADC_GetRecordData

Prototype

PINT16 DrvADC_GetRecordData (

VOID

);

Description

Get the converted ADC value in ADC_RECORD mode.

Parameters

None

Include

Driver/DrvADC.h

Return Value

The pointer to the record data. The data length is 8 samples.

DrvADC_EnableInt

Prototype

```
VOID DrvADC_EnableInt (
        DRVADC_ADC_CALLBACK callback,
        UINT32 u32UserData
);
```

Description

Enable ADC interrupt and setup callback function.

Parameters

callback [in]

The callback funciton.

u32UserData [in]

The user's data to pass to the callback function.

Include

Driver/DrvADC.h

DrvADC_DisableInt

Prototype

```
VOID DrvAdc_DisableInt (VOID);
```

Description

disable ADC interupt.

Parameters

None

Include

Driver/DrvADC.h

DrvADC_Open

Prototype

```
ERRCODE DrvADC_Open(ADC_MODE mode, UINT32 u32ConvClock);
```

Description

Open the ADC conversion or Audio record function.

Parameters

mode [in]

The work mode of ADC. It could be in normal ADC conversion mode or audio recording mode

u32ConvClock [in]

If working in ADC_NORMAL mode, u32ConvClock is the conversion rate. If working in ADC_RECORD mode, u32ConvClock is the sampling rate.

Include

Driver/DrvADC.h

Return Value

E_SUCCESS	Success
E_DRVADC_ARGUMENT	Wrong argument
E_DRVADC_CLOCK	Unable to output a suitable clock

DrvADC_Close

Prototype

VOID DrvAdc_Close (VOID);

Description

Close ADC functions.

Include

Driver/DrvADC.h

DrvADC_StartRecord

Prototype

VOID DrvAdc_StartRecord (VOID);

Description

Start to record Audio data. This function only can be used in ADC_RECORD mode.

Include

Driver/DrvADC.h

DrvADC_EnableLvd

Prototype

VOID DrvADC_EnableLvd(UINT32 u32Level);

Description

Enable low voltage detection function.

Parameters

u32Level [in]

et trigger voltage. it could be 0~7 and the relative voltages are as following table:

The selector vs. detected voltage level

u32Level	Voltage Level (V)
0	2.0
1	2.1
2	2.2
3	2.3
4	2.4
5	2.5
6	2.6
7	2.7

Include

Driver/DrvADC.h

DrvADC_ DisableLvd

Prototype

VOID DrvAdc_ DisableLvd (VOID);

Description

Disable the low voltage detection function.

Include

Driver/DrvADC.h

DrvADC_StopRecord

Prototype

VOID DrvADC_StopRecord(VOID);

Description

Stop recording Audio data. This function only can be used in ADC_RECORD mode.

Include

Driver/DrvADC.h

DrvADC_StartConvert

Prototype

VOID DrvADC_StartConvert(UINT32 u32Channel);

Description

Start to convert ADC data. This function only can be used in ADC_NORMAL mode.

Parameters

u32Channel [in]

The analog input channel and it could be ch2~ch7.

Include

Driver/DrvADC.h

Return Value

E_SUCCESS	Success

DrvADC_IsConvertReady

Prototype

BOOL DrvADC_IsConvertReady(VOID);

Description

Check if ADC (not audio) is converted OK

Include

Driver/DrvADC.h

Return Value

TURE	Conversion finished
FALSE	Under converting

DrvADC_GetConvertClock

Prototype

UINT32 DrvADC_GetConvertClock(VOID);

Description

To get the conversion clock or recording sampling rate.

Include

Driver/DrvADC.h

Return Value

Return the conversion clock if it is in ADC_NORMAL mode.

Return the recording sampling rate if it is in ADC_RECORD mode.

DrvADC_EnableLvdInt

Prototype

```
VOID DrvADC_EnableLvdInt(
        DRVADC_LVD_CALLBACK callback,
        UINT32 u32UserData
);
```

Description

Enable LVD interrupt and setup callback function.

Parameters

callback [in]

The callback function.

u32UserData [in]

The user's data to pass to the callback function.

Include

Driver/DrvADC.h

DrvADC_DisableLvdInt

Prototype

```
VOID DrvADC_DisableLvdInt(VOID);
```

Description

Disable low voltage detection interrupt.

Include

Driver/DrvADC.h

DrvADC_IsLowVoltage

Prototype

```
BOOL DrvADC_IsLowVoltage(VOID);
```

Description

Get the low voltage status.

Include

Driver/DrvADC.h

Return Value

TURE	low voltage detected
FALSE	No low voltage detected

DrvADC_GetRecordReadyFlag

Prototype

BOOL DrvADC_GetRecordReadyFlag(VOID);

Description

Check if the recording data is converted OK.

Include

Driver/DrvADC.h

Return Value

TURE	data is ready
FALSE	data is not ready

DrvADC_ClearRecordReadyFlag

Prototype

VOID DrvADC_ClearRecordReadyFlag(VOID);

Description

To clear the recording ready flag.

Include

Driver/DrvADC.h

DrvADC_SetMICGain

Prototype

VOID DrvADC_SetMICGain(UINT16 u16MicGainLevel);

Description

Set record volume gain.

Parameters

u16MicGainLevel [in]

The volume gain could be 0 ~ 31 dB.

Include

Driver/DrvADC.h

DrvADC_GetMICGain

Prototype

INT16 DrvADC_GetMICGain(VOID);

Description

Get record volume gain.

Include

Driver/DrvADC.h

Return Value

Return the recording gain in dB.

DrvADC_GetVersion

Prototype

UINT32

DrvAdc_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvADC.h

Return Value

Version number:

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

3. DrvAIC Introduction

3.1. AIC Introduction

An interrupt temporarily changes the sequence of program execution to react to a particular event such as power failure, watchdog timer timeout, transmit/receive request from Serial Interface (UART or SPI) Controller, and so on. The ARM7TDMI processor provides two modes of interrupt, Fast Interrupt (FIQ) mode for critical session and Interrupt (IRQ) mode for general purpose. IRQ exception is occurred when nIRQ input is asserted. Similarly, FIQ exception is occurred when nFIQ input is asserted. FIQ has privilege over IRQ and can preempt an on-going IRQ.

Advanced Interrupt Controller (AIC) is capable of dealing with the interrupt requests from a total of 32 different sources. Currently, 31 interrupt sources are defined. Each interrupt source is uniquely assigned to an interrupt channel. For example, the watchdog timer interrupt is assigned to channel 1. The AIC implements a proprietary eight-level priority scheme that differentiates the available 31 interrupt sources into eight priority levels. Interrupt sources within the priority level 0 have the highest priority, and the priority level 7 has the lowest. To work this scheme properly, must specify a certain priority level to each interrupt source during power-on initialization; otherwise, the system shall behave unexpectedly. Within each priority level, interrupt source that is positioned in a lower channel has a higher priority. Interrupt source that is active, enabled, and positioned in the lowest channel within the priority level 0 is promoted to FIQ. Interrupt sources within the priority levels excepting 0 can petition for the IRQ. IRQ can be preempted by the occurrence of the FIQ. Interrupt nesting is performed automatically by AIC.

Though interrupt sources are intrinsically high-level sensitive, AIC can be configured as either low-level sensitive, high-level sensitive, negative-edge triggered, or positive-edge triggered to each interrupt source.

3.2. AIC Feature

The Advanced Interrupt Controller includes following features:

- AMBA APB bus interface
- External interrupts can be programmed as either edge-triggered or level-sensitive
- External interrupts can be programmed as either low-active or high-active
- Flags to reflect the status of each interrupt source
- Individual mask for each interrupt source
- Proprietary 8-level interrupt scheme to ease the burden from the interrupt
- Priority methodology is adopted to allow for interrupt daisy-chaining
- Automatically masking out the lower priority interrupt during interrupt nesting
- Automatically clearing the interrupt flag when the external interrupt source is programmed to be edge-triggered

4. DrvAIC APIs Specification

4.1. Macros

The following macro is usually used to test the sample code and verification the hardware whether is work normal or not.

_DRVAIC_TEST_MODE

Prototype

VOID DRVAIC_TEST_MODE (BOOL bIsTestMode);

Description

Force AIC to test mode or normal mode.

Parameter

bIsTestMode [in]

TRUE: Force AIC to enter test mode

FALSE: Force AIC to normal mode

Include

Driver/DrvAIC.h

_DRVAIC_CLEAR_INT

Prototype

```
VOID _DRVAIC_CLEAR_INT (
  UINT32     u32IntChannels
);
```

Description

Clear the interrupt status of AIC.

Parameter

u32IntChannels [in]

Interrupts channel name. To get the available interrupt sources, refer to Interrupt Sources.

Include

Driver/DrvAIC.h

_DRVAIC_ENABLE_INT

Prototype

```
VOID _DRVAIC_ENABLE_INT (
  UINT32     u32IntChannels
);
```

Description

Enable the selected interrupt channels.

Parameter

u32IntChannels [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

Include

Driver/DrvAIC.h

_DRVAIC_DISABLE_INT

Prototype

```
VOID _DRVAIC_DISABLE_INT (
  UINT32 u32IntChannels
);
```

Description

Disable the corresponding interrupt channels.

Parameter

u32IntChannels [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

Include

Driver/DrvAIC.h

_DRVAIC_TRIGGER_INT

Prototype

```
VOID _DRVAIC_TRIGGER_INT (
  UINT32 u32IntChannels
```

);

Description

Trigger the specified interrupt channels.

Parameter

u32IntChannels [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

Include

Driver/DrvAIC.h

4.2. Functions

DrvAIC_SetGlobalInt

Prototype

VOID DrvAIC_SetGlobalInt (

INT32 intState

);

Description

Set the interrupt mask bit.

Parameters

intState [in]

Value	Means
0 DRVAIC_ENABLE_ALL_INTERRUPTS	Enable all interrupt channels
1 DRVAIC_DISABLE_ALL_INTERRUPTS	Disable all interrupt channels

Include

Driver/DrvAIC.h

DrvAIC_EnableInt

Prototype

INT32

```
DrvAIC_EnableInt(
    AIC_INT_SOURCE unIntNo
);
```

Description

Enable signal interrupt channel.

Parameters

unIntNo [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

Include

Driver/DrvAIC.h

Return Value

E_SUCCESS: The channel is enabled.

E_DRVAIC_INVALID_CHANNEL: The specified interrupt is invalid.

DrvAIC_DisableInt

Prototype

```
INT32
DrvAIC_DisableInt(
    AIC_INT_SOURCE unIntNo
);
```

Description

Disable signal interrupt channel.

Parameters

unIntNo [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

Include

Driver/DrvAIC.h

Return Value

E_SUCCESS: The channel is disabled.

E_DRVAIC_INVALID_CHANNEL: The specified interrupt is invalid.

DrvAIC_GetIntEnableStatus

Prototype

UINT32

DrvAIC_GetIntEnableStatus (

VOID

);

Description

Get how many interrupt channels were enabled.

Parameters

None

Include

Driver/DrvAIC.h

Return Value

The set of the enabled interrupt channels. To get the available interrupt sources, refer to Interrupt Sources.

DrvAIC_InstallFiqHandler

Prototype

PVOID DrvAIC_InstallFiqHandler (

PVOID pfnNewISR

);

Description

The function is used to install FIQ handler to interrupt vector-0x3C.

Parameters

A function pointer points to new ISR.

Include

Driver/DrvAIC.h

Return Value

The old function pointer points to old ISR. The upper layer can use the old function pointer to implement interrupt cascade.

DrvAIC_InstallIrqHandler

Prototype

PVOID DrvAIC_InstallIrqHandler (

PVOID pfnNewISR

);

Description

The function is used to install IRQ handler to interrupt vector-0x38.

Parameters

A function pointer points to new ISR.

Include

Driver/DrvAIC.h

Return Value

The old function pointer points to old ISR. The upper layer can use the old function pointer to implement interrupt cascade.

DrvAIC_InstallISR

Prototype

```
PVOID DrvAIC_InstallISR(
    AIC_INT_LEVEL eintTypeLevel
    AIC_INT_SOURCE eintNo,
    PVOID pfnNewISR,
    UINT32 u32UserData
);
```

Description

Use this function to set up interrupt channel (eIntNo) with a callback function (pfnNewISR) to AIC interrupt vector table. There are total 7 levels for each interrupt channel. The highest priority is 0, and the lowest priority is 7.

Parameters

eintTypeLevel [in]

Interrupt level. To get the available interrupt levels, refer to Interrupt Levels

eintNo [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

pfnNewISR [in]

Callback function that points to interrupt handler bases on the channel name.

u32UserData [in]

A user defined parameter that is always pass to upper level by the callback function.

Include

Driver/DrvAIC.h

Return Value

The old function pointer points to old callback function bases on the specified interrupt channel name. The upper layer can use the old function pointer to implement interrupt cascade.

DrvAIC_SetIntType

Prototype

```
INT32 DrvAIC_SetIntType (
    AIC_INT_SOURCE eintNo,
    AIC_INT_TYPE eintSourceType
);
```

Description

The function is used to set the interrupt trigger type.

Parameters

eintNo [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

eintSourceType [in]

Interrupt trigger type bases on the interrupt channel name. To get the available interrupt trigger type, refer the Interrupt types.

Include

Driver/DrvAIC.h

Return Value

E_SUCCESS: Successful.

E_DRVAIC_INVALID_CHANNEL: The specified interrupt is invalid.

DrvAIC_SetIntPriorityLevel

Prototype

```
INT32 DrvAIC_SetIntPriorityLevel (
    AIC_INT_SOURCE eintNo,
    AIC_INT_LEVEL eintLevel
);
```

Description

The function is used to set the interrupt priority level.

Parameters

eintNo [in]

Interrupt channel name. To get the available interrupt sources, refer to Interrupt Sources.

eintLevel [in]

Interrupt levels bases on the interrupt channel name. To get the available interrupt trigger type, refer the Interrupt Levels

Include

Driver/DrvAIC.h

Return Value

E_SUCCESS: Successful.

E_DRVAIC_INVALID_CHANNEL: The specified interrupt is invalid.

DrvAIC_SetCPSR

Prototype

```
UINT32 DrvAIC_SetCPSR (
    INT_CPSR_SETTING eSetting,
);
```

Description

The function is used to enable or disable I bit or F bit.

Parameters

eSetting [in]

Enable or disable I bit or F bit. To get the available interrupt trigger type, refer the Constant definitions for CPSR

Include

Driver/DrvAIC.h

Return Value

E_SUCCESS: Successful.

E_DRVAIC_INVALID_CPSR_SETTING: The specified setting is invalid.

DrvAIC_InstallExceptionHandler

Prototype

```
PVOID DrvAIC_InstallExceptionHandler (
    EXCEPTION_TYPE eExceptionType,
    PVOID pNewHandler
);
```

Description

The function is used to install the exception handler.

Parameters

eExceptionType [in]

Exception type references the Constant definitions for Exceptions

pfnNewISR [in]

Function pointer that points to exception handler bases on the exception type.

Include

Driver/DrvAIC.h

Return Value

The old function pointer points to old function pointer bases on the specified exception type. The upper layer can use the old function pointer to implement interrupt cascade.

DrvAIC_EndISR

Prototype

VOID DrvAIC_EndISR(AIC_INT_SOURCE intNo);

Description

This function is used to end the ISR for each interrupt channel.

Parameters

intNo [in]

Interrupt channel number.

Include

Driver/DrvAIC.h

Return Value

None

DrvAIC_GetVersion

Prototype

UINT32

DrvAIC_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvAIC.h

Return Value

Version number :

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

4.3. Interrupt Sources

Priority	Channel Name	Source
1	INT_WDT	Watchdog Timer Interrupt
3	INT_GPIO0	External Interrupt 0
4	INT_GPIO1	External Interrupt 1
5	INT_GPIO2	External Interrupt 2
6	INT_GPIO3	External Interrupt 3
7	INT_APU	Audio Processing Unit Interrupt
10	INT_ADC	AD Converter Interrupt
11	INT_RTC	RTC Timer Interrupt
12	INT_UART0	UART0 Interrupt
13	INT_UART1	UART1 Interrupt
14	INT_TM1	Timer1 Interrupt
15	INT_TM0	Timer0 Interrupt
19	INT_USB	USB Device Interface Controller Interrupt
22	INT_PWM0	PWM0 Timer 0 Interrupt
23	INT_PWM1	PWM1 Timer 1 Interrupt
24	INT_PWM2	PWM2 Timer 2 Interrupt
25	INT_PWM3	PWM3 Timer 3 Interrupt
26	INT_I2C	I2C Interrupt
27	INT_SPIMS	SPI Master/Slave Interrupt
29	INT_PWR	System Wake-Up Interrupt
30	INT_SPIM	SPI ROM Interrupt

4.4. Interrupt Levels

Priority	Level
0 (Highest)	AIC_INT_LEVEL0
1	AIC_INT_LEVEL1
2	AIC_INT_LEVEL2
3	AIC_INT_LEVEL3
4	AIC_INT_LEVEL4
5	AIC_INT_LEVEL5
6	AIC_INT_LEVEL6
7	AIC_INT_LEVEL7

4.5. Interrupt types

Type	Meaning
AIC_LOW_LEVEL	Low level trigger interrupt
AIC_HIGH_LEVEL	High level trigger interrupt
AIC_FALLING	Falling edge trigger interrupt
AIC_RISING	Rising edge trigger interrupt

4.6. Constant definitions for CPSR

Type	Value	Meaning
AIC_ENABLE_IRQ	0x7F	Enable IRQ interrupt.
AIC_ENABLE_FIQ	0x8F	Enable FIQ interrupt.
AIC_ENABLE_FIQ_IRQ	0x3F	Enable IRQ and FIQ interrupts.
AIC_DISABLE_IRQ	0x80	Disable IRQ interrupt.
AIC_DISABLE_FIQ	0x40	Disable FIQ interrupt.
AIC_DISABLE_FIQ_IRQ	0xC0	Disable IRQ and FIQ interrupt.

4.7. Constant definitions for Exceptions

Type	Value	Meaning
AIC_SWI	0x0	Software interrupt exception
AIC_D_ABORT	0x1	Data abort exception
AIC_I_ABORT	0x2	Instruction abort exception
AIC_UNDEFINE	0x3	Undefined exception

5. DrvAPU Introduction

5.1. APU Introduction

The main purpose of Audio Processing Unit (APU) is used to playback the audio data (PCM format) which CPU decoded and stored in SRAM. The APU only provides a mono DAC with 16-bit resolution channel. The APU is composed of an AHB Master and built in FIFO and timer. Details please refer to the section in the target chip specification titled Audio Processing Unit.

5.2. APU Feature

The Audio Processing Unit includes following features:

- Built in a monophonic DAC with 16-bit resolution per channel.
- AHB Master with DMA.
- Built in FIFO.

6. DrvAPU APIs Specification

6.1. Macros

_DRVAPU_SET_SAMPLE_RATE

Prototype

```
VOID    _DRVAPU_SET_SAMPLE_RATE (
   UINT32      u32SampleRate
);
```

Description

Specify the sample rate of APU.

Parameter

u32SampleRate [in]

Specify the sample rate of audio file.

Include

Driver/DrvAPU.h

Return Value

None

Notes

The sample rate will generate a clock divider to generate a frequency-divided clock output for Audio DAC.

The generated divider may not be 100% accuracy because of integer calculation.

_DRVAPU_GET_SAMPLE_RATE

Prototype

```
UINT32
_DRVAPU_GET_SAMPLE_RATE (VOID);
```

Description

Get the sample rate of APU.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The sample rate value of APU

Notes

The sample rate will generate according to clock divider for Audio DAC.

The generated sample rate may not be 100% accuracy because of integer calculation.

_DRVAPU_ENABLE

Prototype

VOID _DRVAPU_ENABLE (VOID);

Description

Enable APU to start play.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

None

Notes

It should be called after all APU setting is done.

_DRVAPU_DISABLE

Prototype

VOID _DRVAPU_DISABLE (VOID);

Description

Disable APU to stop play.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

None

_DRVAPU_ISENABLED

Prototype

BOOL

_DRVAPU_ISENABLED (VOID);

Description

Check if APU is enabled.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

TRUE : Enabled

FALSE : Disabled

_DRVAPU_RESET

Prototype

VOID _DRVAPU_RESET (VOID);

Description

Reset the whole APU except for register value.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

None

_DRVAPU_SET_BASE_ADDRESS

Prototype

```
VOID    _DRVAPU_SET_BASE_ADDRESS (
    UINT32      u32BaseAddress
);
```

Description

Set the base address of playback buffer.

Parameter

u32BaseAddress [in]

Specify the base byte address

Include

Driver/DrvAPU.h

Return Value

None

_DRVAPU_GET_BASE_ADDRESS

Prototype

```
UINT32
_DRVAPU_GET_BASE_ADDRESS (VOID);
```

Description

Get the base address of playback buffer.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The base address of playback buffer.

_DRVAPU_SET_TH1_ADDRESS

Prototype

```
VOID    _DRVAPU_SET_TH1_ADDRESS (
    UINT32      u32TH1Address
);
```

Description

Set the threshold 1 address of playback buffer.

Parameter

u32TH1Address [in]

Specify threshold 1 byte address

Include

Driver/DrvAPU.h

Return Value

None

Notes

1. APU TH1 interrupt occurs when APU internal counter matches with threshold 1 address and interrupt is enabled. Data at TH1 address is not processed by APU at the instance when interrupt occurs.
2. The size of TH1 must be multiple of 4.

_DRVAPU_GET_TH1_ADDRESS

Prototype

UINT32

_DRVAPU_GET_TH1_ADDRESS (VOID);

Description

Get the threshold 1 address of playback buffer.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The threshold 1 address of playback buffer.

_DRVAPU_SET_TH2_ADDRESS

Prototype

```
VOID    _DRVAPU_SET_TH2_ADDRESS (
    UINT32      u32TH2Address
);
```

Description

Set the threshold 2 address of playback buffer.

Parameter

u32TH2Address [in]

Specify threshold 2 byte address

Include

Driver/DrvAPU.h

Return Value

None

Notes

1. APU TH2 interrupt occurs when APU internal counter matches with threshold 2 address and interrupt is enabled. Data at TH2 address is not processed by APU at the instance when interrupt occurs. Instead, APU processes data at base address later.
2. The size of TH2 must be multiple of 4.

_DRVAPU_GET_TH2_ADDRESS

Prototype

UINT32

_DRVAPU_GET_TH2_ADDRESS (VOID);

Description

Get the threshold 2 address of playback buffer.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The threshold 2 address of playback buffer.

_DRVAPU_GET_CUR_ADDRESS

Prototype

UINT32

_DRVAPU_GET_CUR_ADDRESS (VOID);

Description

Return the current play address of playback buffer.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The current address of playback buffer.

_DRVAPU_SET_POWER_DOWN

Prototype

```
VOID    _DRVAPU_SET_POWER_DOWN (
   UINT16     u16PowerDown
);
```

Description

Set APU to the power down mode.

Parameter

u16PowerDown [in]

0 (normal mode) ~ 0xFF (power down mode)

Include

Driver/DrvAPU.h

Return Value

None

Notes

To shut down APU and AUDIO DAC completely, _DRVAPU_DISABLE () must be called in addition to _DRVAPU_SET_POWER_DOWN (0xFF). Values must be increased smoothly to avoid abrupt noise during power down.

_DRVAPU_GET_POWER_DOWN

Prototype

```
UINT16
_DRVAPU_GET_POWER_DOWN (VOID);
```

Description

Get the setting of power down mode.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The power down value.

6.2. Functions

DrvAPU_Open

Prototype

```
VOID DrvAPU_Open(
        UINT32 u32BufAddr,
        UINT32 u32Samples,
        UINT32 u32SampleRate
  );
```

Description

To initial the APU DMA buffer and sampling rate. NOTE: Due to hardware limitation, the sampling rate may be differnt to target sampling rate. The actual sample rate could be get by calling DrvAPU_GetSampleClock().

Parameter

u32BufAddr [in]

The APU DMA buffer.

u32Samples [in]

The total samples could be stored in the DMA buffer.

u32SampleRate [in]

The sampling rate.

Include

Driver/DrvAPU.h

Return Value

None

DrvAPU_Close

Prototype

VOID DrvApu_Close (VOID);

Description

Turn off the APU functionality.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

None

DrvAPU_EnableInt

Prototype

VOID DrvAPU_EnableInt(APU_CALLBACK fnPlayCallBack, UINT32 u32UserData);

Description

To enable the APU playback interrupt and install the callback function.

Parameter

fnPlayCallBack [in]

The APU playback interrupt callback function.

u32UserData [in]

The user's data.

Include

Driver/DrvAPU.h

Return Value

None

Note

Use DRVAPU_TH1 | DRVAPU_TH2 to enable multiple interrupts simultaneously.

DrvAPU_DisableInt

Prototype

VOID DrvAPU_DisableInt(VOID);

Description

To disable the APU playback interrupt.

Include

Driver/DrvAPU.h

Return Value

None

DrvAPU_ClearInt

Prototype

```
VOID    DrvAPU_ClearInt (
    UINT32      u32InterruptFlag
);
```

Description

Clear the specified APU interrupt source.

Parameter

u32InterruptFlag [in]

DRVAPU_TH1_INT	Threshold 1 interrupt status.
DRVAPU_TH2_INT	Threshold 2 interrupt status.

Include

Driver/DrvAPU.h

Return Value

None

Note

Use DRVAPU_TH1_INT | DRVAPU_TH2_INT to clear multiple interrupts simultaneously.

DrvAPU_GetInt

Prototype

```
UINT32
DrvAPU_GetInt (
    UINT32      u32InterruptFlag
);
```

Description

Return the status of specified APU interrupt source to acknowledge the play has reached the threshold.

Parameter

u32InterruptFlag [in]

DRVAPU_TH1_INT	Threshold 1 interrupt status.
DRVAPU_TH2_INT	Threshold 2 interrupt status.

Include

Driver/DrvAPU.h

Return Value

DRVAPU_TH1_INT	Threshold 1 interrupt occurs
DRVAPU_TH2_INT	Threshold 2 interrupt occurs
DRVAPU_TH1_INT| DRVAPU_TH2_INT	Threshold 1 & Threshold2 interrupt occurs

DrvAPU_InitFIFO

Prototype

VOID DrvAPU_InitFIFO(INT16 *pi16Buf, UINT32 u32Samples);

Description

This function is used to initial the FIFO buffer and relative variables. It should be called before using any other FIFO relative functions.

Parameter

pi16Buf [in]

The FIFO buffer.

u32Samples [in]

Total samples that the FIFO buffer could supports.

Include

Driver/DrvAPU.h

Return Value

None

DrvAPU_PushFIFO

Prototype

VOID DrvAPU_PushFIFO(INT16 * pi16Buf, UINT32 u32Samples);

Description

This function is used to push the audio data into FIFO. The DrvAPU_InitFIFO() must be called before using this function.

Parameter

pi16Buf [in]

The buffer address of audio data in 16-bit PCM format.

u32Samples [in]

The samples in the audio data buffer.

Include

Driver/DrvAPU.h

Return Value

The samples got from FIFO.

DrvAPU_PopFIFO

Prototype

UINT32 DrvAPU_PopFIFO(INT16 * pi16Buf, UINT32 u32Samples);

Description

This function is used to pop the audio data from FIFO. The DrvAPU_InitFIFO() must be called before using this function.

Parameter

pi16Buf [in]

The buffer address of audio data to store the data from FIFO.

u32Samples [in]

The samples of the audio data to get from FIFO.

Include

Driver/DrvAPU.h

DrvAPU_GetSpaceInFIFO

Prototype

UINT32 DrvAPU_GetSpaceInFIFO(VOID);

Description

To get the free space in the FIFO. The unit is one sample (16-bit).

Include

Driver/DrvAPU.h

Return Value

The free space in the FIFO. This unit is one sample (16-bit).

DrvAPU_GetSamplesInFIFO

Prototype

UINT32 DrvAPU_GetSamplesInFIFO(VOID);

Description

To get the samples in the FIFO.

Include

Driver/DrvAPU.h

Return Value

The samples in the FIFO.

DrvAPU_SetSampleClock

Prototype

ERRCODE DrvAPU_SetSampleClock(UINT32 u32SampleRate);

Description

To set the APU playback sample rate.

Parameter

u32SampleRate The playback sampler rate in Hz.

Include

Driver/DrvAPU.h

Return Value

E_SUCCESS

DrvAPU_GetSampleClock

Prototype

UINT32 DrvAPU_GetSampleClock(VOID);

Description

To calculate the actual sampling rate according to the system and APU clock settings.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

The playback sampling rate of APU.

DrvAPU_StartPlay

Prototype

VOID DrvAPU_StartPlay(VOID);

Description

The APU could be triggred to start play after open and enable interrupt.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

None

DrvAPU_StopPlay

Prototype

VOID DrvAPU_StopPlay(VOID);

Description

Stop to play audio by APU.

Parameter

None

Include

Driver/DrvAPU.h

Return Value

None

DrvAPU_GetVersion

Prototype

UINT32

DrvAPU_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvAPU.h

Return Value

Version number :

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

7. DrvGPIO Introduction

7.1. GPIO introduction

- 37 pins of General Purpose I/O are shared with special feature functions.
- Supported Features of these I/O are: input or output facilities, pull-up resistors.
- All these general purpose I/O functions are achieved by software programming setting and I/O cells selected from universal standard I/O Cell Library.

8. DrvGPIO APIs Specification

8.1. Functions

DrvGPIO_Init

Prototype

ERRCODE DrvGPIO_Init(VOID);

Description

To initialize the GPIO.

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success

DrvGPIO_Open

Prototype

```
ERRCODE DrvGPIO_Open(
    DRVGPIO_PORT      port,
    INT32             i32Bit,
    DRVGPIO_IO        mode,
    DRVGPIO_PULL_UP   pullUp,
    DRVGPIO_DRIVE     drive
);
```

Description

To configure the specified GPIO to use it.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

mode [in]

Set the IO to be IO_INPUT or IO_OUTPUT.

pullUp [in]

Set the IO to be IO_PULL_UP or IO_NO_PULL_UP.

i32Bit [in]

Set the IO to be IO_LOWDRV or IO_HIGHDRV to set low or high current driving.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.
E_DRVGPIO_BUSY	The IO has been used.

DrvGPIO_Close

Prototype

ERRCODE DrvGPIO_Close(DRVGPIO_PORT port, INT32 i32Bit);

Description

To close the opened IO and reset its configurations.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_SetBit

Prototype

ERRCODE DrvGPIO_SetBit(DRVGPIO_PORT port,INT32 i32Bit);

Description

Set the specified IO bit to 1.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_ClrBit

Prototype

ERRCODE DrvGPIO_ClrBit(DRVGPIO_PORT port,INT32 i32Bit);

Description

Clear the specified IO bit to 0.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_GetBit

Prototype

INT32 DrvGPIO_GetBit(DRVGPIO_PORT port, INT32 i32Bit);

Description

Get the value of the specified IO bit.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified IO bit.

DrvGPIO_SetPortBits

Prototype

ERRCODE DrvGPIO_SetPortBits(DRVGPIO_PORT port,INT32 i32Data);

Description

Set the specified IO port.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Data [in]

The data to write to the specified IO port.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_GetPortBits

Prototype

INT32 DrvGPIO_GetPortBits(DRVGPIO_PORT port);

Description

Get the data of the specified IO port.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

Include

Driver/DrvGPIO.h

Return Value

The value of the IO port.

DrvGPIO_GetPortDoutBits

Prototype

INT32 DrvGPIO_GetPortDoutBits(DRVGPIO_PORT port);

Description

Get the Dout register value of the specified IO port.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

Include

Driver/DrvGPIO.h

Return Value

The value of the GPIO DOUT register value.

DrvGPIO_EnableInt

Prototype

```
ERRCODE
DrvGPIO_EnableInt(
    DRVGPIO_PORT        port,
    INT32               i32Bit,
```

```
    DRVGPIO_INT_NUM     intNum,
    DRVGPIO_INT_TYPE    triggerType,
    DRVGPIO_CALLBACK    callback,
    UINT32              u32UserData
);
```

Description

Enable the interrupt function of the specified IO bit and install relative callback function.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

intNum [in]

Select the interrupt to be enabled. It could be IO_INT0, IO_INT1, IO_INT2 or IO_INT3.

triggerType [in]

Set the interrupt trigger type. It could be IO_RISING, IO_FALLING or IO_BOTH_EDGE.

callback [in]

Set the callback function.

u32UserData [in]

Set the data to pass to the callback function.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_DisableInt

Prototype

ERRCODE DrvGPIO_DisableInt(DRVGPIO_PORT port,INT32 i32Bit);

Description

Enable the interrupt function of the specified IO bit and install relative callback function.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

i32Bit [in]

Specified bit of the IO port. It could be 0~15 for GPIOA, 0~9 for GPIOB, and 0~10 for GPIOC.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_SetDebounceTime

Prototype

ERRCODE DrvGPIO_SetDebounceTime(INT32 i32DebounceClk);

Description

To set the debounce timing.

Parameter

i32DebounceClk [in]

The debounce timing is 2^(i32DebounceClk) * APB clock.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_EnableIntDebounce

Prototype

```
ERRCODE DrvGPIO_EnableIntDebounce(
    DRVGPIO_INT_NUM     intNum,
    INT32               i32DebounceClk
);
```

Description

To enable the debounce function of the specified GPIO interrrupt.

Parameter

intNum [in]

Select the interrupt to be enabled. It could be IO_INT0, IO_INT1, IO_INT2 or IO_INT3.

i32DebounceClk [in]

The debounce timing is 2^(i32DebounceClk) * APB clock.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_DisableIntDebounce

Prototype

ERRCODE DrvGPIO_DisableIntDebounce(DRVGPIO_INT_NUM intNum);

Description

To disable the debounce function of the specified GPIO interrupt.

Parameter

intNum [in]

Select the interrupt to be enabled. It could be IO_INT0, IO_INT1, IO_INT2 or IO_INT3.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_GetDebounceTime

Prototype

INT32 DrvGPIO_GetDebounceTime(VOID);

Description

Get the debounce timing setting.

Parameter

NOne

Include

Driver/DrvGPIO.h

Return Value

The debounce time setting..

DrvGPIO_EnableWakeupInt

Prototype

ERRCODE DrvGPIO_EnableWakeupInt(DRVGPIO_INT_NUM intNum);

Description

Enable the weakup function of specified GPIO interrupt.

Parameter

intNum [in]

Select the interrupt to be enabled. It could be IO_INT0, IO_INT1, IO_INT2 or IO_INT3.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_DisableWakeupInt

Prototype

ERRCODE DrvGPIO_EnableWakeupInt(DRVGPIO_INT_NUM intNum);

Description

Disable the weakup function of specified GPIO interrupt.

Parameter

intNum [in]

Select the interrupt to be enabled. It could be IO_INT0, IO_INT1, IO_INT2 or IO_INT3.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_GetLatchValue

Prototype

UINT32 DrvGPIO_GetLatchValue(DRVGPIO_PORT port);

Description

The GPIO port value will be latched when GPIO interrupt. This function could be used to get the latched data.

Parameter

port [in]

Specified GPIO port. It could be GPIOA, GPIOB or GPIOC.

Include

Driver/DrvGPIO.h

Return Value

Return the value latched when GPIO interrupt.

DrvGPIO_GetIntStatus

Prototype

PUINT32 DrvGPIO_GetIntStatus(VOID);

Description

This function is used to return a pointer to the GPIO interrupt status register. The first 32-bit(UINT32) is bit0~15 for GPIOA and bit16~25 for GPIOB. The second 32-bit(UINT32) is bit0~10 for GPIOC

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

A pointer to the GPIO interrupt status register.

DrvGPIO_SetPadReg0

Prototype

ERRCODE DrvGPIO_SetPadReg0(

DRVGPIO_FUNC pwm,
UINT32 u32OutIOBitMap,
UINT32 u32InIOSelect

);

Description

To set the PAD_REG0 to configure the PWM multi-function pins.

Parameter

pwm [in]

Select the PWM channel to configure. It could be FUNC_PWMT0 ~ FUNC_PWMT3.

u32OutIOBitMap [in]

Enable the PWM output through the specified I/O pin in bit map manner. There are 5 bits in this argument and its mapping is as follows:

	BIT0	BIT1	BIT2	BIT3	BIT4
FUNC_PWMT0	GPA[12]	GPB[1]	GPC[3]	GPC[7]	GPB[8]
FUNC_PWMT1	GPA[13]	GPB[2]	GPC[4]	GPC[8]	GPB[9]
FUNC_PWMT2	GPA[15]	GPB[3]	GPC[5]	GPC[9]	GPB[6]
FUNC_PWMT3	GPB[0]	GPB[4]	GPC[6]	GPC[10]	GPB[7]

u32InIOSelect [in]

Select the PWM input through the specified I/O pin. it could be 0~4 and Other value will disable the PWM input. The setting is as follows:

	0	1	2	3	4	Others
FUNC_PWMT0	GPA[12]	GPB[1]	GPC[3]	GPC[7]	GPB[8]	OFF
FUNC_PWMT1	GPA[13]	GPB[2]	GPC[4]	GPC[8]	GPB[9]	OFF
FUNC_PWMT2	GPA[15]	GPB[3]	GPC[5]	GPC[9]	GPB[6]	OFF
FUNC_PWMT3	GPB[0]	GPB[4]	GPC[6]	GPC[10]	GPB[7]	OFF

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_SetPadReg1

Prototype

ERRCODE DrvGPIO_SetPadReg1(

```
DRVGPIO_FUNC ip,
UINT32 u32Option
);
```

Description

To set the PAD_REG1 to configure the IP multi-function pins.

Parameter

ip [in]

Select the IP funciton. It could be FUNC_ICE, FUNC_I2C, FUNC_SPIM1, FUNC_SPIMS, FUNC_SPIM0, FUNC_UART0, FUNC_UART1, FUNC_ADC.

u32Option [in]

To set the operation option for each IP. The detail option is as following table:

	0	1	2	3
FUNC_ICE	OFF	GPB[5:9]	N/A	N/A
FUNC_I2C	OFF	GPA[15], GPB[0]	GPC[9:10]	N/A
FUNC_SPIM1	OFF	GPB[5:7]	GPC[0:2]	N/A
FUNC_SPIMS	OFF	GPA[11:14]	N/A	N/A
FUNC_SPM0	OFF	GPA[8:10]	N/A	N/A
FUNC_UART0	OFF	GPB[1:2]	GPB[1:4]	GPB[3:4]
FUNC_UART1	OFF	GPC[5:6]	GPC[5:8]	GPC[7:8]
FUNC_ADC	8 bits bit-map control			

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_SetPadReg2

Prototype

ERRCODE DrvGPIO_SetPadReg2(UINT32 u32Option);

Description

To set the PAD_REG2 to configure the USB Detection multi-function pins.

Parameter

u32Option [in]

To set the operation option to select the USB Detection pin. The detail option is as following table:

Option	USB Detect Pin
0	OFF
1	GPA[14]
2	GPA[15]
3	GPB[0]
4	GPB[1]
5	GPB[2]
6	GPB[3]
7	GPB[4]
8	GPB[8]
9	GPB[9]
10	GPC[0]
11	GPC[1]
12	GPC[2]
13	GPC[3]
14	GPC[4]
15	OFF

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success.

DrvGPIO_InitFunction

Prototype

ERRCODE DrvGPIO_InitFunction (DRVGPIO_FUNC function, INT32 option);

Description

To initialize the multi-function pin's of the specified function

Parameter

function [in]

Set the function of IO pin. It could be:

FUNC_GPIO/FUNC_PWMT0/FUNC_PWMT1/FUNC_PWMT2/FUNC_PWMT3/
FUNC_MIC/FUNC_ADC/FUNC_ADCIN0/FUNC_ADCIN1/FUNC_ADCIN2/
FUNC_ADCIN3/FUNC_ADCIN4/FUNC_ADCIN5/FUNC_ADCIN6/
FUNC_ADCIN7/FUNC_UART0/FUNC_UART1/FUNC_SPIM0/FUNC_SPIM1
FUNC_SPIMS/FUNC_I2C/FUNC_USBDET

option [in]

Select the different pin to output the specified function

Function	Option	IO
FUNC_GPIO	N/A	All GPIO
FUNC_PWMT0	0~4	GPA12, GPB1, GPB8, GPC3, GPC7
FUNC_PWMT1	0~4	GPA13, GPB2, GPB9, GPC4, GPC8
FUNC_PWMT2	0~4	GPA15, GPB3, GPB6, GPC5, GPC9
FUNC_PWMT3	0~4	GPB0, GPB4, GPB7, GPC6, GPC10
FUNC_MIC	N/A	{GPA0, GPA1}
FUNC_ADC	N/A	{GPA0 ~ GPA7}
FUNC_ADCIN0	N/A	GPA0
FUNC_ADCIN1	N/A	GPA1
FUNC_ADCIN2	N/A	GPA2
FUNC_ADCIN3	N/A	GPA3
FUNC_ADCIN4	N/A	GPA4
FUNC_ADCIN5	N/A	GPA5
FUNC_ADCIN6	N/A	GPA6
FUNC_ADCIN7	N/A	GPA7
FUNC_UART0	N/A	GPB1 ~ GPB4
FUNC_UART1	N/A	GPC5 ~ GPC8
FUNC_SPIM0	N/A	{GPA8, GPA9, GPA10, SPIM0_SS}
FUNC_SPIM1	0~1	{GPB5~GPB7},{GPC0~GPC2}
FUNC_I2C	0~1	{GPA15,GPB0},{GPC9, GPC10}
FUNC_USBDET	0~13	GPA14,GPA15,GPB0,GPB1,GPB2, GPB3,GPB4,GPB8,GPB9,GPC0, GPC1,GPC2, GPC3,GPC4

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments
E_DRVGPIO_GROUP_PIN	The IO pin can't be configured individually

DrvGPIO_InitBitFunction

Prototype

ERRCODE DrvGPIO_InitBitFunction (DRVGPIO_PORT port, DRVGPIO_BIT bit, DRVGPIO_FUNC function);

Description

To initialize the pin function of the specified mult-function pin

Parameter

port [in]

Select the port which to be initialized. It could be GPIOA, GPIOB or GPIOC.

bit [in]

Select the bit of the specified port. It could be 0~15, for GPIOA, 0~9 for GPIOB, 0~10 for GPIOC

function [in]

Set the function of IO pin. It could be:

FUNC_GPIO/FUNC_PWMT0/FUNC_PWMT1/FUNC_PWMT2/FUNC_PWMT3/

FUNC_MIC/FUNC_ADC/FUNC_ADCIN0/FUNC_ADCIN1/FUNC_ADCIN2/

FUNC_ADCIN3/FUNC_ADCIN4/FUNC_ADCIN5/FUNC_ADCIN6/

FUNC_ADCIN7/FUNC_UART0/FUNC_UART1/FUNC_SPIM0/FUNC_SPIM1

FUNC_SPIMS/FUNC_I2C/FUNC_USBDET

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments
E_DRVGPIO_GROUP_PIN	The IO pin can't be configured individually

DrvGPIO_GetDoutBit

Prototype

INT32 DrvGPIO_GetDoutBit (DRVGPIO_PORT port, INT32 i32Bit);

Description

Get the value of the specified IO bit from GPIO Dout register.

Parameter

port [in]

Select the port which to be initialized. It could be GPIOA, GPIOB or GPIOC.

bit [in]

Select the bit of the specified port. It could be 0~15, for GPIOA, 0~9 for GPIOB, 0~10 for GPIOC

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified IO bit

DrvGPIO_GetVersion

Prototype

UINT32

DrvGPIO_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvGPIO.h

Return Value

Version number:

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

9. DrvI2C Introduction

9.1. Introduction

At the low end of the spectrum of communication options for "inside the box" communication is I2C ("eye-squared-see"). The name I2C is shorthand for a standard Inter-IC (integrated circuit) bus.
I2C provides good support for communication with various slow and on-board peripheral devices that are accessed intermittently, while being extremely modest in its hardware resource needs. It is a simple, low-bandwidth, and short-distance protocol. Most available I2C devices operate at speeds up to 400Kbps with some venturing up into the low megahertz range. I2C is easily used to link multiple devices together since it has a built-in addressing scheme. I2C device could act as master or slave. However, the hardware I2C can only act as master.

10. DrvI2C APIs Specification

10.1. Macros

_DRVI2C_SCK_SETHIGH

Prototype

VOID _DRVI2C_SCK_SETHIGH (VOID);

Description

Set signal SCK to high.

Include

Driver/DrvI2C.h

_DRVI2C_SCK_SETLOW

Prototype

VOID _DRVI2C_SCK_SETLOW (VOID);

Description

Set signal SCK to low.

Include

Driver/DrvI2C.h

_DRVI2C_SDA_SETHIGH

Prototype

VOID _DRVI2C_SCK_SETHIGH (VOID);

Description

Set signal SDA to high.

Include

Driver/DrvI2C.h

_DRVI2C_SDA_SETLOW

Prototype

VOID _DRVI2C_SDA_SETLOW (VOID);

Description

Set signal SDA to low.

Include

Driver/DrvI2C.h

_DRVI2C_SCK_GETVALUE

Prototype

BOOL

_DRVI2C_SCK_GETVALUE (VOID);

Description

Get the state of signal SCK.

Include

Driver/DrvI2C.h

_DRVI2C_SDA_GETVALUE

Prototype

BOOL

_DRVI2C_SDA_GETVALUE (VOID);

Description

Get the state of signal SDA.

Include

Driver/DrvI2C.h

10.2. Functions

DrvI2C_Open

Prototype

ERRCODE DrvI2C_Open (UINT32 u32I2cClock);

Description

To open the I2C hardware and configure the I2C bus clock.

Parameter

u32I2cClock [in]

To configure the I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS Succeed

DrvI2C_Close

Prototype

VOID DrvI2C_Close (VOID);

Description

To close the I2C hardware.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_Delay

Prototype

VOID DrvI2C_Delay (UINT32 nCount);

Description

This is a delay function.

Parameter

nCount [in]

The delay count in micro-second.

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_WaitReady

Prototype

ERRCODE DrvI2C_WaitReady (VOID);

Description

Wait for I2C hardware ready.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Success
E_DRVI2C_ARBIT_LOSE	Bus arbitration error
E_DRVI2C_TIME_OUT	I2C time out

DrvI2C_SendStart

Prototype

ERRCODE DrvI2C_SendStart (VOID);

Description

To send START signal to I2C bus.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Success
E_DRVI2C_ARBIT_LOSE	Bus arbitration error
E_DRVI2C_TIME_OUT	I2C time out

DrvI2C_SendStop

Prototype

ERRCODE DrvI2C_SendStop (VOID);

Description

To send STOP signal to I2C bus.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Success
E_DRVI2C_ARBIT_LOSE	Bus arbitration error
E_DRVI2C_TIME_OUT	I2C time out

DrvI2C_WriteByte

Prototype

```
ERRCODE DrvI2C_WriteByte (
   BOOL     bStart,
   UINT8    u8Data,
   BOOL     bCheckAck,
   BOOL     bStop
);
```

Description

Send a byte to I2C bus.

Parameter

bStart [in]

TRUE: Send START condition to I2C bus

FALE: Don't send START condition to I2C bus

u8Data [in]

The byte to send through I2C bus.

bCheckAck [in]

Enable to check ACK after send data.

bStop [in]

TRUE: Send STOP condition to I2C bus

FALE: Don’t send STOP condition to I2C bus

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Success
E_DRVI2C_ARBIT_LOSE	Bus arbitration error
E_DRVI2C_TIME_OUT	I2C time out
E_DRVI2C_NACK	NACK received

DrvI2C_ReadByte

Prototype

```
ERRCODE DrvI2C_ReadByte (
    BOOL      bStart,
    PUINT8    pu8ReadData,
    BOOL      bSendAck,
    BOOL      bStop
);
```

Description

Read one byte from device

Parameter

bStart [in]

TRUE Send START condition to I2C bus

FALE Don’t send START condition to I2C bus

pu8ReadData [in]

Read a byte from I2C bus.

bSendAck [in]

TRUE Send NACK to device

FALSE Send ACK to device

bStop [in]

TRUE Send STOP condition to I2C bus

FALE Don’t send STOP condition to I2C bus

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Succeed

E_DRVI2C_ARBIT_LOSE	Bus arbitration error
E_DRVI2C_TIME_OUT	I2C time out

DrvI2C_GetIntFlag

Prototype

BOOL DrvI2C_GetIntFlag (VOID);

Description

Get I2C interrupt flag. There are three cases which cause the I2C interrupt:

1. Transfer has been completed.
2. Transfer has not been completed but slave response NACK in burst transfer mode.
3. Arbitration error

Parameter

None

Include

Driver/DrvI2C.h

Return Value

TRUE	The I2C interrupt flag is set.
FALSE	The I2C interrupt flag is not set.

DrvI2C_ClrIntFlag

Prototype

VOID DrvI2C_ClrIntFlag (VOID);

Description

Clear the I2C interrupt flag.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_InitSDASCK

Prototype

VOID DrvI2C_InitSDASCK (VOID);

Description

Set pin SDA and SCK to high state (initialize state).

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_IsBusy

Prototype

BOOL DrvI2C_IsBusy(VOID);

Description

This function is used to check if I2C is in busy.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Return TRUE if the I2C is in busy.

DrvI2C_IsBusBusy

Prototype

BOOL DrvI2C_IsBusBusy(VOID);

Description

This function is used to check if I2C bus is in busy. The function will return TRUE if there is START signal on bus and it will return FALSE until STOP signal.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Return TRUE if the I2C bus is in busy.

DrvI2C_IsArbitLost

Prototype

BOOL DrvI2C_IsArbitLost (VOID);

Description

If the arbitration error is found on I2C bus, this function will return TRUE and send START and STOP command to the bus to try to recover the error.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

TRUE: Lost arbitration. It means error on the transfer.

FALSE: No arbitration lost.

DrvI2C_SetBurstCnt

Prototype

ERRCODE DrvI2C_SetBurstCnt(UINT8 u8BurstCnt);

Description

To set the burst transfer count. The I2C could be up to 4 transfers when trigger to send/get data

Parameter

u8BurstCnt [in]

The sequential transfer number. It could be 1~4.

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Success
E_DRVI2C_WRONG_LENGTH	Invalid sequential transfer number.

DrvI2C_SetTxData

Prototype

VOID DrvI2C_SetTxData(UINT32 u32TxData);

Description

Set the data to send.

Parameter

u32TxData [in]

Specify the transfer data. High byte is transferred first.

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_SendCmd

Prototype

VOID DrvI2C_SendCmd(I2C_CMD cmd);

Description

Send the I2C command to the hardware to do the relative action.

Parameter

cmd [in].

The I2C command. It could be I2C_ACK, I2C_WRITE, I2C_READ, I2C_STOP, I2C_START. It is also able to send two commands at the same time by OR, i.e. I2C_START | I2C_READ | I2C_ACK| I2C_STOP. The sequences of the I2C actions are START ➔ READ ➔ ACK ➔ STOP when these commands are send at the same time.

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_IsACK

Prototype

BOOL DrvI2C_IsACK (VOID);

Description

This function is used to check if ACK received from slave after sending data to slave device.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

TRUE	Acknowledge received.
FALSE	No acknowledge received.

DrvI2C_GetRxData

Prototype

UINT8 DrvI2C_GetRxData (VOID);

Description

To read the last byte which is received from I2C bus

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Return the last byte which is received from I2C bus.

DrvI2C_EnableInt

Prototype

VOID DrvI2C_EnableInt(VOID);

Description

To enable the I2C interrupt.

Parameter

bIsInterrupt [in]

TRUE	Enable the hardware I2C interrupt
FALSE	Disable the hardware I2C interrupt

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_IsIntEnabled

Prototype

BOOL DrvI2C_IsIntEnabled(VOID);

Description

Get the interrupt enable or disable status.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

TRUE = I2C Interrupt enabled. FALSE = I2C Interrupt disabled.

DrvI2C_ClearInt

Prototype

VOID DrvI2C_ClearInt(VOID);

Description

To clear the I2C interrupt flag.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_GetVersion

Prototype

UINT32

DrvI2C_GetVersion (VOID);

Description

Return the current version number.

Include

Driver/DrvI2C.h

Return Value

Version number :

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

11. DrvPWM Introduction

11.1. PWM Introduction

The basic components in a PWM set is pre-scalar, clock divider, 16-bit counter, 16-bit comparator, inverter, dead-zone generator. They are all driven by system 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-scalar. 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 toggle mode, it is reloaded automatically and start to generate next cycle. User can set counter as one-shot mode instead of toggle mode. If counter is set as one-shot mode, counter will stop and generate one interrupt request when it reaches zero.

12. DrvPWM APIs Specification

12.1. Constant Definition

Name	Value	Description
DRVPWM_TIMER0	0x0	PWM Timer 0
DRVPWM_TIMER1	0x1	PWM Timer 1
DRVPWM_TIMER2	0x2	PWM Timer 2
DRVPWM_TIMER3	0x3	PWM Timer 3
DRVPWM_CAP0	0x0	PWM Capture 0
DRVPWM_CAP1	0x1	PWM Capture 1
DRVPWM_CAP2	0x2	PWM Capture 2
DRVPWM_CAP3	0x3	PWM Capture 3
DRVPWM_CAP_NO_INT	0	No PWM Capture 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_TOGGLE_MODE	TRUE	PWM Timer Toggle mode
DRVPWM_ONE_SHOT_MODE	FALSE	PWM Timer One-shot mode

12.2. Functions

DrvPWM_IsTimerEnabled

Prototype

BOOL

DrvPWM_IsTimerEnabled (

UINT8 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

TURE: The specified timer is enabled.

FALSE: The specified timer is disabled.

DrvPWM_SetTimerCounter

Prototype

VOID DrvPWM_SetTimerCounter (

UINT8 u8Timer,

UINT16 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)

Include

Driver/DrvPWM.h

Return Value

None

Note

If the counter is set to 0, the timer will stop.

DrvPWM_GetTimerCounter

Prototype

```
UINT32  DrvPWM_GetTimerCounter (
   UINT8       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.

DrvPWM_EnableInt

Prototype

```
VOID DrvPWM_EnableInt (
    UINT8       u8Timer,
    UINT8       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

DrvPWM_DisableInt

Prototype

```
VOID DrvPWM_DisableInt (
    UINT8       u8Timer,
    UINT8       u8Int
);
```

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.

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.

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_ClearInt

Prototype

```
VOID DrvPWM_ClearInt (
    UINT8       u8Timer
```

);

Description

This function is used to clear 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

DrvPWM_GetIntFlag

Prototype

BOOL DrvPWM_GetIntFlag (

UINT8 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

TRUE: The specified interrupt occurs.

FLASE: The specified interrupt doesn't occur.

DrvPWM_GetRisingCounter

Prototype

```
UINT16 DrvPWM_GetRisingCounter (
   UINT8       u8Capture
);
```

Description

The 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

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

DrvPWM_GetFallingCounter

Prototype

```
UINT16
DrvPWM_GetFallingCounter (
```

UINT8 u8Capture

);

Description

The 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

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

DrvPWM_GetCaptureIntStatus

Prototype

BOOL DrvPWM_GetCaptureIntStatus (

UINT8 u8Capture,

UINT8 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.

DRVPWM_CAP_RISING_FLAG: The capture rising interrupt flag.

DRVPWM_CAP_FALLING_FLAG: The capture falling interrupt flag.

Include

Driver/DrvPWM.h

Return Value

TRUE: The specified interrupt occurs.

FALSE: The specified interrupt doesn't occur.

DrvPWM_ClearCaptureIntStatus

Prototype

```
VOID DrvPWM_ClearCaptureIntStatus (
    UINT8       u8Capture,
    UINT8       u8IntType
);
```

Description

Clear the rising / falling transition interrupt 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.

DRVPWM_CAP_RISING_FLAG: The capture rising interrupt flag.

DRVPWM_CAP_FALLING_FLAG: The capture falling interrupt flag.

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_Open

Prototype

```
VOID DrvPWM_Open (VOID);
```

Description

Enable PWM engine clock and reset PWM.

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_Close

Prototype

VOID DrvPWM_Close (VOID);

Description

Disable PWM engine clock and the I/O enable

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_EnableDeadZone

Prototype

```
VOID DrvPWM_EnableDeadZone (
    UINT8       u8Timer,
    UINT8       u8Length,
    BOOL        bEnableDeadZone,
);
```

Description

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

Parameter

u8Timer [in]

Specify the timer

DRVPWM_TIMER0: PWM timer 0.

DRVPWM_TIMER1: PWM timer 1.

DRVPWM_TIMER2: PWM timer 2.

DRVPWM_TIMER3: PWM timer 3.

u8Length [in]

Specify Dead Zone Length : 0~255.

bEnableDeadZone [in]

Enable DeadZone (TRUE) / Diasble DeadZone (FALSE)

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_Enable

Prototype

```
VOID DrvPWM_Enable (
   UINT8      u8Timer,
   BOOL       bEnable
);
```

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.

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_SetTimerClk

Prototype

```
UINT32 DrvPWM_SetTimerClk (
    UINT8       u8Timer,
    S_DRVPWM_TIME_DATA_T *sPt
);
```

Description

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

***sPt [in]**

It includes the following parameter

u8Frequency: The timer/capture frequency

u8HighPulseRatio: High pulse ratio

u8Mode: DRVPWM_ONE_SHOT_MODE / DRVPWM_TOGGLE_MODE

bInverter: Inverter Enable (TRUE) / Inverter Disable (FALSE)

u8ClockSelector: Clock Selector

DRVPWM_CLOCK_DIV_1:

DRVPWM_CLOCK_DIV_2:

DRVPWM_CLOCK_DIV_4:

DRVPWM_CLOCK_DIV_8:

DRVPWM_CLOCK_DIV_16:

(The parameter takes effect when u8Frequency = 0)

u8PreScale: Prescale (2 ~ 256)

(The parameter takes effect when u8Frequency = 0)

u32Duty: Pulse duty

(The parameter takes effect when u8Frequency = 0 or u8Timer = DRVPWM_CAP0/DRVPWM_CAP1/DRVPWM_CAP2/DRVPWM_CAP3)

Include

Driver/DrvPWM.h

Return Value

TRUE: The specified interrupt occurs.

FLASE: The specified interrupt doesn't occur.

Note

1. The function will set the frequency property automatically when user set a nonzero frequency value
2. When setting the frequency value to zero, user also can set frequency property (Clock selector/Prescale/Duty) by himself.
3. The function can set the proper frequency property (Clock selector/Prescale) for capture function and user needs to set the proper pulse duty by himself.

DrvPWM_SetTimerIO

Prototype

```
VOID DrvPWM_SetTimerIO (
  UINT8     u8Timer,
  BOOL      bEnable
);
```

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.

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvPWM.h

Return Value

None

DrvPwm_GetVersion

Prototype

UINT32

DrvPwm_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvPWM.h

Return Value

Version number :

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

13. DrvRTC Introduction

13.1. General RTC Controller Introduction

Real Time Clock (RTC) block can be operated by independent power supply while the system power is off. The RTC uses a 32.768 KHz external crystal. A built in RTC is designed to generate the periodic interrupt signal. The period can be 0.25/ 0.5/ 1/ 2/ 4/ 8 second. There is RTC overflow counter and it can be adjusted by software.

13.2. RTC Features

- There is a time counter for user to check time.
- ”Power on” has “time out” function to avoid current always existence.
- When battery is low at wake-up moment, system hangs up; H/W provides “time out” function to turn off system.
- When battery is inserted, the default of switch is on. (low)
- Support time tick interrupt
- Support wake up function

14. DrvRTC APIs Specification

14.1. Constant Definition

Name	Value	Description
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_LEAP_YEAR	1	Leap year
DRVRTC_TICK_1_SEC	0	1 tick per second
DRVRTC_TICK_1_2_SEC	1	2 tick per second
DRVRTC_TICK_1_4_SEC	2	4 tick per second
DRVRTC_TICK_1_8_SEC	3	8 tick per second
DRVRTC_TICK_1_16_SEC	4	16 tick per second
DRVRTC_TICK_1_32_SEC	5	32 tick per second
DRVRTC_TICK_1_64_SEC	6	64 tick per second
DRVRTC_TICK_1_128_SEC	7	128 tick per second
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
DRVRTC_ALARM_INT	0x01	Alarm Interrupt
DRVRTC_TICK_INT	0x02	Tick Interrupt
DRVRTC_ALL_INT	0x03	All Interrupt
DRVRTC_IOC_IDENTIFY_LEAP_YEAR	0	Identify the leap year command
DRVRTC_IOC_SET_TICK_MODE	1	Set tick mode command
DRVRTC_IOC_GET_TICK	2	Get tick command
DRVRTC_IOC_RESTORE_TICK	3	Restore tick command

DRVRTC_IOC_ENABLE_INT	4	Enable interrupt command
DRVRTC_IOC_DISABLE_INT	5	Disable interrupt command
DRVRTC_IOC_SET_CURRENT_TIME	6	Set Current time command
DRVRTC_IOC_SET_ALAMRM_TIME	7	Set Alarm time command
DRVRTC_IOC_SET_FREQUENCY	8	Set Frequency command
DRVRTC_CURRENT_TIME	0	Current time
DRVRTC_ALARM_TIME	1	Alarm time

14.2. Functions

DrvRTC_SetFrequencyCompenation

Prototype

```
ERRCODE
DrvRTC_SetFrequencyCompenation (
    FLOAT      fnumber;
);
```

Description

Set Frequency Compenation Data

Parameter

fnumber [in]

Specify the Compenation value.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FCR_VALUE: Wrong Compenation value

DrvRTC_WriteEnable

Prototype

```
ERRCODE
DrvRTC_WriteEnable (VOID);
```

Description

Access PW to AER to make access other register enable

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FAILED : Failed.

DrvRTC_Init

Prototype

ERRCODE

DrvRTC_Init (VOID);

Description

Initial RTC and install ISR.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_Open

Prototype

ERRCODE

DrvRTC_Open (

S_DRVRTC_TIME_DATA_T *sPt

);

Description

Set Current time and install ISR.

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

pfnAlarmCallBack : The alarm call back function pointer

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_Read

Prototype

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

Description

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

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

pfnAlarmCallBack : The alarm call back function pointer

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_Write

Prototype

```
ERRCODE
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

pfnAlarmCallBack : The alarm call back function pointer

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_Ioctl

Prototype

```
ERRCODE
DrvRTC_Ioctl (
    INT32           i32Num
    E_DRVRTC_CMD    eCmd,
    UINT32          u32Arg0,
    FLOAT           fArg1
);
```

Description

Support some commands for application.

Parameter

i32Num [in]

Not used

eCmd [in]

The command for application

DRVRTC_IOC_IDENTIFY_LEAP_YEAR: Get the leap year

DRVRTC_IOC_SET_TICK_MODE: Set Tick mode

DRVRTC_IOC_GET_TICK: Get the tick counter

DRVRTC_IOC_RESTORE_TICK : Restore the tick counter

DRVRTC_IOC_ENABLE_INT: Enable interrupt

DRVRTC_IOC_DISABLE_INT: Disable interrupt

DRVRTC_IOC_SET_CURRENT_TIME: Set currnet time

DRVRTC_IOC_SET_ALAMRM_TIME: Set alarm time

DRVRTC_IOC_SET_FREQUENCY : Set Frequency Compensation Data

u32Arg0 [in]

1. The buffer address to store the return leap year value (DRVRTC_IOC_IDENTIFY_LEAP_YEAR)
2. The buffer address that stored the tick mode data (DRVRTC_IOC_SET_TICK_MODE)
3. The buffer address to store the return tick number(DRVRTC_IOC_GET_TICK)
4. The buffer address to store the interrupt type to be enabled (DRVRTC_IOC_ENABLE_INT)
5. The buffer address to store the interrupt type to be disabled (DRVRTC_IOC_ DISABLE _INT)
6. The buffer address that stored the Frequency Compensation value (DRVRTC_IOC_SET_FREQUENCY)

fArg1 [in]

Not used.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_Close

Prototype

ERRCODE

DrvRTC_Close (VOID);

Description

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_GetVersion

Prototype

UINT32

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

15. DrvSIO Introduction

15.1. SerialIO Introduction

The SerialIO driver provides a printf like interface to print out the string. This library could be used for debugging, print message and get characters through UART or SWI.

15.2. SerialIO Feature

The SerialIO includes following features:

- Print formatted string.
- Print message and get characters through UART port
- Print message and get characters through semi-hosted SWI

16. DrvSIO APIs Specification

16.1. Functions

DrvSIO_printf

Prototype

VOID DrvSIO_printf (

char * pcStr,

...

)

Description

This function provides the c like printf function to print out the message through UART0. The UART0 must be initialized before using the DrvSIO_printf. Please refer to UART driver for more information to initialize the UART0.

The DrvSIO_printf can support most of the formatted string however it can't support floating-point.

Parameter

pcStr [in]

Formatted string

…[in]

Formatted string parameters

Include

Driver/DrvSerialIO.h

Return Value

None

DrvSIO_SemiPrintf

Prototype

VOID DrvSIO_SemiPrintf (

char * pcStr,

...

)

Description

The usage of this function is the same as DrvSIO_printf. However, it uses SWI of ARM to output the characters, i.e. it uses the semi-hosted interface of ARM to print out the message. Therefore, the semi-hosted function of the ARM debug tool must be enabled before using this function.

Parameter

pcStr [in]

Formatted string

…[in]

Formatted string parameters

Include

Driver/DrvSerialIO.h

Return Value

None

DrvSIO_GetChar

Prototype

INT8

DrvSIO_GetChar (VOID)

Description

This function is used to get a character form the UART0. The UART0 must be initialized before this function being called. Please refer to UART driver for more information.

Parameter

None

Include

DrvSerialIO.h

Return Value

The character value

DrvSIO_SemiGetChar

Prototype

INT8

DrvSIO_SemiGetChar (VOID)

Description

This function is used to get a character form the ARM semi-hosted SWI. The semi-hosting function of the ARM debug tool must be enabled before using this function.

Parameter

None

Include

Driver/DrvSerialIO.h

Return Value

The character value

DrvSIO_kbhit

Prototype

BOOL

DrvSIO_kbhit (VOID)

Description

The DrvSIO_kbhit could be used to check if any character is inputted through UART0.

Parameter

None

Include

DrvSerialIO.h

Return Value

TRUE	There is character input.
FALSE	No character input.

DrvSIO_PutChar

Prototype

VOID DrvSIO_PutChar (

UINT8 ch

)

Description

The DrvSIO_PutChar is used to send out a character through UART0.

Parameter

ch [in]

The character to be sent out.

Include

DrvSerialIO.h

Return Value

None

DrvSIO_SemiPutChar

Prototype

VOID DrvSIO_SemiPutChar (

UINT8 ch

)

Description

The DrvSIO_PutChar is used to send out a character through semi-hosted SWI.

Parameter

ch [in]

The character to be sent out.

Include

DrvSerialIO.h

Return Value

None

DrvSIO_SetDebugPort

Prototype

VOID DrvSIO_SetDebugPort (

UINT8 u8Port

)

Description

The DrvSIO_SetDebugPort is used to select UART0 or UART1 for debug port.

Parameter

u8Port [in]

To set the output UART port from UART0 or UART1. 0 = UART0, 1 = UART1.

Include

DrvSerialIO.h

Return Value

None

DrvSIO_GetVersion

Prototype

UINT32

DrvSIO_GetVersion (VOID);

Description

Return the current version number of driver.

Include

DrvSerialIO.h

Return Value

Version number:

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

17. DrvSI2C Introduction

17.1. SI2C Introduction

At the low end of the spectrum of communication options for "inside the box" communication is I2C ("eye-squared-see"). The name I2C is shorthand for a standard Inter-IC (integrated circuit) bus.

I2C provides good support for communication with various slow and on-board peripheral devices that are accessed intermittently, while being extremely modest in its hardware resource needs. It is a simple, low-bandwidth, and short-distance protocol. Most available I2C devices operate at speeds up to 400Kbps with some venturing up into the low megahertz range. I2C is easily used to link multiple devices together since it has a built-in addressing scheme.

The software I2C driver uses no special hardware IP for I2C in 501, however, it uses GPIO pins to send or receive the clock and data signals to generate the I2C signals. This is what called software I2C (SI2C) and the SI2C driver doesn’t support master mode.

17.2. SI2C Feature

The SI2C driver is a low cost software solution. Upper layer can specify any GPIO pins to simulation I2C function.

- Bit unit.
- Low cost.
- Flexible

18. DrvSI2C APIs Specification

18.1. Functions

DrvSI2C_Open

Prototype

```
ERRCODE DrvSI2C_Open (
    DRVGPIO_PORT    u32SckIoPort,
    UINT32          u32SckIoBit,
    DRVGPIO_PORT    u32SdaIoPort,
    UINT32          u32SdaIoBit
);
```

Description

To specify the GPIO pins for SDA/SCK of I2C bus and initial the software I2C.

Parameter

u32SckIoPort [in]

The GPIO port where the SCK IO pin belong to.

32SckIoBit [in]

The bit for SCK IO pin.

u32SdaIoPort [in]

The GPIO port where the SDA IO pin belong to.

u32SdaIoBit [in]

The bit for SDA IO pin.

Include

Driver/DrvI2C.h

Return Value

E_SUCCESS	Success

DrvSI2C_Close

Prototype

VOID DrvSI2C_Close (VOID);

Description

Reset the SDA/SCK IO pins to be input to disable software I2C.

Parameter

None

Include

Driver/DrvSI2C.h

Return Value

None

DrvSI2C_SendStart

Prototype

VOID DrvSI2C_SendStart (VOID);

Description

Send start condition to I2C bus.

Parameter

None

Include

Driver/DrvSI2C.h

Return Value

None

DrvI2c_SendStop

Prototype

VOID DrvI2c_SendStop (VOID);

Description

Send stop condition to I2C bus.

Parameter

None

Include

Driver/DrvSI2C.h

Return Value

None

DrvSI2C_ReadByte

Prototype

```
ERRCODE DrvSI2C_ReadByte (
    BOOL    bStart,
    PUINT8  pu8ReadData,
    BOOL    bSendAck,
    BOOL    bStop
);
```

Description

Read one-byte data from device and the most significant bit (MSB) first.

Parameter

bStart [in]

Enable to send START signal before send data.

pu8ReadData [out]

The byte to read through I2C bus.

bSendAck [in]

Enable to send ACK after read data.

bStop [in]

Enable to send STOP signal at the end.

Include

Driver/DrvSI2C.h

Return Value

E_SUCCESS	Success

DrvSI2C_WriteByte

Prototype

```
ERRCODE DrvSI2C_WriteByte (
    BOOL    bStart,
```

```
    UINT8       u8Data,
    BOOL        bCheckAck,
    BOOL        bStop
);
```

Description

Write one byte to device and the most significant bit (MSB) first.

Parameter

bStart [in]

TRUE = Enable to send START signal before send data.

u8Data [in]

The byte to send through I2C bus.

bCheckAck [in]

Enable to check ACK after send data.

bStop [in]

TRUE = Enable to send STOP signal at the end.

Include

Driver/DrvSI2C.h

Return Value

E_SUCCESS	Success
E_DRVI2C_NACK	NACK received

DrvI2c_GetVersion

Prototype

```
UINT32
DrvI2c_GetVersion (VOID);
```

Description

Return the current version number.

Include

Driver/DrvSI2C.h

Return Value

Version number :

31	30:24	23:16	15:0

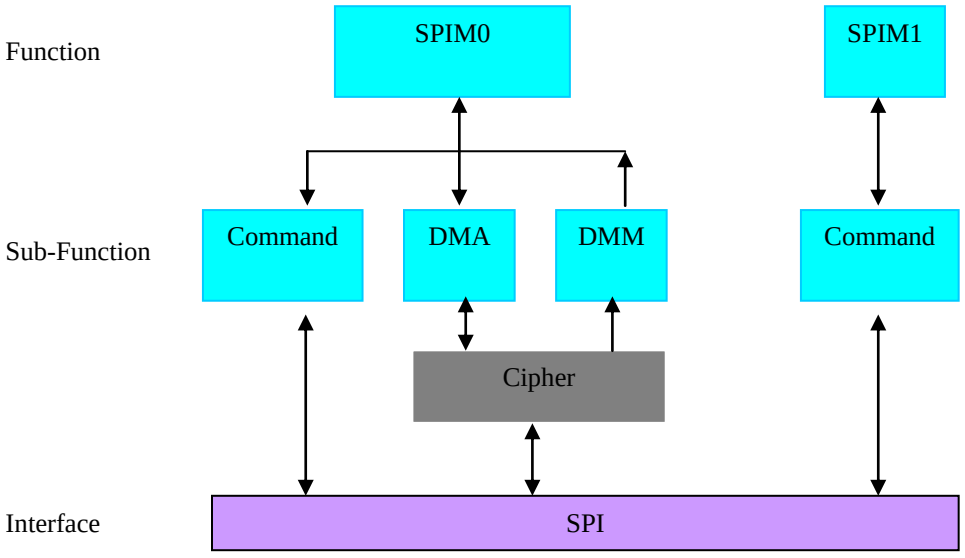
● 0: not an error code ● 1: an error code	Should be 0.	Module ID. See Module_ID.h	Error ID

19. DrvSPIM Introduction

19.1. SPIM Introduction

The SPIM is designed as a special SPI hardware, it supports SPI Master only and two channels called SPIM0 and SPIM1. The SPIM0 and SPIM1 use the same SPI kernel with different chip selection I/O, thus they can't transfer data through SPI at the same time. Furthermore, SPIM0 is designed to let CPU can fetch the code/data in SPI ROM directly and this is called direct memory map mode. After necessary initializations, the contents in SPI ROM could be mapping to 0x40000000 and CPU could execute the code in SPI ROM by fetch code from 0x4xxxxxxx directly. The SPIM1 can't work with SPIM0 at the same time, thus it is invalid in direct memory map mode, furthermore, the SPIM1 doesn't have dedicated chip select I/O. Therefore, A GPIO pin is necessary to be the chip select I/O when using SPIM1.

SPIM0 supports 3 sub functions. The first is command-read and command-write. The second is DMA-read and DMA-write. The third is direction memory mapping (DMM) mode. However, the SPIM1 only supports command-read and command-write. In the DMA mode or DMM mode of SPIM0, it also supports security function to encrypt/decrypt the contents in SPI ROM directly. This security function lets the CPU could access the encrypted SPI ROM data without software efforts.



19.2. SPIM Feature

The Advanced Interrupt Controller includes following features:

- AMBA AHB interface compatible
- Support SPI master mode
- Variable length of transfer word up to 32 bits
- Provide burst mode operation, transmit/receive can be executed up to four times in one transfer
- 2 slave/device select lines. One is dedicated chip select pin. The other is emulation by general purpose I/O.

20. DrvSPIM APIs Specification

20.1. Functions

DrvSPIM_Init

Prototype

VOID DrvSPIM_Init (VOID);

Description

This function is used to initialize the SPIM including enable the engine clock, reset IP and install default interrupt.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_Open

Prototype

```
ERRCODE DrvSPIM_Open (
    UINT32          u32SpimFreq,
    SPIM_TYPE       type,
    INT32           i32BitLength
);
```

Description

This function is used to initialize the SPIM, including SPI clock frequency, SPI bus timing and number of bits per transfer. The SPI clock is calculated according to APB clock and may not be exactly the same as u32SpimFreq. Furthermore, the function DrvSPIM_SetDivider could be used if the user only want to change the SPIM0 clock to slowdown or speedup the SPI bus clock.

Parameters

u32SpimFreq [in]

The SPI clock frequency.

type [in]

The SPI sinal timing. It could be SPI_TYPE0, SPI_TYPE1, SPI_TYPE2 or SPI_TYPE3. The default is SPI_TYPE1.

i32BitLength [in]

The number of bits per transfer. It could be 1~32 bits.

Include

Driver/DrvSPIM.h

DrvSPIM_Close

Prototype

VOID DrvSPIM_Close (VOID);

Description

Release pin function and disable clock for SPIR engine.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_SetDivider

Prototype

VOID DrvSPIM_SetDivider (UINT32 u32Divider);

Description

This function is used to set the divider of SPIM. The SPI SCLK = APB Clock / (u32Divider * 2). if u32Divider=0 ==> SCLK = APB Clock

Parameters

u32Divider [in]

The divider number.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_GetSPIMClock

Prototype

UINT32 DrvSPIM_GetSPIMClock (VOID);

Description

This function is used to get the clock frequency of SPIM.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

The clock frequency of SPIM in kHz.

DrvSPIM_SetBurstCnt

Prototype

ERRCODE DrvSPIM_SetBurstCnt (UINT32 u32TransferNumbers);

Description

To set the sequential transfer numbers and the number of bits per transfer.

Parameters

u32TransferNumbers [in]

This field specifies how many transmit/receive numbers should be executed in one transfer. Max transfer number is 4 and minimum is 1.

Include

Driver/DrvSPIM.h

Return Value

E_SUCCESS	Success
E_DRVSPIM_WRONG_TRANSFER_NUMBER	Wrong transfer number

DrvSPIM_GetBurstLength

Prototype

VOID DrvSPIM_GetBurstLength (

PUINT32 pu32TransferNumbers,

PUINT32 pu32TransferBits

);

Description

To get the sequential transfer numbers and the number of bits per transfer.

Parameters

pu32TransferNumbers [in]

This field specifies how many transmit/receive numbers should be executed in one transfer. Max transfer number is 4.

pu32TransferBits [in]

This field specifies how many bits are transmitted in one transmit/receive. Up to 32 bits can be transmitted. Max transfer bit is 32 bits for each transfer.

Value	Transfer bits
1	1 bit
…	…
31	31 bits
0	32 bits

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_SetBurstInterval

Prototype

ERRCODE DrvSPIM_SetBurstInterval (UINT32 u32SuspendInterval);

Description

To get the sequential transfer numbers and the number of bits per transfer.

Parameters

u32SuspendInterval [in]

The interval between sequential transfers. The unit is in SCLK of SPIM. It could be 2~17 SCLK

Include

Driver/DrvSPIM.h

Return Value

E_SUCCESS	Success.
E_DRVSPIM_WRONG_INTERVAL	The interval is out of range.

DrvSPIM_GetBurstInterval

Prototype

UINT32 DrvSPIM_GetBurstInterval(VOID);

Description

To get the interval between sequential transfers. The unit is in SCLK clock of SPIM.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

The interval between sequential transfers. The unit is in SCLK clock of SPIM. It would be 2~17 SCLK.

DrvSPIM_DmmSetReadMode

Prototype

VOID DrvSPIM_DmmSetReadMode (E_DRVSPIM_READMODE eReadMode);

Description

The function will set SPIM0 to direction memory mapping (DMM) mode and configure its access method in this mode. NOTE: In DMM mode, the SPIM1 is invalid.

Parameters

eReadMode [in]

The SPIM read mode. It could be E_DRVSPIM_STANDARD_READ, E_DRVSPIM_FAST_READ, or E_DRVSPIM_FAST_DUAL_READ.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_DmmGetReadMode

Prototype

E_DRVSPIM_READMODE DrvSPIM_DmmGetReadMode (VOID);

Description

The function will return the SPIM0 read mode. It may be E_DRVSPIM_STANDARD_READ, E_DRVSPIM_FAST_READ or E_DRVSPIM_FAST_DUAL_READ.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

Return the SPIM0 read mode.

DrvSPIM_IsBusy

Prototype

BOOL DrvSPIM_IsBusy (VOID);

Description

The function will return the SPIM0 or SPIM1 busy flag. This function also needs to be call when changing between different working modes, ex. DMM to Command mode.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

TRUE = SPIM0 or SPIM1 is in busy.

DrvSPIM_TriggerDMA

Prototype

```
VOID DrvSPIM_TriggerDMA (
    PUINT8      pu8SpiAddr,
    UINT32      u32DmaLen,
    PUINT8      pu8RamAddr,
    BOOL        bWrite
```

);

Description

The function is used to set the DMA and trigger it to copy the data form or to SPI Flash.

Parameters

pu8SpiAddr [in]

Specify the SPIM0 address for DMA read or DMA write.

u32DmaLen [in]

DMA length, It should be 256 byte alignment.

pu8RamAddr [in]

Specify the SRAM address for DMA read or DMA write.

bWrite [in]

TRUE = Write data to SPI Flash. FALSE = Read data from SPI.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_TerminatDMA

Prototype

VOID DrvSPIM_TerminatDMA (VOID);

Description

This function is used to terminate the DMA mode and return command mode.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_SingleRead

Prototype

UINT32 DrvSPIM_SingleRead (VOID);

Description

This function is used to read the data form SPI bus.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

Return the data got from the SPIM bus.

DrvSPIM_SingleWrite

Prototype

INT32 DrvSPIM_SingleWrite (UINT32 u32data);

Description

This function is used to write the data to SPI bus.

Parameters

u32Data [in]

The data to transfer out through the SPI bus.

Include

Driver/DrvSPIM.h

Return Value

E_SUCCESS	Success
E_DRVSPIM_TIMEOUT	Transfer time out

DrvSPIM_BurstRead

Prototype

INT32 DrvSPIM_BurstRead (PUINT32 pu32buf, INT32 i32burstCnt);

Description

This function is used to read data from SPI bus by burst transfer.

Parameters

pu32Buf [in]

The data buffer to store the data from SPI bus.

i32BurstCnt [in]

The burst transfer count when trigger SPI to transfer. It could be 1~4.

Include

Driver/DrvSPIM.h

Return Value

Return the transfer count, i.e. the data count read from SPI bus

DrvSPIM_BurstWrite

Prototype

INT32 DrvSPIM_BurstWrite (PUINT32 p32buf, INT32 i32burstCnt);

Description

This function is used to write the data to SPI bus by burst transfer.

Parameters

pu32Buf [in]

The data buffer contains the data to write data to SPI bus.

i32BurstCnt [in]

The burst transfer count when trigger SPI to transfer. It could be 1~4.

Include

Driver/DrvSPIM.h

Return Value

Return the transfer count, i.e. the data count write to SPI bus

DrvSPIM_DumpRxRegister

Prototype

INT32 DrvSPIM_DumpRxRegister (PUINT32 pu32Buf, INT32 i32BurstCnt);

Description

This function is used to read the data in the Rx registers.

Parameters

pu32Buf [in]

The data buffer to store the data in the Rx register.

i32BurstCnt [in]

The number of data to read from Rx registers. It could be 1~4.

Include

Driver/DrvSPIM.h

Return Value

Return the data count reading from Rx registers.

DrvSPIM_SetTxRegister

Prototype

INT32 DrvSPIM_SetTxRegister (PUINT32 pu32Buf, INT32 i32BurstCnt);

Description

This function is used to write the data to Tx registers.

Parameters

pu32Buf [in]

The data buffer contains the data to write to Tx registers.

i32BurstCnt [in]

The data count to write to Tx registers. It could be 1~4.

Include

Driver/DrvSPIM.h

Return Value

Return the number of data writing to Tx registers.

DrvSPIM_SetGo

Prototype

VOID DrvSPIM_SetGo (VOID);

Description

This function is used to set GO_BUSY bit to trigger the SPIM.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_SetBitLength

Prototype

INT32 DrvSPIM_SetBitLength (INT32 i32BitLength);

Description

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

Parameters

i32BitLength [in]

The bit length for transfer (1~32 bits)

Include

Driver/DrvSPIM.h

Return Value

E_SUCCESS	Success.
E_DRVSPIM_INVALID_BITLENGTH	The bit length out of range.

DrvSPIM_EnableInt

Prototype

VOID DrvSPIM_EnableInt(PVOID pfnCallbackISR, UINT32 u32UserData);

Description

This function is used to enable the SPIM interrupt and install its callback function.

Parameters

pfnCallbackISR [in]

The SPIM interrupt callback funciton.

u32UserData [in]

The user's data to pass to the callback function.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_DisableInt

Prototype

VOID DrvSPIM_DisableInt(VOID);

Description

This function is used to disable the SPIM interrupt and uninstall its callback function.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_GetIntFlag

Prototype

BOOL DrvSPIM_GetIntFlag(VOID);

Description

This function will return the SPIM interrupt flag. If the flag is TRUE, it means the transfer is done.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

TRUE = There is SPIM interrupt. FALSE = There is no SPIM interrupt.

DrvSPIM_ClearInt

Prototype

VOID DrvSPIM_ClearInt(VOID);

Description

This function will clear the SPIM interrupt flag.

Parameters

None

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_ChipSelect

Prototype

VOID DrvSPIM_ChipSelect(SPIM_CHANNEL channel, BOOL bActive);

Description

This function is used to control the chip select signal of SPIM. The default active level of SPIM is active low, thus the CS would be low when active. NOTE: because the hardware limitation, the SPM0, SPIM1 can't work at the same time.

Parameters

channel [in]

Select SPIM channel. It could be SPIM0 or SPIM1.

bActive [in]

FALSE = Active chip select, TRUE = de-active chip select.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_SetSPIM1CS

Prototype

VOID DrvSPIM_SetSPIM1CS(DRVGPIO_PORT port, INT32 i32Bit);

Description

This function is used to specify a GPIO as the chip select of SPIM1 (master mode).

Parameters

port [in]

The port where contain the IO of the chip select of SPIM1.

i32Bit [in]

The pin of the specified port for chip select.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_GetSPIM1CS

Prototype

VOID DrvSPIM_GetSPIM1CS(DRVGPIO_PORT *port, INT32 *pi32Bit);

Description

This function is used to get the GPIO which is utilized as the chip select of SPIM1 (master mode).

Parameters

port [out]

To return the port where contain the IO of the chip select of SPIM1.

pi32Bit [out]

To return the pin of the port for chip select.

Include

Driver/DrvSPIM.h

Return Value

None

DrvSPIM_GetVersion

Prototype

UINT32

DrvSPIM_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvAIC.h

Return Value

Version number :

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

21. DrvSPIMS Introduction

21.1. SPIMS Introduction

The SPIMS provides one general SPI bus and including master and slave functions. The SPIMS provides to configure the SPI time for maximum flexibility including LSB or MSB first, latch by clock rising or falling and up to 32-bit per transfer. It also provides full-duplex and supports burst transfer to maximum the transfer performance.

21.2. General Feature

- Support SPI master/slave functions
- Full duplex synchronous serial data transfer
- Variable length of transfer word up to 32 bits
- Provide burst mode operation, transmit/receive can be executed up to four times in one transfer
- MSB or LSB first data transfer
- Rx and Tx on both rising or falling edge of serial clock independently

22. DrvSPIMS APIs Specification

22.1. Macro

_DRVSPIMS_STATUS

Prototype

_DRVSPIMS_STATUS

Description

The macro _DRVSPIMS_STATUS can obtain the status of SPIMS

Include

Driver/DrvSPIMS.h

Return Value

None

_DRVSPIMS_CONTROL

Prototype

_DRVSPIMS_CONTROL(val)

Description

The macro_DRVSPIMS_CONTROL can control the protocol of SPIMS.

Parameters

val [in]

Set the control value of SPIMS function.

Include

Driver/DrvSPIMS.h

Return Value

None

_DRVSPIMS_SENDDATA0

Prototype

_DRVSPIMS_SENDDATA0(u32val)

Description

The macro _DRVSPIMS_SENDDATA0 can send the 4-byte data into the Data transmit register 0 for SPIMS.

Parameters

u32val [in]

The 4-byte data to be written into data transmit register 0.

Include

Driver/DrvSPIMS.h

Return Value

None

_DRVSPIMS_SENDDATA1

Prototype

_DRVSPIMS_SENDDATA1(u32val)

Description

The macro _DRVSPIMS_SENDDATA1 can send the 4-byte data into the Data transmit register 1 for SPIMS.

Parameters

u32val [in]

The 4-byte data to be written into data transmit register 1.

Include

Driver/DrvSPIMS.h

Return Value

None

_DRVSPIMS_SENDDATA2

Prototype

_DRVSPIMS_SENDDATA2(u32val)

Description

The macro _DRVSPIMS_SENDDATA2 can send the 4-byte data into the Data transmit register 2 for SPIMS.

Parameters

u32val [in]

The 4-byte data to be written into data transmit register 2.

Include

Driver/DrvSPIMS.h

Return Value

None

_DRVSPIMS_SENDDATA3

Prototype

_DRVSPIMS_SENDDATA3(u32val)

Description

The macro _DRVSPIMS_SENDDATA3 can send the 4-byte data into the Data transmit register 3 for SPIMS.

Parameters

u32val [in]

The 4-byte data to be written into data transmit register 3.

Include

Driver/DrvSPIMS.h

Return Value

None

_DRVSPIMS _GETDATA0

Prototype

DRVSPIMS GETDATA0

Description

The macro _DRVSPIMS_GETDATA0 can read the 4-byte data from the Data receive register 0 for SPIMS.

Include

Driver/DrvSPIMS.h

Return Value

The 4-byte data to be read from the data receive register 0

_DRVSPIMS_GETDATA1

Prototype

_DRVSPIMS_GETDATA1

Description

The macro _DRVSPIMS_GETDATA1 can read the 4-byte data from the Data receive register 1 for SPIMS.

Include

Driver/DrvSPIMS.h

Return Value

The 4-byte data to be read from the data receive register 1

_DRVSPIMS_GETDATA2

Prototype

_DRVSPIMS_GETDATA2

Description

The macro _DRVSPIMS_GETDATA2 can read the 4-byte data from the Data receive register 2 for SPIMS.

Include

Driver/DrvSPIMS.h

Return Value

The 4-byte data to be read from the data receive register 2

_DRVSPIMS_GETDATA3

Prototype

_DRVSPIMS_GETDATA3

Description

The macro _DRVSPIMS_GETDATA3 can read the 4-byte data from the Data receive register 3 for SPIMS.

Include

Driver/DrvSPIMS.h

Return Value

The 4-byte data to be read from the data receive register 3

22.2. Functions

DrvSPIMS_Init

Prototype

INT32 DrvSPIMS_Init(VOID);

Description

This function is used to initial SPIMS function.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

E_SUCCESS	Success
E_DRVSPIMS_ERR_INIT	Initial failed.

DrvSPIMS_Open

Prototype

```
ERRCODE DrvSPIMS_Open(
   E_SPIMS_MODE eMode,
   E_SPIMS_TRANS_TYPE eType,
   INT32 i32bitLength
);
```

Description

This function is used to open SPIMS module. It decides the SPIMS to work on master or slave mode, SPI bus timing and bit length per transfer. The SPI clock frequency is default to be 500 kHz and can be set by calling DrvSPIMS_SetClock().

Parameters

eMode [in]

To work in Master or Slave mode.

eType [in]

Transfer types, i.e the bus timing. it could be SPIMS_TYPE0~SPIMS_TYPE7.

i32bitLength [in]

bit length per transaction.

Include

Driver/DrvSPIMS.h

Return Value

E_SUCCESS	Success.
E_DRVSPIMS_ERR_BIT_LENGTH	The bit length is out of range.
E_DRVSPIMS_ERR_BUSY	The SPIMS is in busy.

DrvSPIMS_Close

Prototype

VOID DrvSPIMS_Close (VOID);

Description

This function is used to close SPIMS module.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_SetEndian

Prototype

VOID DrvSPIMS_SetEndian (E_SPIMS_ENDIAN eEndian);

Description

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

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_SetBitLength

Prototype

ERRCODE DrvSPIMS_SetBitLength(INT32 i32BitLength);

Description

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

Parameters

i32BitLength [in]

The bit length for transfer (1~32 bits).

Include

Driver/DrvSPIMS.h

Return Value

E_SUCCESS	Success.
E_DRVSPIMS_ERR_BIT_LENGTH	The bit length is out of range.

DrvSPIMS_EnableAutoCS

Prototype

VOID DrvSPIMS_EnableAutoCS (E_SPIMS_POLAR ePolar);

Description

This function is used to enable the auto chip select function and configure the active level of chip select signal. The auto chip select means the SPIMS will active the chip select I/O when transmitting data and de-active the chip select I/O after one transfer is finished. For some devices, the chip select could be active for many transfers and user should disable the auto chip select function to control the chip select I/O manually for these devices. Furthermore, the chip select I/O could be configured as low active or high active. The SPIMS will set the chip select I/O according to the active level during transfer when auto chip select function is enabled.

Parameters

ePolar [in]

To configure the CS as ACTIVE_LOW or ACTIVE_HIGH.

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_DisableAutoCS

Prototype

VOID DrvSPIMS_DisableAutoCS (VOID);

Description

This function is used to disable the automatic chip selection function. If it is necessary to keep chip select I/O low or high when do transfers, it is necessary to disable the automatic chip selection function and control the chip select I/O manually.

Include

Driver/DrvSPIMS.h

Return Value

Non

DrvSPIMS_SetCS

Prototype

VOID DrvSPIMS_SetCS(VOID);

Description

This function is used to set CS as high. It is valid only when automatic chip selection function is disabled.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_ClrCS

Prototype

VOID DrvSPIMS_ClrCS(VOID);

Description

This function is used to set CS as low. It is valid only when automatic chip selection function is disabled

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_Busy

Prototype

BOOL DrvSPIMS_Busy(VOID);

Description

This function is used to return the SPI busy status.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

TURE = The SPIMS is in busy. FALSE = The SPIMS is not in busy.

DrvSPIMS_BurstTransfer

Prototype

ERRCODE DrvSPIMS_BurstTransfer(INT32 i32BurstCnt, INT32 i32Interval);

Description

This function is used to configure the burst transfer settings.

Parameters

i32BurstCnt [in]

Set the transactions per transfer. It could be 1~4.

i32Interval [in]

The delay clocks between successive transactions. It could be 2~17.

Include

Driver/DrvSPIMS.h

Return Value

E_SUCCESS	Success.
E_DRVSPIMS_ERR_BURST_CNT	The burst count is out of range.
E_DRVSPIMS_ERR_TRANSMIT_INTERVAL	The interval is out of range.

DrvSPIMS_SetClock

Prototype

UINT32 DrvSPIMS_SetClock(UINT32 u32Clock);

Description

This function is used to configure the SPI bus clock. NOTE: The actual clock may be different to the target SPI clock due to hardware limitation.

Parameters

u32Clock [in]

The target SPI bus clock frequency.

Include

Driver/DrvSPIMS.h

Return Value

Return the divisor. (SPI clock is calculated by APB clock / divisor)

DrvSPIMS_GetClock

Prototype

UINT32 DrvSPIMS_SetClock(UINT32 u32Clock);

Description

This function is used to get the SPI bus clock.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

The current SPI bus clock frequency in Hz.

DrvSPIMS_EnableInt

Prototype

```
VOID DrvSPIMS_EnableInt(
    PFN_DRVSPIMS_CALLBACK    pfnCallback,
    UINT32                   u32UserData
);
```

Description

This function is used to enable the SPI interrupt and install the callback function.

Parameters

pfnCallback [in]

The callback function to be called when interrupt. The prototype of the callback function is: INT32 (*PFN_DRVSPIMS_CALLBACK)(UINT32 userData);

u32UserData [in]

The parameter which needs to be passed to the callback.

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_DisableInt

Prototype

VOID DrvSPIMS_DisableInt(VOID);

Description

This function is used to disable the SPI interrupt.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_SingleRead

Prototype

UINT32 DrvSPIMS_SingleRead(VOID);

Description

This function is used to read the data form SPI bus.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

The data got from the SPI bus.

DrvSPIMS_SingleWrite

Prototype

INT32 DrvSPIMS_SingleWrite (UINT32 u32Data);

Description

This function is used to write the data to SPI bus.

Parameters

u32Data [in]

The data to transfer out through the SPI bus.

Include

Driver/DrvSPIMS.h

Return Value

The data count to write to SPI bus. If write time-out, it will return 0.

DrvSPIMS_BurstRead

Prototype

INT32 DrvSPIMS_BurstRead (PUINT32 pu32Buf, INT32 i32BurstCnt);

Description

This function is used to read data from SPI bus by burst transfer.

Parameters

pu32Buf [in]

The data buffer to store the data from SPI bus.

i32BurstCnt [in]

The burst transfer count when trigger SPI to transfer. It could be 0~4.

Include

Driver/DrvSPIMS.h

Return Value

Return the transfer count, i.e. the data count read from SPI bus

DrvSPIMS_BurstWrite

Prototype

INT32 DrvSPIMS_BurstWrite (PUINT32 p32Buf, INT32 i32BurstCnt);

Description

This function is used to write the data to SPI bus by burst transfer.

Parameters

pu32Buf [in]

The data buffer contains the data to write data to SPI bus.

i32BurstCnt [in]

The burst transfer count when trigger SPI to transfer. It could be 0~4.

Include

Driver/DrvSPIMS.h

Return Value

Return the transfer count, i.e. the data count write to SPI bus

DrvSPIMS_DumpRxRegister

Prototype

INT32 DrvSPIMS_DumpRxRegister (PUINT32 pu32Buf, INT32 i32BurstCnt);

Description

This function is used to read the data in the Rx registers.

Parameters

pu32Buf [in]

The data buffer to store the data in the Rx register.

i32BurstCnt [in]

The number of data to read from Rx registers. It could be 0~4.

Include

Driver/DrvSPIMS.h

Return Value

Return the data count reading from Rx registers.

DrvSPIMS_SetTxRegister

Prototype

INT32 DrvSPIMS_SetTxRegister (PUINT32 pu32Buf, INT32 i32BurstCnt);

Description

This function is used to write the data to Tx registers.

Parameters

pu32Buf [in]

The data buffer contains the data to write to Tx registers.

i32BurstCnt [in]

The data count to write to Tx registers. It could be 0~4.

Include

Driver/DrvSPIMS.h

Return Value

Return the number of data writing to Tx registers.

DrvSPIMS_SetGo

Prototype

VOID DrvSPIMS_SetGo (VOID);

Description

This function is used to set GO_BUSY bit to trigger the SPIM.

Parameters

None

Include

Driver/DrvSPIMS.h

Return Value

None

DrvSPIMS_GetVersion

Prototype

UINT32 DrvSPIMS_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvSPIMS.h

Return Value

Version number:

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

23. DrvSYS Introduction

23.1. System Introduction

The System Manager has the following functions,

- System memory map
- Data bus connection with external memory
- Product identifier register
- Bus arbitration
- PLL module
- Clock divider setting for some special functions needing clocks except system bus clock
- Clock select, enable, and power saving control register
- Power-On setting reading and alternating

23.2. System Feature

- Support power down, idle, memory idle, and wake up.
- Get and set system clock.
- Bus arbitration control.

24. DrvSYS APIs Specification

24.1. Constant Definition

24.1.1. IP Clock Switch

Table 24-1: IP clocks switch

Name	Mask Value	Description
E_DRVSYS_TMR_CLK	0x01	Timer clock switch
E_DRVSYS_WDT_CLK	0x02	Watch dog clock switch
E_DRVSYS_RTC_CLK	0x04	RTC clock switch
E_DRVSYS_UART0_CLK	0x08	UART0 clock switch
E_DRVSYS_UART1_CLK	0x10	UART1 cock switch
E_DRVSYS_PWM_CLK	0x20	PWM clock switch
E_DRVSYS_I2C_CLK	0x40	I2C clock switch
E_DRVSYS_SPIMS_CLK	0x100	SPIMS clock switch
E_DRVSYS_ADC_CLK	0x200	ADC clock switch
E_DRVSYS_CPU_CLK	0x10000	CPU clock switch
E_DRVSYS_APB_CLK	0x20000	APB clock switch
E_DRVSYS_USBD_CLK	0x400000	USB device clock switch
E_DRVSYS_SPIM_CLK	0x800000	SPIM clock switch
E_DRVSYS_APU_CLK	0x1000000	APU clock switch

24.1.2. IP Reset

Table 24-2: IP reset

Name	Mask Value	Description
E_DRVSYS_UART0_RST	0x01	UART0 reset
E_DRVSYS_UART1_RST	0x02	UART1 reset
E_DRVSYS_TMR_RST	0x20	Timer and watchdog controller reset
E_DRVSYS_PWM_RST	0x100	PWM controller reset
E_DRVSYS_I2C_RST	0x200	I2C controller reset
E_DRVSYS_SPIM_RST	0x400	SPIM0 and SPIM1 controller reset
E_DRVSYS_UDC_RST	0x800	USB device controller reset
E_DRVSYS_APU_RST	0x10000	APU controller reset
E_DRVSYS_SRAM_RST	0x2000000	SRAM controller reset
E_DRVSYS_GPIO_RST	0x8000000	GPIO reset
E_DRVSYS_ADC_RST	0x10000000	ADC controller reset
E_DRVSYS_SPIMS_RST	0x40000000	SPI master/slave reset

24.2. Functions

DrvSYS_Open

Prototype

```
ERRCODE
DrvSYS_Open (
    UINT32 u32ExtClockKHz,
    UINT32 u32PllClockKHz
);
```

Description

Open the system manager function and set the external clock source frequency and the target PLL clock. The PLL clock will be close to the target frequency if possible.

Parameter

u32ExtClockKHz [in]

Specify the frequency of external crystal. If the frequency is 12MHz, the parameter should be set as “12000”.

u32PllClockKHz [in]

Specify the target frequency of PLL. If the frequency is 144MHz, the parameter should be set as “144000”.

Include

Driver/DrvSYS.h

Return Value

E_SUCCESS: Successful.

Note

1. This API must be called before other APIs.
2. Here assume that the crystal with frequency higher than 9MHz is used.

DrvSYS_SetSystemClockSource

Prototype

```
VOID
DrvSYS_SetSystemClockSource(
    E_DRVSYS_SRC_CLK eSysClkSrc
);
```

Description

To set the system clock source. The system clock source may be from external crystal or PLL.

Parameter

eSysClkSrc [in]

Select the system clock source. It could be E_DRVSYS_SYS_EXT or E_DRVSYS_SYS_PLL.

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_GetSystemClockSource

Prototype

```
E_DRVSYS_SRC_CLK
DrvSYS_GetSystemClockSource(VOID);
```

Description

Get system clock source. It may be E_DRVSYS_SYS_EXT or E_DRVSYS_SYS_PLL.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The system clock source. It could be E_DRVSYS_SYS_EXT or E_DRVSYS_SYS_PLL.

DrvSYS_GetEXTClock

Prototype

UINT32

DrvSYS_GetEXTClock(VOID);

Description

Get external clock frequency. The clock UNIT is in kHz.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The external clock frequency.

DrvSYS_GetPLLClock

Prototype

UINT32

DrvSYS_GetPLLClock(VOID);

Description

Get output frequency of PLL clock. Unit: KHz.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

PLL works frequency. Unit: KHz.

DrvSYS_SetAHBClock / DrvSYS_SetCPUClock

Prototype

```
UINT32
DrvSYS_SetAHBClock(
   UINT32 u32AhbClockKHz
);
UINT32
DrvSYS_SetCPUClock(
   UINT32 u32CPUClockKHz
);
```

Description

Set the AHB (system) or CPU clock base on the external clock or internal PLL that is specified in function-DrvSYS_Open. The CPU clock is always equal to AHB clock. And the source clock of AHB or CPU clock comes from PLL. The real AHB clock will be equal to or less than the specified clock. The root cause is caused by the hardware divider is limited. There is a default divider divide by 2 that cascades between PLL and AHB.

Parameter

u32AhbClockKHz / u32CPUClockKHz [in]

Target AHB/CPU works frequency.

Include

Driver/DrvSYS.h

Return Value

E_SUCCESS: Successful.

E_DRVSYS_AHB_SRC_ERR: Error code.

DrvSYS_SetAPBClock

Prototype

```
UINT32
DrvSYS_SetAPBClock(
   UINT32 u32ApbClockKHz
);
```

Description

Set the APB clock base on the AHB clock. The real APB clock will be equal to or less than the specified APB clock. The root cause is caused by the hardware divider is limited. The source clock of APB clock comes from AHB.

Parameter

u32APBClockKHz [in]

Target APB works frequency.

Include

Driver/DrvSYS.h

Return Value

E_SUCCESS: Successful.

E_DRVSYS_APB_SRC_ERR: Error code.

DrvSYS_GetAHBClock / DrvSYS_GetCPUClock

Prototype

UINT32

DrvSYS_GetAHBClock(VOID);

UINT32

DrvSYS_GetCPUClock(VOID);

Description

Get the AHB or CPU clock base on the AHB clock. The AHB clock is always equal to CPU clock.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

AHB or CPU works frequency.

DrvSYS_GetAPBClock

Prototype

UINT32

DrvSYS_GetAPBClock(VOID);

Description

Get the APB clock. The APB clock is always equal to or lower than AHB/CPU clock.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

APB works frequency.

DrvSYS_SetIPClock

Prototype

```
VOID
DrvSYS_SetIPClock(
    E_DRVSYS_IP_CLK_EN eIpClkEn,
    BOOL bIsEnable
);
```

Description

Enable or disable each IP's clock.

Parameter

eIpClkEn [in]

Enumeration for IP clock switch, reference the IP Clock Switch

bIsEnable [in]

TRUE: Enable IP clock.

FALSE: Disable IP clock.

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_GetIPClock

Prototype

```
BOOL
DrvSYS_GetIPClock(
```

```
    E_DRVSYS_IP_CLK_EN eIpClkEn,
);
```

Description

Get the status of IP's clock switch.

Parameter

eIpClkEn [in]

Enumeration for IP clock switch, reference the IP Clock Switch

Include

Driver/DrvSYS.h

Return Value

TRUE: IP clock is enabled.

FALSE: IP clock is disabled.

DrvSYS_ResetIP

Prototype

```
VOID
DrvSYS_ResetIP(
    E_DRVSYS_IP_RST eIpRst
);
```

Description

Reset IP.

Parameter

eIpRst [in]

Enumeration for IP reset, reference the IP Reset

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_ResetCPU

Prototype

```
VOID
DrvSYS_ResetCPU(VOID);
```

Description

To reset the CPU only without reset any other IP module.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_RoughDelay

Prototype

```
VOID
DrvSYS_RoughDelay (
    UINT32 u32Delay
);
```

Description

A roughly software delay, the delay unit is micro second.

Parameter

u32Delay [in]

Delay loop counter. Unit is micro second.

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_SetSRAMBase

Prototype

```
ERRCODE
DrvSYS_SetSRAMBase (
    UINT32      u32Bank,
    UINT32      u32TagAddr,
    BOOL        bValid
);
```

Description

Set the mapping address and valid flag to specified RAM bank

Parameters

u32Bank [in]

Specify the RAM bank number to get the mapping address. The bank number is from 0 to 15.

u32TagAddr [in]

Mapping address. The value is from 0 to 0x1FFFF800. The values must be 2KB alignment.

bValid [in]

TRUE: The specify RAM bank number is valid

FALSE: The specify RAM bank number is invalid

Include

Driver/DrvSYS.h

Return Value

E_SUCCESS: Succeed

<0: Error code. Reference to ARM SDS Error Code Collection Document

Note

All banks should not have same address space.

DrvSYS_GetSRAMBase

Prototype

```
ERRCODE
DrvSYS_GetSRAMBase (
    UINT32     u32Bank,
    PUINT32    pu32TagAddr,
    PBOOL      pbValid
);
```

Description

Get the mapping address and valid flag for specified RAM bank

Parameters

u32Bank [in]

Specify the RAM bank number to get the mapping address. The bank number is from 0 to 15.

pu32TagAddr [out]

Mapping address. The value is from 0 to 0x1FFFF800. The values must be 2KB alignment.

pbValid [out]

The specify RAM bank number is valid or invalid.

Include

Driver/DrvSYS.h

Return Value

E_SUCCESS: Succeed

<0: Error code. Reference to ARM SDS Error Code Collection Document

Note

All banks should not have same address space.

DrvSYS_SetBusPriority

Prototype

```
VOID
DrvSYS_SetBusPriority (
  BOOL      bIsAhbPriority,
  );
```

Description

Set the bus priority.

Parameters

bisAhbPriority [in]

FALSE: AHB master uses fixed priority mode.

TRUE: AHB master uses rotated priority mode.

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_GetBusPriority

Prototype

```
BOOL
DrvSYS_GetBusPriority (
  VOID
```

);

Description

Get the bus priority.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

FALSE: AHB master uses fixed priority mode.

TRUE: AHB master uses rotated priority mode.

DrvSYS_SetHighSpeedInt

Prototype

VOID

DrvSYS_SetHighSpeedInt (

BOOL bIsHighSpeedInt,

);

Description

It can be used to reduce the interrupt latency in a real-time system, set this bit, the CPU will has the highest AHB priority.

Parameters

bIsHighSpeedInt [in]

FALSE: CPU has the default priority.

TRUE: CPU has the highest AHB priority

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_GetHighSpeedInt

Prototype

BOOL

DrvSYS_GetHighSpeedInt (

VOID

);

Description

Get the CPU priority.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

FALSE: CPU has the default priority.

TRUE: CPU has the highest AHB priority.

DrvSYS_EnableIdle

Prototype

VOID

DrvSYS_EnableIdle(VOID);

Description

Set system enter idle mode. CPU clock will be stop. Peripheral clock keep running. Timers or GPIO interrupt can wake up the system.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_EnablePowerDown

Prototype

VOID

DrvSYS_PowerDown (VOID);

Description

Set system enters power down mode. CPU, AHB, APB clock will be stop. RTC clock still keep running. GPIO and RTC interrupts can wake up the system.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_GetVersion

Prototype

UINT32

DrvSYS_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvSYS.h

Return Value

Version number :

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

25. DrvTimer Introduction

25.1. Timer Introduction

The timer module includes two channels, TIMER0~TIMER1, 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 possesses features such as adjustable resolution, programmable counting period, and detailed information. The timer can generate an interrupt signal upon timeout, or provide the current value of count during operation. The timer module also provides the watchdog timer function to handle the system crash issue.

25.2. Timer Feature

The timer processing unit includes following features:

- Compliant with the AMBA APB
- Each channel with a 8-bit prescale counter/32-bit counter and an interrupt request signal.
- Independent clock source for each channel(TCLK0,TCLK1)
- Maximum uninterrupted time = (1 / 25 MHz) * (2^8) * (2^32 - 1) when TCLK = 25 MHz

26. DrvTimer APIs Specification

26.1. Macro

_DRVTIMER_ENABLE

Prototype

```
VOID    _DRVTIMER_ENABLE (
    UINT8 u8Timer
);
```

Description

Enable Timer to start.

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

None

_DRVTIMER_DISABLE

Prototype

```
VOID    _DRVTIMER_DISABLE (
    UINT8 u8Timer
);
```

Description

Disable Timer to stop.

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

None

_DRVTIMER_SET_DEBUG

Prototype

```
VOID    _DRVTIMER_SET_DEBUG (
  UINT8     u8Timer
  BOOL      bDebug
);
```

Description

Control timer clock in the debug mode.

Parameter

u8Timer [in]

Specify timer 0/1.

bDebug [in]

FALSE: Timer clock pauses in the debug mode

TRUE: Timer clock always runs in the normal or debug mode

Include

Driver/DrvTimer.h

Return Value

None

_DRVTIMER_SET_FREQUENCY

Prototype

```
VOID    _DRVTIMER_SET_ FREQUENCY (
  UINT8     u8Timer
  UINT32    u32Initial,
  UINT32    u32Divider
);
```

Description

Set the frequency of timer.

The output frequency = the selected clock frequency / u32Initial / (u32Divider+1)

Parameter

u8Timer [in]

Specify timer 0/1.

u32Initial [in]

0x0 ~ 0xFFFFFFFF: Specify initial count

u32Divider [in]

0x0 ~ 0x255: Specify divider

Include

Driver/DrvTimer.h

Return Value

None

_DRVTIMER_SET_MODE

Prototype

```
VOID    _DRVTIMER_SET_ MODE (
   UINT8       u8Timer
   UINT32      u32Mode
);
```

Description

Set timer operating mode

Parameter

u8Timer [in]

Specify timer 0/1.

u8Mode [in]

DRVTIMER_ONESHOT_MODE: One-shot mode

DRVTIMER_PERIOD_MODE: Period mode

DRVTIMER_TOGGLE_MODE: Toggle mode

DRVTIMER_UNINTERRUPTED_MODE: Uninterrupted mode

Include

Driver/DrvTimer.h

Return Value

None

_DRVTIMER_GET_MODE

Prototype

```
UINT8  _DRVTIMER_GET_ MODE (
   UINT8       u8Timer
);
```

Description

Get timer operating mode

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

The timer operating mode

DRVTIMER_ONESHOT_MODE: One-shot mode

DRVTIMER_PERIOD_MODE: Period mode

DRVTIMER_TOGGLE_MODE: Toggle mode

DRVTIMER_UNINTERRUPTED_MODE: Uninterrupted mode

_DRVTIMER_GETPRESCALE

Prototype

```
UINT8  _DRVTIMER_GETPRESCALE (
          UINT8       u8Timer
);
```

Description

Set the prescale value of timer.

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

The prescale value

_DRVTIMER_GETINITIALCOUNT

Prototype

```
UINT32        _DRVTIMER_GETINITIALCOUNT (
  UINT8       u8Timer
);
```

Description

Get timer initial counter value

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

The timer initial counter value

_DRVTIMER_ISENABLED

Prototype

```
BOOL   _DRVTIMER_ISENABLED (
  UINT8       u8Timer
);
```

Description

Get the enable status of timer

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

The timer enable status

_DRVTIMER_READ_UNDERFLOW_FLAG

Prototype

```
BOOL   _DRVTIMER_READ_UNDERFLOW_FLAG (
   UINT8       u8Timer
);
```

Description

Read the underflow flag of timer

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

The timer underflow flag

_DRVTIMER_CLEAR_UNDERFLOW_FLAG

Prototype

```
VOID    _DRVTIMER_CLEAR_UNDERFLOW_FLAG (
   UINT8       u8Timer
);
```

Description

Clear the underflow flag of timer.

Parameter

u8Timer [in]

Specify timer 0/1.

Include

Driver/DrvTimer.h

Return Value

None

26.2. Function

DrvTIMER_GetStatus

Prototype

UINT32 DrvTIMER_GetStatus(VOID);

Description

This function is used to return read TIMER TISR register to get timer interrupt status.

Parameter

None

Include

Driver/DrvTimer.h

Return Value

The data of register TISR.

DrvTIMER_SetTimerEvent

Prototype

```
INT32 DrvTIMER_SetTimerEvent(
    TIMER_CHANNEL    channel,
    UINT32           uTimeTick,
    PVOID            pvFun ,
    UINT32           parameter
);
```

Description

This function is used to install a event to timer0 or timer1.

Parameter

channel [in]

TIMER0/TIMER1.

uTimeTick [in]

The tick value which want to execute event.

pvFun [in]

The event function pointer.

parameter [in]

An parameter,was defined by user,which send to callback function.

Include

Driver/DrvTimer.h

Return Value

The event number which contains this event

DrvTIMER_ClearTimerEvent

Prototype

```
VOID DrvTIMER_ClearTimerEvent(
    TIMER_CHANNEL    channel,
    UINT32           uTimeEventNo
);
```

Description

This function is used to remove an installed event in timer0 or timer1.

Parameter

channel [in]

TIMER0/TIMER1.

uTimeEventNo [in]

EVENT No. it could be 0 ~ TIMER_EVENT_COUNT-1.

Include

Driver/DrvTimer.h

Return Value

None

DrvWDT_ResetCount

Prototype

```
VOID DrvWDT_ResetCount(VOID);
```

Description

This function is used to reset WDT Tick to avoid time-out to restart system.

Parameter

None

Include

Driver/DrvTimer.h

Return Value

None

DrvWDT_SetEvent

Prototype

VOID DrvWDT_SetEvent(PVOID pvFun);

Description

This function is used to install a callback function in WDT.

Parameter

pvFun [in]

The event function pointer.

Include

Driver/DrvTimer.h

Return Value

None

DrvTIMER_ResetTicks

Prototype

INT32 DrvTIMER_ResetTicks(TIMER_CHANNEL channel);

Description

This function is used to reset TIMER Tick..

Parameter

channel [in]

TIMER channel TIMER0 / TIMER1.

Include

Driver/DrvTimer.h

Return Value

E_SUCCESS	Success
E_DRVTIMER_CHANNEL	Unsupported timer channel

DrvTIMER_Init

Prototype

VOID DrvTIMER_Init(VOID);

Description

This function is used to initial TIMER when system boot up.

Parameter

None

Include

Driver/DrvTimer.h

Return Value

None

DrvTIMER_Open

Prototype

```
INT32 DrvTIMER_Open(
    TIMER_CHANNEL    channel,
    UINT32           uTicksPerSecond,
    TIMER_OPMODE     mode
);
```

Description

This function is used to set and start TIMER.

Parameter

channel [in]

TIMER Channel TIMER0/TIMER1.

uTickPerSecond [in]

Tick per second.

Mode [in]

Operation Mode One-Shot / Periodic / Toggle. It could be ONESHOT_MODE, PERIODIC_MODE or TOGGLE_MODE.

Include

Driver/DrvTimer.h

Return Value

E_SUCCESS	Success.
E_DRVTIMER_CMD	Command error.
E_DRVTIMER_EIO	Timer is not initialized by DrvTIMER_Init().

DrvTIMER_GetTicks

Prototype

UINT32 DrvTIMER_GetTicks(TIMER_CHANNEL channel);

Description

This function is used to return Timer ticks.

Parameter

channel [in]

It could be TIMER0/TIMER1.

Include

Driver/DrvTimer.h

Return Value

Return the current ticks of TIMER0 or TIMER1.

DrvTIMER_Delay

Prototype

VOID DrvTIMER_Delay (UINT32 uTicks);

Description

This function is used to set a delay time if necessary. The TIMER0 is used in this delay function thus it needs to be opened and initialized first.

Parameter

uTicks [in]

The delay time, and it is depend on Timer CLK.

Include

Driver/DrvTimer.h

Return Value

None

DrvTIMER_Ioctl

Prototype

```
INT32 DrvTIMER_Ioctl(
    TIMER_CHANNEL    channel,
    TIMER_CMD        uCmd,
    UINT32           uArg1,
    PVOID            pvFun
```

);

Description

To process the general control of timer. The following table listed the command, parameters and relative descriptions.

Command	Argument	Description
TIMER_IOC_START_COUNT	Not used	Start timer counter
TIMER_IOC_STOP_COUNT	Not used	Stop timer counter
TIMER_IOC_ENABLE_INT	Not used	Enable the timer interrupt
TIMER_IOC_DISABLE_INT	Not used	Disable the timer interrupt
TIMER_IOC_RESET_TIMER	Not used	Reset timer counter
TIMER_IOC_SET_PRESCALE	uArg1	uArg1 is the pre-scale value for timer counter. The value could be 0 ~ 255 and resulting the counter clock to be divided by 1 ~ 256.
TIMER_IOC_SET_INITIAL_COUNT	uArg1	This command is used to specify the initial value of timer counter. Due to the timer counter is 16-bit, the uArg1 could be 0 ~ 65535 and the timer counter will down count to 0 from the initial count value when timer start.

Parameter

channel [in]

It could be TIMER0/TIMER1.

uCmd [in]

The I/O control commands, e.x. TIMER_IOC_START_COUNT.

uArg1 [in]

The first parameter for specified command.

pvFun [in]

The event function pointer.

Include

Driver/DrvTimer.h

Return Value

E_SUCCESS	Success.
E_DRVTIMER_CMD	Invalid command.

DrvTIMER_Close

Prototype

INT32 DrvTIMER_Close(TIMER_CHANNEL channel);

Description

The function is used to disable timer.

Parameter

channel [in]

It could be TIMER0 or TIMER1.

Include

Driver/DrvTimer.h

Return Value

E_SUCCESS	Success.
E_DRVTIMER_CMD	Invalid timer channel.

DrvWDT_Init

Prototype

INT32 DrvWDT_Init(VOID);

Description

The function is used to initial Watch Dog Timer.

Parameter

None

Include

Driver/DrvTimer.h

Return Value

None

DrvWDT_Open

Prototype

INT32 DrvWDT_Open(INT32 hander ,WDT_INTERVAL level);

Description

The function is used to set WDT Interval and Star WDT Timer to count.

Parameter

hander [in]

Reserved.

level [in]

WDT time-out level. It could be LEVEL1, LEVEL2, LEVEL3 or LEVEL4.

Include

Driver/DrvTimer.h

Return Value

E_SUCESS	Success

DrvWDT_Ioctl

Prototype

INT32 DrvWDT_Ioctl(INT32 hander ,WDT_CMD uCmd , UINT32 uArg1);

Description

The function of Watching Dog Timer I/O Control API.

Parameter

hander [in]

Reserved.

uCmd [in]

WDT IOCTL command.

uArg1 [in]

First argument of the command.

Include

Driver/DrvTimer.h

Return Value

E_SUCCESS	Success
E_DRVTIMER_CMD	Invalid I/O command

DrvWDT_Close

Prototype

VOID DrvWDT_Close(VOID);

Description

The function is used to Stop Wathch Dog Timer and Disable WDT Interrupt.

Parameter

None

Include

Driver/DrvTimer.h

Return Value

None

DrvTimer_GetVersion

Prototype

UINT32

DrvTimer_GetVersion (VOID);

Description

Return the current version number of driver.

Include

Driver/DrvTimer.h

Return Value

Version number :

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

27. DrvUART Introduction

27.1. 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.

27.2. UART Feature

The UART includes following features:

- 64 byte/16 byte entry FIFOs for received and transmitted data payloads
- Full Modem control functions (CTS, RTS, DSR, DTR, RI and DCD) are supported (only CTS/RTS (low-active) can be used in this version).
- Programmable baud-rate generator that allows the internal clock to be divided by 2 to (2^16 + 1) to generate an internal 16X clock.
- Full set of MODEM control functions (CTS, RTS, DSR, DTR, RI and DCD).
- 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

28. DrvUART APIs Specification

28.1. Constant Definition

Name	Value	Description
DRVUART_PORT0	0x000	UART port 0
DRVUART_PORT1	0x100	UART port 1
DRVUART_MOSINT	0x8	MODEM Interrupt
DRVUART_RLSNT	0x4	Receive Line Interrupt
DRVUART_THREINT	0x2	Transmit Holding Register Empty Interrupt
DRVUART_RDAINT	0x1	Receive Data Available Interrupt and Time-out Interrupt
DRVUART_IIR_RLS	0x06	Receive Line Status Interrupt
DRVUART_IIR_RDA	0x04	Receive Data Available Interrupt
DRVUART_IIR_TOUT	0x0C	Time-out Interrupt
DRVUART_IIR_THRE	0x02	Transmit Holding Register Empty Interrupt
DRVUART_IIR_MOS	0x00	MODEM Status Interrupt
DRVUART_TOUTINT	0x30	Time-out Interrupt.
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.
DRVUART_PARITY_EVEN	0x18	Even parity enable
DRVUART_PARITY_ODD	0x08	Odd parity enable
DRVUART_PARITY_NONE	0x00	None parity
DRVUART_PARITY_MARK	0x28	Parity mask
DRVUART_PARITY_SPACE	0x38	Parity space
DRVUART_STOPBITS_1	0x000	Number of stop bit: Stop bit length is 1 bit.
DRVUART_STOPBITS_1_5	0x4	Number of stop bit: Stop bit length is 1.5 bit when character length is 5 bits.

DRVUART_STOPBITS_2	0x4	Number of stop bit: Stop bit length is 2 bit when character length is 6, 7 or 8 bits.
DRVUART_FIFO_1BYTES	0x00	RX FIFO interrupt trigger level is 1 byte
DRVUART_FIFO_4BYTES	0x10	RX FIFO interrupt trigger level is 4 bytes
DRVUART_FIFO_8BYTES	0x20	RX FIFO interrupt trigger level is 8 bytes
DRVUART_FIFO_14BYTES	0x30	RX FIFO interrupt trigger level is 14 bytes
DRVUART_FIFO_30BYTES	0x40	RX FIFO interrupt trigger level is 30 bytes
DRVUART_FIFO_46BYTES	0x50	RX FIFO interrupt trigger level is 46 bytes
DRVUART_FIFO_62BYTES	0x60	RX FIFO interrupt trigger level is 62 bytes
DRVUART_CLKSRC_EXT	0x00	Clock source from crystal in
DRVUART_CLKSRC_PLL	0x40	Clock source from divided MPLL clock
DRVUART_CLKSRC_PLL_DIV2	0x80	Clock source from divided MPLL clock / 2

28.2. Macro

_DRVUART_RECEIVEBYTE

Prototype

UINT8

_DRVUART_RECEIVEBYTE (UINT16 u16Port);

Description

Reads 1 byte from the receive FIFO of UART.

Parameter

u16Port [in]

Specify UART0/UART1

Include

Driver/DrvUART.h

Return Value

The data in Rx FIFO.

_DRVUART_SENDBYTE

Prototype

VOID _DRVUART_SENDBYTE (

UINT16 u16Port

BYTE byData

);

Description

Write 1 byte into the transmission FIFO of UART.

Parameter

u16Port [in]

Specify UART0/UART1

byData [in]

The byte date we want to send.

Include

Driver/DrvUART.h

Return Value

None

_DRVUART_SET_DIVIDER

Prototype

```
VOID    _DRVUART_SET_DIVIDER (
  UINT16     u16Port
  UINT16     u16Divider
);
```

Description

Set the baud rate divider.

Parameter

u16Port [in]

Specify UART0/UART1

u16Divider [in]

The baud rate divider.

Include

Driver/DrvUART.h

Return Value

None

Notes

Baud Rate = Crystal Clock / {16 * [Divisor + 2]}

_DRVUART_GET_DIVIDER

Prototype

UINT16

_DRVUART_SET_DIVIDER (UINT16 u16Port);

Description

Get the baud rate divider.

Parameter

u8Port [in]

Specify UART0/UART1

Return Value

The baud rate divider.

Include

Driver/DrvUART.h

Return Value

None

_DRVUART_INTERRUPTID

Prototype

UINT8

_DRVUART_INTERRUPRID (UINT16 u16Port);

Description

The interrupt identification for the interrupt method of UART. The interrupt identification is from the specification of UART. Please refer to the specification for more details.

Parameter

u16Port [in]

Specify UART0/UART1

Include

Driver/DrvUART.h

Return Value

0: MODEM status

2: The transmission FIFO is available to be written

4: The receive FIFO is available to be read

6: Receiver Line status

12: The receive FIFO time out to be read

_DRVUART_RECEIVEAVAILABLE

Prototype

UINT8

_DRVUART_RECEIVEAVAILABLE (UINT16 u16Port);

Description

Checks the status of UART receive FIFO. If the FIFO is available; user can read the data of UART receive FIFO.

Parameter

u16Port [in]

Specify UART0/UART1

Include

Driver/DrvUART.h

Return Value

1: The receive FIFO is available to be read.

0: The receive FIFO has no data to be read.

_DRVUART_TRANSMITAVAILABLE

Prototype

UINT8

_DRVUART_TRANSMITAVAILABLE (UINT16 u16Port);

Description

Checks the status of UART transmission FIFO. If the FIFO is available; user can write the data into the UART transmission FIFO.

Parameter

u16Port [in]

Specify UART0/UART1

Include

Driver/DrvUART.h

Return Value

1: The transmission FIFO is available to be written.

0: The transmission FIFO is not available to be written.

_DRVUART_SET_RTS

Prototype

```
VOID
_DRVUART_SET_RTS (
        UINT16    u16Port,
        UINT8     u8Value
);
```

Description

Specify the RTS output value.

Parameter

u16Port [in]

Specify UART0/UART1

u8Vaule [in]

Specify the RTS output value.

Include

Driver/DrvUART.h

Return Value

None

_DRVUART_SET_TIMEOUT_COMPARATOR

Prototype

```
UINT8
_DRVUART_SET_TIMEOUT_COMPARATOR (
        UINT16    u16Port,
        UINT8     u8TOUT
);
```

Description

Specify the time out comparator.

Parameter

u16Port [in]

Specify UART0/UART1

u8TOUT [in]

Specify the time out comparator.

Include

Driver/DrvUART.h

Return Value

None

_DRVUART_GET_TIMEOUT_COMPARATOR

Prototype

UINT8

_DRVUART_GET_TIMEOUT_COMPARATOR (UINT16 u16Port);

Description

Get the time out comparator.

Parameter

u16Port [in]

Specify UART0/UART1

Include

Driver/DrvUART.h

Return Value

The time out comparator.

28.3. Functions

DrvUART_Open

Prototype

ERRCODE

DrvUART_Open (

UART_PORT port,

UART_T *sParam

);

Description

The function is used to initialize UART

Parameter

Port [in]

Specify UART0/UART1

sParam [in]

Specify the property of UART. It includes

u32BaudRate: Baud rate

u8cParity: NONE/EVEN/ODD parity

u8cDataBits: DRVUART_DATA_BITS_5 to DRVUART_DATA_BITS_8

u8cStopBits: DRVUART_STOPBITS_1 to DRVUART_STOPBITS_2

u8cRxTriggerLevel: LEVEL_1_BYTE to LEVEL_62_BYTES

u8TimeOut: Time out value

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success.

E_DRVUART_ERR_PORT_INVALID: Wrong port

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

DrvUART_Close

Prototype

```
VOID    DrvUART_Close (
    UART_PORT      Port
);
```

Description

Close the UART clock and disable the AIC channel.

Parameter

Port [in]

Specify UART0/UART1

Include

Driver/DrvUART.h

Return Value

None

DrvUART_Set_Clock

Prototype

```
VOID
DrvUART_Set_Clock (
    UINT8       u8ClockSrc,
    UINT8       u8ClockDiv
);
```

Description

DrvUART_Set_Clock sets the UART clock source and divider.

Parameters

u8ClcokSrc [in]

Specify the UART clock source

DRVUART_CLKSRC_EXT : Clock source from crystal in.

DRVUART_CLKSRC_PLL : Clock source from divided MPLL clock

DRVUART_CLKSRC_PLL_DIV2 : Clock source from divided MPLL clock / 2

u8ClockDiv [in]

Specify the UART clock divider.

Include

Driver/DrvUART.h

Return Value

None

DrvUART_Get_Clock

Prototype

```
VOID
DrvUART_Get_Clock (
    PUINT8      pu8ClockSrc,
    PUINT8      pu8ClockDiv
);
```

Description

DrvUART_Get_Clock gets the UART clock source and divider.

Parameters

pu8ClcokSrc [in]

Specify the address of buffer to store the UART clock source

pu8ClockDiv [in]

Specify the address of buffer to store the UART clock divider.

Include

Driver/DrvUART.h

Return Value

None

DrvUART_EnableInt

Prototype

```
VOID    DrvUART_EnableInt (
    UINT8       u8Port
    UINT32      u32InterruptFlag,
    PFN_DRVUART_CALLBACK pfncallback
);
```

Description

The function is used to enable UART Interrupt and Install the call back function

Parameter

u8Port [in]

Specify UART0/UART1

u32InterruptFlag [in]

DRVUART_WAKEUPT : 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.

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.

DrvUART_IsIntEnabled

Prototype

```
UINT32
DrvUART_IsIntEnabled (
    UINT16      u16Port
    UINT32      u32InterruptFlag
);
```

Description

The function is used to get the interrupt enable status

Parameter

u16Port [in]

Specify UART0/UART1

u32InterruptFlag [in]

DRVUART_WAKEUPT : 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

1: Enable.

0: Disable.

Note

It is recommended to query one interrupt at a time.

DrvUART_DisableInt

Prototype

```
VOID    DrvUART_DisableInt (
    UINT16      u16Port
```

```
    UINT32      u32InterruptFlag
);
```

Description

The function is used to disable UART Interrupt and uninstall the call back function

Parameter

u16Port [in]

Specify UART0/UART1

u32InterruptFlag [in]

DRVUART_WAKEUPT : 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.

DrvUART_ClearInt

Prototype

```
UINT8
DrvUART_ClearInt (
    UINT16      u16Port
    UINT32      u32InterruptFlag
);
```

Description

The function is used to clear UART Interrupt

Parameter

u16Port [in]

Specify UART0/UART1

u32InterruptFlag [in]

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

For MODEM Status Interrupt, return the value of MODEM Status Register.

For Receive Line Status Interrupt, return the value of Line Status Register.

For Transmit Holding Register Empty Interrupt, return the value of Interrupt Identification Register.

For Receive Data Available Interrupt, return the value of Receive Buffer Register.

For Time-out Interrupt, clear the RX FIFO and return 0.

Note

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

DrvUART_GetIntStatus

Prototype

```
BOOL
DrvUART_GetIntStatus (
    UINT16      u16Port
    UINT32      u32InterruptFlag
);
```

Description

The function is used to get the interrupt status

Parameter

u16Port [in]

Specify UART0/UART1

u32InterruptFlag [in]

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: None.

1: Interrupt occurs.

Note

It is recommended to poll one interrupt at a time.

DrvUART_SetFIFOTriggerLevel

Prototype

```
VOID    DrvUART_SetFIFOTriggerLevel (
    UINT16      u16Port
    UINT32      u32TriggerLevel
);
```

Description

The function is used to set Rx FIFO Trigger Level

Parameter

u16Port [in]

Specify UART0/UART1

u32TriggerLevel [in]

RX FIFO interrupt trigger level.

DRVUART_FIFO_1BYTES : 1 bytes.

DRVUART_FIFO_4BYTES : 4 bytes.

DRVUART_FIFO_8BYTES : 8 bytes.

DRVUART_FIFO_14BYTES : 14 bytes.

DRVUART_FIFO_30BYTES : 30 bytes.

DRVUART_FIFO_46BYTES : 46 bytes.

DRVUART_FIFO_62BYTES : 62 bytes.

Include

Driver/DrvUART.h

Return Value

None

DrvUART_GetCTS

Prototype

```
VOID
DrvUART_GetCTS (
    UINT16     u16Port,
    PUINT8     pu8CTSValue,
    PUINT8     pu8CTSChangeState
)
```

Description

Get CTS value and CTS change state

Parameter

u16Port [in]

Specify UART0/UART1

pu8CTSValue [in]

Specify the buffer to receive the CTS value

pu8CTSChangeState [in]

Specify the buffer to receive the CTS change state

Include

Driver/DrvUART.h

Return Value

None

DrvUART_SetRTS

Prototype

```
VOID
DrvUART_GetCTS (
    UINT16     u16Port,
    UINT8      u8Value
)
```

Description

Set RTS value

Parameter

u16Port [in]

Specify UART0/UART1

u8Value [in]

Specify the RTS value

Include

Driver/DrvUART.h

Return Value

None

DrvUART_SetRxTimeOut

Prototype

```
VOID
DrvUART_SetRxTimeOut (
  UINT16    u16Port,
  UINT8     u8TimeOut
}
```

Description

The function is used to set Rx Time Out Value

Parameter

u16Port [in]

Specify UART0/UART1

u8TimeOut [in]

Specify the Time out value

Include

Driver/DrvUART.h

Return Value

None

DrvUART_Read

Prototype

```
ERRCODE
DrvUART_Read (
  UINT16    u16Port
```

```
    PUINT8    pu8RxBuf,
    UINT32    u32ReceiveBytes
);
```

Description

The function is used to read Rx data from RX buffer

Parameter

u16Port [in]

Specify UART0/UART1

pu8RxBuf [out]

Specify the buffer to receive the data of receive FIFO.

u32ReceiveBytse [in]

Specify the bytes number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success.

E_DRVUART_TIMEOUT: FIFO polling timeout.

DrvUART_Write

Prototype

```
ERRCODE
DrvUART_Write(
    UINT16    u16Port
    PUINT8    pu8TxBuf,
    UINT32    u32SendBytes
);
```

Description

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

Parameter

u168Port [in]

Specify UART0/UART1

pu8TxBuf [in]

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

u32SendBytse [in]

Specify the byte number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success

E_DRVUART_TIMEOUT: FIFO polling timeout

DrvUART_GetVersion

Prototype

UINT32

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

29. DrvUSB Introduction

29.1. USB introduction

This article is provided for manufacturers who are using USB IP to complete their USB applications. It is assumed that the reader is familiar with the Universal Serial Bus Specification, Revision 1.1.

29.2. USB Feature

- Conform to USB1.1 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 256 bytes internal SRAM as USB buffer.
- Provide remote wake-up capability.

29.3. Call Flow

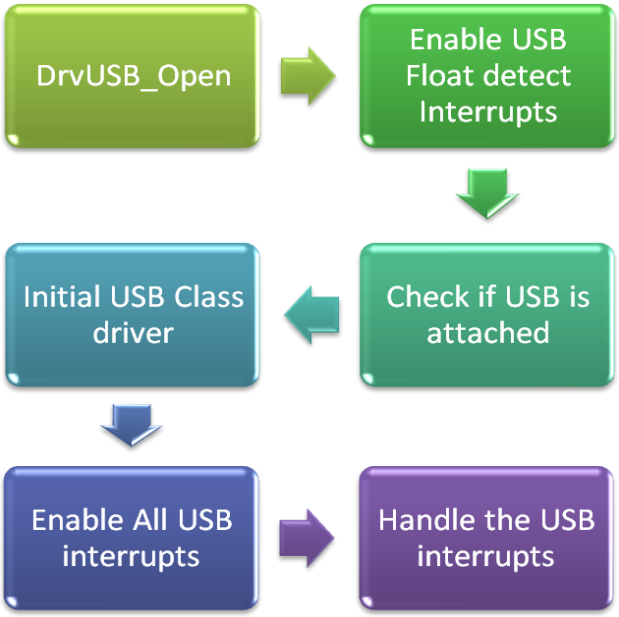


Figure 29-1: USB Driver Call Flow

30. DrvUSB APIs Specification

30.1. Constant Definition

Name	Value	Description
DRVUSB_DETSEL_GPA14	1	USB connection detect pin from GPA[14]
DRVUSB_DETSEL_GPA15	2	USB connection detect pin from GPA[15]
DRVUSB_DETSEL_GPB0	3	USB connection detect pin from GPB[0]
DRVUSB_DETSEL_GPB1	4	USB connection detect pin from GPB[1]
DRVUSB_DETSEL_GPB2	5	USB connection detect pin from GPB[2]
DRVUSB_DETSEL_GPB3	6	USB connection detect pin from GPB[3]
DRVUSB_DETSEL_GPB4	7	USB connection detect pin from GPB[4]
DRVUSB_DETSEL_GPB8	8	USB connection detect pin from GPB[8]
DRVUSB_DETSEL_GPB9	9	USB connection detect pin from GPB[9]
DRVUSB_DETSEL_GPC0	10	USB connection detect pin from GPC[0]
DRVUSB_DETSEL_GPC1	11	USB connection detect pin from GPC[1]
DRVUSB_DETSEL_GPC2	12	USB connection detect pin from GPC[2]
DRVUSB_DETSEL_GPC3	13	USB connection detect pin from GPC[3]
DRVUSB_DETSEL_GPC4	14	USB connection detect pin from GPC[4]

30.2. Macro Functions

_DRVUSB_ENABLE_MISC_INT

Prototype

```
static __inline
VOID _DRVUSB_ENABLE_MISC_INT (
   UINT32     u32Flags
);
```

Description

Enable/Disable miscellaneous interrupts.

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

_DRVUSB_ENABLE_WAKEUP

Prototype

static __inline

VOID _DRVUSB_ENABLE_WAKEUP (VOID);

Description

Enable USB wakeup function.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_WAKEUP

Prototype

static __inline

VOID _DRVUSB_DISABLE_WAKEUP (VOID);

Description

Disable USB wakeup function.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_ENABLE_WAKEUP_INT

Prototype

static __inline

VOID _DRVUSB_ENABLE_WAKEUP_INT (VOID);

Description

Enable wakeup interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_WAKEUP_INT

Prototype

static __inline

VOID _DRVUSB_DISABLE_WAKEUP_INT (VOID);

Description

Disable wakeup interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_CLEAR_WAKEUP_INT

Prototype

static __inline

UINT32

_DRVUSB_CLEAR_WAKEUP_INT (VOID);

Description

Clear wakeup interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return FLODET register value

_DRVUSB_ENABLE_FLD_INT

Prototype

static __inline

VOID _DRVUSB_ENABLE_FLD_INT (VOID);

Description

Enable float-detection interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_FLD_INT

Prototype

static __inline

VOID _DRVUSB_DISABLE_FLD_INT (VOID);

Description

Disable float-detection interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_CLEAR_FLD_INT

Prototype

static __inline

UINT32

_DRVUSB_CLEAR_FLD_INT (VOID);

Description

Clear float-detection interrupts.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return FLODET register value

_DRVUSB_ENABLE_USB_INT

Prototype

static __inline

VOID _DRVUSB_ENABLE_USB_INT (VOID);

Description

Enable USB interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_USB_INT

Prototype

static __inline

VOID _DRVUSB_DISABLE_USB_INT (VOID);

Description

Disable USB interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_CLEAR_USB_INT

Prototype

static __inline

UINT32

_DRVUSB_CLEAR_USB_INT (VOID);

Description

Clear USB interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return STS register value

_DRVUSB_ENABLE_BUS_INT

Prototype

static __inline

VOID _DRVUSB_ENABLE_BUS_INT (VOID);

Description

Enable bus interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_BUS_INT

Prototype

static __inline

VOID _DRVUSB_DISABLE_BUS_INT (VOID);

Description

Disable bus interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_CLEAR_BUS_INT

Prototype

static __inline

UINT32

_DRVUSB_CLEAR_BUS_INT (VOID);

Description

Clear bus interrupts.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return ATTR register value

_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL

Prototype

```
static __inline
VOID   _DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL (
   UINT32     u32EPNum
);
```

Description

Clear EP In/Out Ready and respond STALL,

Parameter

u32EPNum[in]

EP number(valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Notes

Here, EP means number of USB IP EP configuration, not USB EP.

_DRVUSB_CLEAR_EP_READY

Prototype

```
static __inline
VOID   _DRVUSB_CLEAR_EP_READY (
   UINT32     u32EPNum
);
```

Description

Clear EP In/Out Ready.

Parameter

u32EPNum[in]

EP number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Notes

Here, EP means number of USB IP EP configuration, not USB EP.

_DRVUSB_SET_SETUP_BUF

Prototype

```
static __inline
VOID _DRVUSB_SET_SETUP_BUF (
    UINT32      u32BufAddr
);
```

Description

Specify buffer address for Setup transaction.

Parameter

u32BufAddr [in]

Buffer address for setup token. Must be USB_BA+0x100 ~ USB_BA+0x1FF.

Include

Driver/DrvUsb.h

Return Value

None

Notes

u32BufAddr must be between USB_BA+0x100 ~ USB_BA+0x1FF and must be multiples of 8.

_DRVUSB_SET_EP_BUF

Prototype

```
static __inline
VOID _DRVUSB_SET_EP_BUF (
    UINT32      u32EPNum,
    UINT32      u32BufAddr
```

);

Description

Specify buffer address for EP.

Parameter

u32EPNum [in]

EP number (valid value: 0 ~ 5).

u32BufAddr [in]

Buffer address.

Include

Driver/DrvUsb.h

Return Value

None

Notes

u32BufAddr must be between USB_BA+0x100 ~ USB_BA+0x1FF and must be multiples of 8.

Here, EP means number of USB IP EP configuration, not USB EP.

_DRVUSB_TRIG_EP

Prototype

```
static __inline
VOID    _DRVUSB_TRIG_EP (
    UINT32      u32EPNum,
    UINT32      u32TrigSize
);
```

Description

Trigger next transaction for EP.

Parameter

u32EPNum [in]

EP number (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

Notes

Here, EP means number of USB IP EP configuration, not USB EP.

_DRVUSB_GET_EP_DATA_SIZE

Prototype

```
static __inline
UINT32
_DRVUSB_GET_EP_DATA_SIZE (
    UINT32      u32EPNum
);
```

Description

Length of data transmitted to or received from Host for EP.

Parameter

u32EPNum [in]

EP number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

Return MXPLDx, where x = 0 ~ 5

Notes

Here, EP means number of USB IP EP configuration, not USB EP.

_DRVUSB_SET_EP_TOG_BIT

Prototype

```
static __inline
VOID    _DRVUSB_SET_EP_TOG_BIT (
    UINT32      u32EPNum,
    BOOL        bData0
)
```

Description

Specify Data0 or Data1 after IN token toggle automatically after Host ACK.

Parameter

u32EPNum [in]

EP number (valid value: 0 ~ 5).

bData0 [in]

Specify Data0 or Data1 for Data In transaction.

Include

Driver/DrvUsb.h

Return Value

None

Notes

Here, EP means number of USB IP EP configuration, not USB EP.

_DRVUSB_SET_EVF

Prototype

```
static __inline
VOID _DRVUSB_SET_EVF (
   UINT32     u32Data
);
```

Description

Set Interrupt Event Flag

Parameter

u32Data [in]

Specify Data In EVF

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_GET_EVF

Prototype

```
static __inline
UINT32
_DRVUSB_GET_EVF (VOID);
```

Description

Get Interrupt Event Flag

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return EVF register value

_DRVUSB_CLEAR_EP_STALL

Prototype

```
static __inline
VOID _DRVUSB_CLEAR_EP_STALL (
  UINT32    u32EPNum
);
```

Description

Clear EP Force device to response STALL

Parameter

u32EPNum [in]

EP number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_TRIG_EP_STALL

Prototype

```
static __inline
VOID _DRVUSB_TRIG_EP_STALL (
  UINT32    u32EPNum
);
```

Description

Trigger EPx (x = 0 ~ 5) Force device to response STALL

Parameter

u32EPNum [in]

EP number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_CLEAR_EP_DSQ

Prototype

```
static __inline
  VOID _DRVUSB_CLEAR_EP_DSQ (
  UINT32     u32EPNum
);
```

Description

Clear EP Specify Data 0 or 1 after IN token toggle automatically after host ACK

Parameter

u32EPNum [in]

EP number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFG

Prototype

```
static __inline
  VOID _DRVUSB_SET_CFG (
        UINT32     u32CFGNum,
        UINT32     u32Data
  );
```

Description

Configure Set CFG.

Parameter

u32CFGNum [in]

CFG number (valid value: 0 ~ 5).

u32Data [in]

Specify Data In CFG

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_GET_CFG

Prototype

```
static __inline
UINT32
_DRVUSB_GET_CFG (
   UINT32     u32CFGNum
);
```

Description

Configure Get CFG.

Parameter

u32CFGNum [in]

CFG number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

Return CFG register value

_DRVUSB_SET_FADDR

Prototype

```
static __inline
VOID _DRVUSB_SET_FADDR (
   UINT32     u32Addr
)
```

Description

Set Function Address

Parameter

u32Addr [in]

Specify address in EVF

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_GET_FADDR

Prototype

static __inline

UINT32

_DRVUSB_GET_FADDR (VOID)

Description

Get Function Address

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return ADDR register value

_DRVUSB_SET_STS

Prototype

static __inline

VOID _DRVUSB_SET_STS (

UINT32 u32Data

)

Description

Set System states

Parameter

u32Data [in]

Specify data in STS register

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_GET_STS

Prototype

static __inline

UINT32

_DRVUSB_GET_STS (VOID)

Description

Get System states

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return STS register value

_DRVUSB_SET_CFGP

Prototype

```
static __inline
VOID _DRVUSB_SET_CFGP(
    UINT8       u8CFGPNum,
    UINT32      u32Data
);
```

Description

Configure Set CFGP.

Parameter

u8CFGPNum[in]

CFGP register number (valid value: 0 ~ 5).

u32Data [in]

Specify data in CFGP register

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_GET_CFGP

Prototype

```
static __inline
UINT32
_DRVUSB_GET_CFGP (
   UINT32     u32CFGPNum
);
```

Description

Configure Get CFGP.

Parameter

u32CFGPNum[in]

CFGP register number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

Return CFGP register value

_DRVUSB_ENABLE_USB

Prototype

```
static __inline
VOID _DRVUSB_ENABLE_USB (VOID)
```

Description

Enable USB, PHY and use remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_USB

Prototype

static __inline

VOID _DRVUSB_DISABLE_USB (VOID)

Description

Disable USB, PHY and use remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_PHY

Prototype

static __inline

VOID _DRVUSB_DISABLE_PHY (VOID)

Description

Disable PHY and don't use remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_ENABLE_SE0

Prototype

static __inline

VOID _DRVUSB_ENABLE_SE0 (VOID)

Description

Enable SE0

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_DISABLE_SE0

Prototype

static __inline

VOID _DRVUSB_DISABLE_SE0 (VOID)

Description

Disable SE0

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFGP0

Prototype

static __inline

VOID _DRVUSB_SET_CFGP0 (

UINT32 u32Data

)

Description

Stall control and In/out ready clear flag of endpoint 0.

Parameter

u32Data [in]

The data that writes to endpoint 0.

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFGP1

Prototype

```
static __inline
VOID _DRVUSB_SET_CFGP1 (
   UINT32     u32Data
)
```

Description

Stall control and In/out ready clear flag of endpoint 1.

Parameter

u32Data [in]

The data that writes to endpoint 1.

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFGP2

Prototype

```
static __inline
VOID _DRVUSB_SET_CFGP2 (
   UINT32     u32Data
)
```

Description

Stall control and In/out ready clear flag of endpoint 2.

Parameter

u32Data [in]

The data that writes to endpoint 2.

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFGP3

Prototype

```
static __inline
VOID _DRVUSB_SET_CFGP3 (
  UINT32    u32Data
)
```

Description

Stall control and In/out ready clear flag of endpoint 3.

Parameter

u32Data [in]

The data that writes to endpoint 3.

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFGP4

Prototype

```
static __inline
VOID _DRVUSB_SET_CFGP4 (
  UINT32    u32Data
)
```

Description

Stall control and In/out ready clear flag of endpoint 4.

Parameter

u32Data [in]

The data that writes to endpoint 4.

Include

Driver/DrvUsb.h

Return Value

None

_DRVUSB_SET_CFGP5

Prototype

```
static __inline
VOID _DRVUSB_SET_CFGP5 (
   UINT32     u32Data
)
```

Description

Stall control and In/out ready clear flag of endpoint 5.

Parameter

u32Data [in]

The data that writes to endpoint 5.

Include

Driver/DrvUsb.h

Return Value

None

30.3. Functions

DrvUSB_GetVersion

Prototype

```
UINT32
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

DrvUSB_Open

Prototype

ERRCODE

DrvUsb_Open (

PVOID 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 u32EPAddr;
    UINT32 u32MaxPacketSize;
    UINT8 *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 501 USB controller.

g_sBusOps:

The structure type of g_sBusOps is as follows:

```
typedef struct
```

```
{
    PFN_DRVUSB_CALLBACK     apfnCallback;
    PVOID                   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

DrvUSB_Close

Prototype

VOID DrvUsb_Close (VOID);

Description

Close USB controller and disable USB interrupt.

Include

Driver/DrvUsb.h

DrvUSB_PreDispatchEvent

Prototype

VOID DrvUSB_PreDispatchEvent(VOID);

Description

Pre-dispatch event base on EVF register.

Parameter

None

Include

Driver/DrvUsb.h

DrvUSB_Isr_PreDispatchEvent

Prototype

VOID DrvUSB_Isr_PreDispatchEvent(VOID)

Description

Pre-dispatch event base on EVF register and dispatch them at the same time. This function can be called in interrupt handler.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

DrvUSB_DispatchEvent

Prototype

VOID DrvUSB_Isr_PreDispatchEvent(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

DrvUSB_IsData0

Prototype

BOOL DrvUSB_IsData0(UINT32 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

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.

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

DrvUSB_GetEpIdentity

Prototype

UINT32 DrvUSB_GetEpIdentity(UINT32 u32EpNum, UINT32 u32EpAttr)

Description

To get endpoint id base on endpoint number and direction. The endpoint id is used to identify the hardware endpoint resource. The range of endpoint id 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

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 number.
6	Can't get relative endpoint id according to the input endpoint number.

DrvUSB_GetEpId

Prototype

UINT32 DrvUSB_GetEpId(UINT32 u32EpNum)

Description

Get endpoint id base on endpoint number. 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 number with direction information at bit 7.

Include

Driver/DrvUSB.h

Return Value

0~5	The endpoint id of specified endpoint number.
6	Can't get relative endpoint id according to the input endpoint number.

DrvUSB_DataOutTrigger

Prototype

ERRCODE DrvUSB_DataOutTrigger(UINT32 u32EpNum, UINT32 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.
u32Size	Maximum size want to receive from USB

Include

Driver/DrvUSB.h

Return Value

0~5	The endpoint id of specified endpoint number.
6	Can't get relative endpoint id according to the input endpoint number.

DrvUSB_GetOutData

Prototype

UINT8* DrvUSB_GetOutData(UINT32 u32EpNum, UINT32 *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.
u32Size	Data size received from USB

Include

Driver/DrvUSB.h

Return Value

0~5	The endpoint id of specified endpoint number.
6	Can't get relative endpoint id according to the input endpoint number.

DrvUSB_DataIn

Prototype

ERRCODE DrvUSB_DataIn(UINT32 u32EpNum, const UINT8 * u8Buffer, UINT32 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.

u8Buffer	The data buffer for DATA IN token.
u32Size	The size of data buffer.

Include

Driver/DrvUSB.h

Return Value

E_SUCCESS	Successful
E_DRVUSB_SIZE_TOO_LONG	The size is larger than maximum packet size

DrvUSB_BusResetCallback

Prototype

VOID DrvUSB_BusResetCallback(PVOID 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

E_SUCCESS	Successful
E_DRVUSB_SIZE_TOO_LONG	The size is larger than maximum packet size

DrvUSB_InstallClassDevice

Prototype

PVOID 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

DrvUSB_InstallCtrlHandler

Prototype

```
ERRCODE DrvUSB_InstallCtrlHandler(
    PVOID                           *device,
    S_DRVUSB_CTRL_CALLBACK_ENTRY    *psCtrlCallbackEntry,
    UINT32                          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

E_SUCCESS	Success
E_DRVUSB_NULL_POINTER	NULL function pointer

DrvUSB_CtrlSetupAck

Prototype

VOID DrvUSB_CtrlSetupAck(PVOID 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

DrvUSB_CtrlDataInAck

Prototype

VOID DrvUSB_CtrlDataInAck(PVOID 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

DrvUSB_CtrlDataOutAck

Prototype

VOID DrvUSB_CtrlDataOutAck(PVOID 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

DrvUSB_CtrlDataInDefault

Prototype

void DrvUSB_CtrlDataInDefault(PVOID 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

DrvUSB_CtrlDataOutDefault

Prototype

VOID DrvUSB_CtrlDataOutDefault(PVOID 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

DrvUSB_Reset

Prototype

VOID DrvUSB_Reset(UINT32 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

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

DrvUSB_ClrCtrlReadyAndTrigStall

Prototype

VOID DrvUSB_ClrCtrlReadyAndTrigStall(VOID)s

Description

Clear control pipe ready flag that was set by MXPLD and send STALL.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_GetSetupBuffer

Prototype

UINT32 DrvUSB_GetSetupBuffer(VOID)

Description

Get setup buffer address of USB SRAM.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Setup buffer address

DrvUSB_GetFreeSram

Prototype

UINT32 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

DrvUSB_EnableSelfPower

Prototype

VOID DrvUSB_EnableSelfPower(VOID)

Description

Enable self-power attribution.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_DisableSelfPower

Prototype

VOID DrvUSB_DisableSelfPower(VOID)

Description

Disable self-power attribution.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_IsSelfPowerEnabled

Prototype

BOOL DrvUSB_IsSelfPowerEnabled(PBOOL pbVoid)

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.

DrvUSB_EnableRemoteWakeup

Prototype

VOID DrvUSB_EnableRemoteWakeup(VOID)

Description

Enable remote wakeup attribution.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_DisableRemoteWakeup

Prototype

VOID DrvUSB_DisableRemoteWakeup(VOID)

Description

Disable remote wakeup attribution.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_IsRemoteWakeupEnabled

Prototype

BOOL DrvUSB_IsRemoteWakeupEnabled (PBOOL pbVoid)

Description

Return remote wakeup is enable or disable.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

TRUE	Support remote wakeup
FALSE	Not support remote wakeup

DrvUSB_SetMaxPower

Prototype

INT32 DrvUSB_SetMaxPower(UINT32 u32MaxPower)

Description

Configure max power. The unit is 2mA. Maximum MaxPower 0xFA (500mA), default is 0x32 (100mA)

Parameter

None

Include

Driver/DrvUSB.h

Return Value

E_SUCCESS	Successful

DrvUSB_GetMaxPower

Prototype

INT32 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)

DrvUSB_EnableUsb

Prototype

INT32 DrvUSB_GetMaxPower(VOID)

Description

Enable USB and PHY.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_DisableUsb

Prototype

VOID DrvUSB_DisableUsb(S_DRVUSB_DEVICE *psDevice)

Description

Disable USB and PHY.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

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

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

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

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

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

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

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

DrvUSB_CtrlSetupSetAddress

Prototype

VOID DrvUSB_CtrlSetupSetAddress(PVOID pVoid)

Description

Setup ACK handler for set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupClearSetFeature

Prototype

VOID DrvUSB_DispatchMiscEvent(S_DRVUSB_DEVICE *psDevice)

Description

Setup ACK handler for clear feature command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupGetConfiguration

Prototype

VOID DrvUSB_CtrlSetupGetConfiguration(PVOID pVoid)

Description

Setup ACK handler for Get configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupGetStatus

Prototype

VOID DrvUSB_CtrlSetupGetStatus(PVOID pVoid)

Description

Setup ACK handler Get status command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupGetInterface

Prototype

VOID DrvUSB_CtrlSetupGetInterface(PVOID pVoid)

Description

Setup ACK handler for get interface command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupSetConfiguration

Prototype

VOID DrvUSB_CtrlSetupSetConfiguration(PVOID pVoid)

Description

Setup ACK handler for Set configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlDataInSetAddress

Prototype

VOID DrvUSB_CtrlDataInSetAddress(PVOID pVoid)

Description

Setup ACK handler for Set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

31. DrvAUDIO Introduction

31.1. AUDIO Introduction

The DrvAUDIO driver is based on DrvAPU and DrvADC to create a simplest way to support audio play and audio recording. The driver will handle interrupt callbacks and buffer management to reduce the complexity of audio play and recoding.

31.2. AUDIO Feature

The audio driver supports following features:

- To initial the audio buffer and callbacks for playing or recording.
- Supports noise reduction when stop at playing.
- Supports block and non-block API for playing and recoding data update.
- Supports ticks to get play or recording interrupt counter.

32. DrvAUDIO APIs Specification

32.1. Functions

DrvAUDIO_SetPlayVolume

Prototype

VOID DrvAUDIO_SetPlayVolume(INT32 volume)

Description

This function is used to translate the volume gain from dB value to Q12 linear gain and store in gPlayVolume.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_SetRecVolume

Prototype

VOID DrvAUDIO_SetRecVolume(INT32 volume)

Description

This function is used to translate the volume gain from dB value to Q12 linear gain and store in gRecVolume.

Parameters

Volume - [in], The play volume setting. it could be -32768(MUTE) and MIN_REC_VOLUME ~ MAX_REC_VOLUME.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_PlayInit

Prototype

VOID DrvAUDIO_PlayInit(INT32 sampleRate)

Description

This function is used to initial the APU, set sampling rate, and install its interrupt. It must be called before play audio by DrvAUDIO library.

Parameters

sampleRate - [in], To set the play sampling rate.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_PlayFinish

Prototype

VOID DrvAUDIO_PlayFinish(VOID)

Description

This function is used to stop audio play and close APU hardware.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_RecordInit

Prototype

VOID DrvAUDIO_RecordInit(INT32 sampleRate)

Description

This function is used to inital record hardware, install interrupts and set default MIC gain.

Parameters

sampleRate - [in], To set the play sampling rate.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_RecordFinish

Prototype

VOID DrvAUDIO_RecordFinish(VOID)

Description

This function is used to stop recording and clock recording hardware.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_FillPlayDataAndBlock

Prototype

VOID DrvAUDIO_FillPlayDataAndBlock(INT16 *pi16Buf, UINT32 u32Samples)

Description

This function is used to fill the new audio samples. The function will return only when all samples are filled into audio play buffer.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_FillPlayData

Prototype

UINT32 DrvAUDIO_FillPlayData(INT16 *pi16Buf, UINT32 u32Samples)

Description

This function is used to fill the new audio samples. If there is no enough space for input samples, the function will return 0. Otherwise, it return the number of new audio samples.

Parameters

pi16Buf - [in], The buffer pointer to the input data buffer.

u32Samples - [in], The number of samples in the input data buffer.

Include

Driver/DrvAUDIO.h

Return Value

0 If the free audio play buffer is larger than u32Samples.

u32Samples If the free audio play buffer is smaller or equal to u32Samples.

DrvAUDIO_GetRecDataAndBlock

Prototype

VOID DrvAUDIO_GetRecDataAndBlock(INT16 *pi16Buf, UINT32 u32Samples)

Description

This function is used to get recording data with specified number of samples. The function will be blocked until the number of samples of recording data has been got.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_GetRecData

Prototype

UINT32 DrvAUDIO_GetRecData(INT16 *pi16Buf, UINT32 u32Samples)

Description

This function is used to get recording data with specified number of samples. The function will return 0 if no enough samples to get and return the requested samples if successful.

Include

Driver/DrvAUDIO.h

Return Value

0 If the number of recording samples is small than u32Samples.

u32Samples If the number of recording samples is larger or equal to u32Samples.

DrvAUDIO_GetRecBufSamples

Prototype

UINT32 DrvAUDIO_GetRecBufSamples(VOID)

Description

This function is used to get the number of samples in recording buffer.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

The number of samples in recording buffer.

DrvAUDIO_StopPlay

Prototype

VOID DrvAUDIO_StopPlay(VOID)

Description

This function is used to stop audio play. By default, it just set the stop play flag to stop and this can avoid hardware stop noise.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_GetPlayBufSamples

Prototype

UINT32 DrvAUDIO_GetPlayBufSamples(VOID)

Description

This function is used to get the number of samples in audio play buffer.

Include

Driver/DrvAUDIO.h

Return Value

The number of samples in audio play buffer.

DrvAUDIO_StartPlay

Prototype

VOID DrvAUDIO_StartPlay(VOID)

Description

Enable LVD interrupt and setup callback function.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_StartRecord

Prototype

VOID DrvAUDIO_StartRecord(VOID)

Description

This function is used to start recording. The DrvAUDIO_RecordInit() should be called before using this function.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_StopRecord

Prototype

VOID DrvAUDIO_StopRecord(VOID)

Description

This function is used to stop recording.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_GetPlayTicks

Prototype

UINT32 DrvAUDIO_GetPlayTicks(VOID)

Description

This function is used to return the APU interrupt counter value.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

Return the counter value of APU interrupt counter.

DrvAUDIO_SetPlayTicks

Prototype

VOID DrvAUDIO_SetPlayTicks(UINT32 u32Ticks)

Description

This function is used to set the APU interrupt counter value.

Parameters

The new APU interrupt counter value.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_GetRecTicks

Prototype

UINT32 DrvAUDIO_GetRecTicks(VOID)

Description

This function is used to return the ADC interrupt counter value.

Parameters

None

Include

Driver/DrvAUDIO.h

Return Value

Return the counter value of ADC interrupt counter.

DrvAUDIO_SetRecTicks

Prototype

VOID DrvAUDIO_SetRecTicks(UINT32 u32Ticks)

Description

This function is used to set the ADC interrupt counter value.

Parameters

The new ADC interrupt counter value.

Include

Driver/DrvAUDIO.h

Return Value

None

DrvAUDIO_GetVersion

Prototype

UINT32 DrvAUDIO_GetVersion(VOID)

Description

To return the driver version.

Include

Driver/DrvAUDIO.h

Return Value

Version number:

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

33. DrvSDCARD Introduction

33.1. Call Flow

- Directly connect to Secure Digital (SD) flash memory card.
- Only support SPI mode. The SD library is implemented in SPI interface and no dedicated DMA mechanism to speed up the data access.

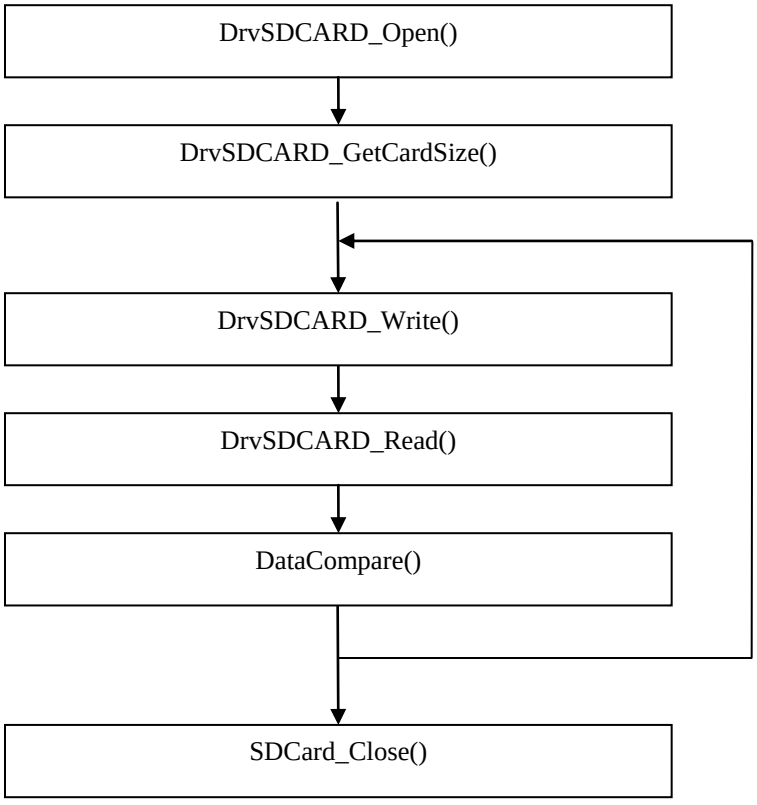


Figure 33-1: SD Card library Call Flow

34. DrvSDCARD APIs Specification

34.1. Functions

DrvSDCARD_Open

Prototype

ERRCODE
DrvSDCARD_Open(
PVOID pVoid
);

Description

Check multifunction I/O control register and initiate SD card.

Parameter

pVoid [in | out]

Reserved.

Include

Driver /DrvSDCARD.h
Driver/ DrvSDCARD_SDdef.h

Return Value

E_SUCCESS: Success
ERRCODE: Error

Notes

To access SD card, this API must be invocated first.

DrvSDCARD_Close

Prototype

VOID DrvSDCARD_Close(VOID);

Description

Exit SD card library. After the invocation of this API, SD card cannot be accessed.

Include

Driver /DrvSDCARD.h

Driver/ DrvSDCARD_SDdef.h

Return Value

None

DrvSDCARD_GetCardSize

Prototype

```
BOOL
DrvSDCARD_GetCardSize(
   PVOID pVoid,
   PUINT32 pu32TotSecCnt
);
```

Description

Get SD card size in sector unit.

Parameter

pVoid [in]

Reserved

pu32TotSecCnt [out]

Caller-allocated buffer to receive the amount of sectors.

Include

Driver /DrvSDCARD.h

Driver/ DrvSDCARD_SDdef.h

Return Value

TRUE: Success

FALSE: Failure

DrvSDCARD_Read

Prototype

```
ERRCODE
DrvSDCARD_Read(
```

UINT32 u32SecAddr,
UINT32 u32SecCnt,
PUINT8 pu8Buffer
);

Description

Read successive sectors from storage.

Parameter

u32StartSecAddr [in]

Start sector address to transfer

u32SecCnt [in]

The amount of sectors for transferring

pu8Buffer [out]

Caller-allocated buffer where called reads from storage and writes to buffer

Include

Driver /DrvSDCARD.h
Driver/ DrvSDCARD_SDdef.h

Return Value

E_SUCCESS: Success
ERRCODE: Error

DrvSDCARD_Write

Prototype

ERRCODE
DrvSDCARD_Write(
UINT32 u32SecAddr,
UINT32 u32SecCnt,
PUINT8 pu8Buffer
);

Description

Write successive sectors to storage

Parameter

u32StartSecAddr [in]

Start sector address to transfer

u32SecCnt [in]

The amount of sectors for transferring

pu8Buffer [out]

Caller-allocated buffer where called reads from and writes to storage

Include

Driver /DrvSDCARD.h

Driver/ DrvSDCARD_SDdef.h

Return Value

E_SUCCESS: Success

ERRCODE: Error

DrvSDCARD_GetCardType

Prototype

ERRCODE

DrvSDCARD_GetCardType(VOID);

Description

Return the SD Card Type.

Include

Driver /DrvSDCARD.h

Driver/ DrvSDCARD_SDdef.h

Return Value

SD_CARD: Normal SD Card

SD20_HIGH_CARD: SDHC Card

DrvSDCARD_GetVersion

Prototype

UINT32

DrvSDCARD_GetVersion(VOID);

Description

Return the current version number of SD library.

Include

Driver/SDCard.h

Return Value

Version number :

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

35. Revision History

Version	Date	Description
V1.00.001	Nov 20, 2008	● Created
V1.00.002	May 14, 2009	● Modified for second release
V1.01.001	Nov 10, 2009	● Modify USB driver and Add AUDIO, SDCAR driver.

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.