

NUC1xx Driver Reference Guide V1.00.001

Publication Release Date: Jan. 2010

Support Chips:
NUC1xx Series

Support Platforms:
Nuvoton

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

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

This documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from the Nuvoton Technology Corp.

Nuvoton Technology Corp. All rights reserved.

Table of Contents

1. DrvSYS Introduction.....	14
1.1. Introduction.....	14
2. DrvSYS APIs Specification	15
2.1. Constant Definition.....	15
2.1.1. IP Reset.....	15
2.1.2. IP Clock Enable Control.....	15
2.2. Functions.....	16
DrvSYS_ReadProductID	16
DrvSYS_GetRstSrc	17
DrvSYS_ClearRstSrc	17
DrvSYS_ResetIP	17
DrvSYS_ResetCPU	18
DrvSYS_ResetChip	18
DrvSYS_EnableBOD	19
DrvSYS_SelectBODVolt.....	19
DrvSYS_EnableBODRst.....	19
DrvSYS_EnableBODLowPowerMode.....	20
DrvSYS_EnableLowVoltRst	20
DrvSYS_GetBODState.....	21
DrvSYS_EnableTempatureSensor.....	21
DrvSYS_SetPORDisCode	22
DrvSYS_UnlockKeyAddr	22
DrvSYS_LockKeyAddr.....	22
DrvSYS_SetRCAdjValue.....	23
DrvSYS_SetIPClock.....	23
DrvSYS_SetHCLKSource.....	24
DrvSYS_SetSysTickSource.....	24
DrvSYS_SetIPClockSource.....	25
DrvSYS_SetClockDivider	25
DrvSYS_SetOscCtrl	26
DrvSYS_EnablePWRWUInt	26
DrvSYS_EnablePowerDown.....	27
DrvSYS_SetPowerDownWaitCPU	28
DrvSYS_SetPllSrc	28
DrvSYS_SetPLLPowerDown.....	28
DrvSYS_GetEXTClock.....	29
DrvSYS_GetPllContent	29
DrvSYS_GetPLLClock	30
DrvSYS_GetHCLK	30
DrvSYS_Open	31

3. DrvUART Introduction.....	32
3.1. UART Introduction.....	32
3.2. UART Feature.....	32
4. DrvUART APIs Specification	33
4.1. Constant Definition.....	33
4.2. Functions.....	34
DrvUART_Open.....	34
DrvUART_Close	35
DrvUART_EnableInt.....	35
DrvUART_IsIntEnabled.....	36
DrvUART_DisableInt.....	37
DrvUART_ClearInt	38
DrvUART_GetIntStatus	38
DrvUART_SetFIFOTriggerLevel.....	39
DrvUART_GetCTS	40
DrvUART_SetRTS.....	41
DrvUART_SetRxTimeOut	41
DrvUART_Read	42
DrvUART_Write	42
DrvUART_SetPDMA.....	43
DrvUART_OpenIRCR	44
DrvUART_OpenLIN.....	44
DrvUART_GetVersion.....	45
5. DrvTIMER Introduction	46
5.1. Timer Introduction	46
5.2. Timer Feature	46
6. DrvTIMER APIs Specification.....	47
6.1. Function	47
DrvTIMER_GetStatus	47
DrvTIMER_SetTimerEvent.....	47
DrvTIMER_ClearTimerEvent	48
DrvTIMER_ResetTicks.....	48
DrvTIMER_Init.....	49
DrvTIMER_Open	49
DrvTIMER_GetTicks	50
DrvTIMER_Delay	50
DrvTIMER_Ioctl	51
DrvTIMER_Close.....	52
DrvWDT_Open	52
DrvWDT_ResetCount	53
DrvWDT_Ioctl.....	53
DrvWDT_Close.....	54

DrvTIMER_GetVersion	54
7. DrvGPIO Introduction.....	55
7.1. GPIO introduction.....	55
8. DrvGPIO APIs Specification	56
8.1. Functions.....	56
DrvGPIO_Open	56
DrvGPIO_Close.....	56
DrvGPIO_SetBit.....	57
DrvGPIO_ClrBit.....	57
DrvGPIO_GetBit	58
DrvGPIO_SetPortBits.....	58
DrvGPIO_GetPortBits	59
DrvGPIO_GetPortDoutBits	59
DrvGPIO_EnableInt	60
DrvGPIO_DisableInt	60
DrvGPIO_SetDebounceTime	61
DrvGPIO_EnableDebounce.....	61
DrvGPIO_DisableDebounce	62
DrvGPIO_GetDebounceTime	63
DrvGPIO_GetIntStatus.....	63
DrvGPIO_InitFunction	63
DrvGPIO_GetDoutBit	64
DrvGPIO_BitIsUsed.....	65
DrvGPIO_SetBitMask	65
DrvGPIO_ClrBitMask	66
DrvGPIO_SetPortMask	66
DrvGPIO_ReadPortMask	67
DrvGPIO_InstallSR	67
DrvGPIO_GetVersion	68
9. DrvADC Introduction	69
9.1. ADC Introduction	69
9.2. ADC Feature	69
10. DrvADC APIs Specification.....	70
10.1. Type Definition.....	70
10.2. Macros.....	70
_DRVADC_CONV	70
10.3. Functions.....	71
DrvADC_Open	71
DrvADC_Close.....	72
DrvADC_SetAdcChannel.....	72

DrvADC_ConfigAdcChannel7	72
DrvADC_SetAdcInputMode	73
DrvADC_SetAdcOperationMode	73
DrvADC_SetAdcClkSrc	74
DrvADC_SetAdcDivisor	74
DrvADC_EnableAdcInt	74
DrvADC_DisableAdcInt	75
DrvADC_EnableAdcmp0Int	75
DrvADC_DisableAdcmp0Int	76
DrvADC_EnableAdcmp1Int	76
DrvADC_DisableAdcmp1Int	77
DrvADC_GetConversionRate	77
DrvADC_ExtTriggerEnable	78
DrvADC_ExtTriggerDisable	78
DrvADC_StartConvert	78
DrvADC_StopConvert	79
DrvADC_IsConversionDone	79
DrvADC_GetConversionData	80
DrvADC_PdmaEnable	80
DrvADC_PdmaDisable	80
DrvADC_IsDataValid	81
DrvADC_IsDataOverrun	81
DrvADC_Adcmp0Enable	82
DrvADC_Adcmp0Disable	82
DrvADC_Adcmp1Enable	83
DrvADC_Adcmp1Disable	84
DrvADC_SelfCalEnable	84
DrvADC_IsCalDone	84
DrvADC_SelfCalDisable	85
DrvADC_GetVersion	85

11. DrvSPI Introduction86

11.1. SPI Introduction	86
------------------------------	----

11.2. General Feature	86
-----------------------------	----

12. DrvSPI APIs Specification87

12.1. Constant Definition	87
---------------------------------	----

12.2. Functions	88
-----------------------	----

DrvSPI_Open	88
DrvSPI_Close	89
DrvSPI_Set2BitSerialDataIOMode	89
DrvSPI_SetEndian	90
DrvSPI_SetBitLength	90
DrvSPI_SetByteSleep	91
DrvSPI_SetByteEndian	92
DrvSPI_SetTriggerMode	92
DrvSPI_SetSlaveSelectActiveLevel	93
DrvSPI_GetLevelTriggerStatus	93
DrvSPI_EnableAutoCS	94

DrvSPI_DisableAutoCS	95
DrvSPI_SetCS	95
DrvSPI_ClrCS	96
DrvSPI_Busy	96
DrvSPI_BurstTransfer	97
DrvSPI_SetClock.....	97
DrvSPI_GetClock1	98
DrvSPI_GetClock2	99
DrvSPI_SetVariableClockPattern	99
DrvSPI_SetVariableClockFunction	100
DrvSPI_EnableInt.....	100
DrvSPI_DisableInt.....	101
DrvSPI_SingleRead	101
DrvSPI_SingleWrite	102
DrvSPI_BurstRead	103
DrvSPI_BurstWrite.....	103
DrvSPI_DumpRxRegister	104
DrvSPI_SetTxRegister.....	104
DrvSPI_SetGo	105
DrvSPI_GetJoyStickIntType	105
DrvSPI_SetJoyStickStatus.....	106
DrvSPI_GetJoyStickMode.....	106
DrvSPI_StartPMDA	107
DrvSPI_GetVersion.....	108
13. DrvI2C Introduction.....	109
13.1. Introduction.....	109
13.2. Feature.....	109
14. DrvI2C APIs Specification	110
14.1. Functions.....	110
DrvI2C_Open	110
DrvI2C_Close	110
DrvI2C_SetClock	111
DrvI2C_GetClock.....	111
DrvI2C_SetAddress	112
DrvI2C_SetAddressMask	112
DrvI2C_GetStatus.....	113
DrvI2C_WriteData	113
DrvI2C_ReadData	114
DrvI2C_Ctrl.....	114
DrvI2C_GetIntFlag.....	115
DrvI2C_ClearIntFlag	115
DrvI2C_EnableInt.....	116
DrvI2C_DisableInt	116
DrvI2C_InstallCallBack	116
DrvI2C_UninstallCallBack.....	117
DrvI2C_EnableTimeoutCount.....	118
DrvI2C_ClearTimeoutFlag	118

15. DvRTC Introduction.....	120
15.1. General RTC Controller Introduction	120
15.2. RTC Features	120
16. DvRTC APIs Specification	121
16.1. Constant Definition	121
16.2. Functions	122
DvRTC_SetFrequencyCompenation	122
DvRTC_WriteEnable	122
DvRTC_Init.....	123
DvRTC_Open.....	123
DvRTC_Read	124
DvRTC_Write	125
DvRTC_Iocctl	126
DvRTC_Close	127
DvRTC_GetVersion	128
17. DvCAN Introduction	129
17.1. CAN Introduction	129
17.2. CAN Feature	129
18. DvCAN APIs Specification.....	130
18.1. Function	130
DvCAN_Open	130
DvCAN_DisableInt	130
DvCAN_EnableInt	131
DvCAN_GetErrorStatus.....	132
DvCAN_ReadMsg	132
DvCAN_SetAcceptanceFilter	132
DvCAN_SetMaskFilter	133
DvCAN_WaitReady	133
DvCAN_WriteMsg	134
DvCAN_GetVersion	135
19. DvPWM Introduction.....	136
19.1. PWM Introduction	136
20. DvPWM APIs Specification	137
20.1. Constant Definition	137
20.2. Functions	138

DrvPWM_IsTimerEnabled	138
DrvPWM_SetTimerCounter	138
DrvPWM_GetTimerCounter	139
DrvPWM_EnableInt	139
DrvPWM_DisableInt	140
DrvPWM_ClearInt	141
DrvPWM_GetIntFlag	142
DrvPWM_GetRisingCounter	142
DrvPWM_GetFallingCounter	143
DrvPWM_GetCaptureIntStatus	143
DrvPWM_ClearCaptureIntStatus	144
DrvPWM_Open	145
DrvPWM_Close	145
DrvPWM_EnableDeadZone	145
DrvPWM_Enable	146
DrvPWM_SetTimerClk	147
DrvPWM_SetTimerIO	148
DrvPWM_SelectClockSource	149
21. DrvPS2 Introduction	150
21.1. PS2 Introduction	150
21.2. PS2 Feature	150
22. DrvSP2 APIs Specification.....	151
22.1. Macro	151
DRVPS2_OVERRIDE	151
DRVPS2_PS2CLK	151
DRVPS2_PS2DATA	152
DRVPS2_CLRFIFO	152
DRVPS2_ACKNOTALWAYS	153
DRVPS2_RXINTENABLE	153
DRVPS2_RXINTDISABLE	153
DRVPS2_TXINTENABLE	154
DRVPS2_TXINTDISABLE	154
DRVPS2_PS2ENABLE	155
DRVPS2_PS2DISABLE	155
DRVPS2_TXFIFO	155
DRVPS2_SWOVERRIDE	156
DRVPS2_INTCLR	156
DRVPS2_RXDATA	157
DRVPS2_TXDATAWAIT	157
DRVPS2_TXDATA	158
DRVPS2_TXDATA0	158
DRVPS2_TXDATA1	159
DRVPS2_TXDATA2	159
DRVPS2_TXDATA3	160
DRVPS2_ISTXEMPTY	160
DRVPS2_ISFRAMEERR	160
DRVPS2_ISRXBUSY	161

22.2. Functions.....	161
DrvPS2_Open	161
DrvPS2_Close	162
DrvPS2_EnableInt	162
DrvPS2_DisableInt	163
DrvPS2_IsIntEnabled	163
DrvPS2_ClearIn.....	164
DrvPS2_GetIntStatus.....	164
DrvPS2_SetTxFIFODepth.....	164
DrvPS2_Read	165
DrvPS2_Write	165
DrvPS2_GetVersion	166
23. DrvFMC Introduction	167
23.1. Introduction.....	167
23.2. Feature.....	167
24. DrvFMC APIs Specification	168
24.1. Functions.....	168
DrvFMC_EnableISP	168
DrvFMC_BootSelect	168
DrvFMC_GetBootSelect	169
DrvFMC_EnableLDUpdate.....	169
DrvFMC_EnablePowerSaving	169
DrvFMC_ReadCID.....	170
DrvFMC_ReadDID	170
DrvFMC_Write.....	171
DrvFMC_Read	171
DrvFMC_Erase.....	172
DrvFMC_WriteConfig	172
DrvFMC_ReadDataFlashBaseAddr	173
25. DrvUSB Introduction	174
25.1. Introduction.....	174
25.2. Feature.....	174
25.3. Call Flow.....	175
26. DrvUSB APIs Specification.....	176
26.1. Macro Functions	176
_DRVUSB_ENABLE_MISC_INT	176
_DRVUSB_ENABLE_WAKEUP	176
_DRVUSB_DISABLE_WAKEUP	177
_DRVUSB_ENABLE_WAKEUP_INT.....	177
_DRVUSB_DISABLE_WAKEUP_INT.....	178

_DRVUSB_ENABLE_FLD_INT	178
_DRVUSB_DISABLE_FLD_INT	178
_DRVUSB_ENABLE_USB_INT	179
_DRVUSB_DISABLE_USB_INT	179
_DRVUSB_ENABLE_BUS_INT	180
_DRVUSB_DISABLE_BUS_INT	180
_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL	180
_DRVUSB_CLEAR_EP_READY	181
_DRVUSB_SET_SETUP_BUF	182
_DRVUSB_SET_EP_BUF	182
_DRVUSB_TRIG_EP	183
_DRVUSB_GET_EP_DATA_SIZE	183
_DRVUSB_SET_EP_TOG_BIT	184
_DRVUSB_SET_EVF	185
_DRVUSB_GET_EVF	185
_DRVUSB_CLEAR_EP_STALL	186
_DRVUSB_TRIG_EP_STALL	186
_DRVUSB_CLEAR_EP_DSQ	187
_DRVUSB_SET_CFG	187
_DRVUSB_GET_CFG	188
_DRVUSB_SET_FADDR	188
_DRVUSB_GET_FADDR	189
_DRVUSB_SET_STS	189
_DRVUSB_GET_STS	190
_DRVUSB_SET_CFGP	190
_DRVUSB_GET_CFGP	191
_DRVUSB_ENABLE_USB	191
_DRVUSB_DISABLE_USB	191
_DRVUSB_DISABLE_PHY	192
_DRVUSB_ENABLE_SE0	192
_DRVUSB_DISABLE_SE0	193
_DRVUSB_SET_CFGP0	193
_DRVUSB_SET_CFGP1	194
_DRVUSB_SET_CFGP2	194
_DRVUSB_SET_CFGP3	195
_DRVUSB_SET_CFGP4	195
_DRVUSB_SET_CFGP5	196
 26.2. Functions	 196
DrvUSB_GetVersion	196
DrvUSB_Open	197
DrvUSB_Close	198
DrvUSB_PreDispatchEvent	198
DrvUSB_Isr_PreDispatchEvent	199
DrvUSB_DispatchEvent	199
DrvUSB_IsData0	199
DrvUSB_GetUsbState	200
DrvUSB_SetUsbState	200
DrvUSB_GetEpIdentity	201
DrvUSB_GetEpId	201
DrvUSB_DataOutTrigger	202
DrvUSB_GetOutData	202
DrvUSB_DataIn	203
DrvUSB_BusResetCallback	203

DrvUSB_InstallClassDevice	204
DrvUSB_InstallCtrlHandler	204
DrvUSB_CtrlSetupAck	204
DrvUSB_CtrlDataInAck.....	205
DrvUSB_CtrlDataOutAck	205
DrvUSB_CtrlDataInDefault	206
DrvUSB_CtrlDataOutDefault.....	206
DrvUSB_Reset	206
DrvUSB_ClrCtrlReady	207
DrvUSB_ClrCtrlReadyAndTrigStall	207
DrvUSB_GetSetupBuffer	208
DrvUSB_GetFreeSram	208
DrvUSB_EnableSelfPower.....	208
DrvUSB_DisableSelfPower.....	209
DrvUSB_IsSelfPowerEnabled.....	209
DrvUSB_EnableRemoteWakeup.....	209
DrvUSB_DisableRemoteWakeup.....	210
DrvUSB_IsRemoteWakeupEnabled.....	210
DrvUSB_SetMaxPower.....	211
DrvUSB_GetMaxPower	211
DrvUSB_EnableUsb.....	211
DrvUSB_DisableUsb.....	212
DrvUSB_PreDispatchWakeupEvent	212
DrvUSB_PreDispatchFdtEvent	213
DrvUSB_PreDispatchBusEvent	213
DrvUSB_PreDispatchEPEvent	213
DrvUSB_DispatchWakeupEvent.....	214
DrvUSB_DispatchMiscEvent.....	214
DrvUSB_DispatchEPEvent	215
DrvUSB_CtrlSetupSetAddress	215
DrvUSB_CtrlSetupClearSetFeature	215
DrvUSB_CtrlSetupGetConfiguration	216
DrvUSB_CtrlSetupGetStatus.....	216
DrvUSB_CtrlSetupGetInterface	217
DrvUSB_CtrlSetupSetConfiguration.....	217
DrvUSB_CtrlDataInSetAddress	217

27. DrvPDMA Introduction219

27.1. PDMA Introduction	219
27.2. PDMA Feature	219

28. DrvPDMA APIs Specification220

28.1. Functions.....	220
DrvPDMA_Init.....	220
DrvPDMA_Close	220
DrvPDMA_CHEnableTransfer.....	221
DrvPDMA_CHSoftwareReset.....	221
DrvPDMA_Open.....	222
DrvPDMA_ClearInt	222
DrvPDMA_PollInt.....	223

DrvPDMA_SetAPBTransferWidth	224
DrvPDMA_SetCHForAPBDevice	224
DrvPDMA_DisableInt	225
DrvPDMA_EnableInt	225
DrvPDMA_GetAPBTransferWidth.....	226
DrvPDMA_GetCHForAPBDevice.....	227
DrvPDMA_GetCurrentDestAddr	227
DrvPDMA_GetCurrentSourceAddr	228
DrvPDMA_GetCurrentTransferCount	228
DrvPDMA_GetInternalBufPointer	229
DrvPDMA_GetSharedBufData	229
DrvPDMA_GetTransferLength.....	230
DrvPDMA_InstallCallBack.....	231
DrvPDMA_IsCHBusy	231
DrvPDMA_IsIntEnabled	232
DrvPDMA_IsIntEnabled	233
DrvPDMA_GetVersion	233
 29. Revision History	 235

1. DrvSYS Introduction

1.1. Introduction

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

- System memory map
- System interrupt map
- Product ID register
- System management registers for chip and module functional reset and multi-function pin control.
- Brown-Out and chip miscellaneous control.
- Clock generator
- System clock and peripherals clock
- Power down mode

2. DrvSYS APIs Specification

2.1. Constant Definition

2.1.1. IP Reset

Table 2-1: IP reset

Name	Value	Description
E_SYS_GPIO_RST	1	GPIO reset
E_SYS_TMR0_RST	2	Timer0 reset
E_SYS_TMR1_RST	3	Timer1 reset
E_SYS_TMR2_RST	4	Timer2 reset
E_SYS_TMR3_RST	5	Timer3 reset
E_SYS_I2C0_RST	8	I2C0 reset
E_SYS_I2C1_RST	9	I2C1 reset
E_SYS_SPI0_RST	12	SPI0 reset
E_SYS_SPI1_RST	13	SPI1 reset
E_SYS_SPI2_RST	14	SPI2 reset
E_SYS_SPI3_RST	15	SPI3 reset
E_SYS_UART0_RST	16	UART0 reset
E_SYS_UART1_RST	17	UART1 reset
E_SYS_PWM_RST	20	PWM reset
E_SYS_ACMP_RST	22	Analog Comparator reset
E_SYS_PS2_RST	23	PS2 reset
E_SYS_CAN0_RST	24	CAN0 reset
E_SYS_CAN1_RST	25	CAN1 reset
E_SYS_USBD_RST	27	USB device reset
E_SYS_ADC_RST	28	ADC reset
E_SYS_PDMA_RST	32	PDMA reset

2.1.2. IP Clock Enable Control

Table 2-2: IP Clock Enable

Name	Value	Description
E_SYS_WD_CLK	0	Watch Dog clock enable control
E_SYS_RTC_CLK	1	RTC clock enable control
E_SYS_TMR0_CLK	2	Timer0 clock enable control
E_SYS_TMR1_CLK	3	Timer1 clock enable control
E_SYS_TMR2_CLK	4	Timer2 clock enable control
E_SYS_TMR3_CLK	5	Timer3 clock enable control
E_SYS_I2C0_CLK	8	I2C0 clock enable control
E_SYS_I2C1_CLK	9	I2C1 clock enable control
E_SYS_SPI0_CLK	12	SPI0 clock enable control
E_SYS_SPI1_CLK	13	SPI1 clock enable control
E_SYS_SPI2_CLK	14	SPI2 clock enable control
E_SYS_SPI3_CLK	15	SPI3 clock enable control
E_SYS_UART0_CLK	16	UART0 clock enable control
E_SYS_UART1_CLK	17	UART1 clock enable control
E_SYS_PWM01_CLK	20	PWM01 clock enable control
E_SYS_PWM23_CLK	21	PWM23 clock enable control
E_SYS_CAN0_CLK	24	CAN0 clock enable control
E_SYS_CAN1_CLK	25	CAN1 clock enable control
E_SYS_USBD_CLK	27	USB device clock enable control
E_SYS_ADC_CLK	28	ADC clock enable control
E_SYS_ACMP_CLK	30	Analog Comparator clock enable control
E_SYS_PS2_CLK	31	PS2 clock enable control
E_SYS_PDMA_CLK	33	PDMA clock enable control
E_SYS_ISP_CLK	34	Flash ISP controller clock enable control

2.2. Functions

DrvSYS_ReadProductID

Prototype

```
uint32_t DrvSYS_ReadProductID(void);
```

Description

To read product ID.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

Product ID

DrvSYS_GetRstSrc

Prototype

uint32_t DrvSYS_GetRstSrc(void);

Description

To get reset source from last operation

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The value in RSTSRC register.

DrvSYS_ClearRstSrc

Prototype

uint32_t DrvSYS_ClearRstSrc(uint32_t u32Src);

Description

Clear reset source by write 0.

Parameter

u32Src [in]

The corresponding bit of reset source.

Include

Driver/DrvSYS.h

Return Value

0 Succeed

DrvSYS_ResetIP

Prototype

```
void DrvSYS_ResetIP(E_SYS_IP_RST eIpRst);
```

Description

To reset IP include GPIO, Timer0, Timer1, Timer2, Timer3, I2C0, I2C1, SPI0, SPI1, SPI2, SPI3, UART0, UART1, PWM, ACMP, PS2, CAN0, CAN1, USB0, ADC, and PDMA.

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 CPU.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_ResetChip

Prototype

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

Description

To reset whole chip.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_EnableBOD

Prototype

```
void DrvSYS_EnableBOD(int32_t i32Enable);
```

Description

To enable Brown-Out function.

Parameter

i32Enable [in]

1:enable, 0:disable

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SelectBODVolt

Prototype

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

Description

To select BOD threshold voltage

Parameter

u8Volt [in]

3: 4.5V, 2: 3.8V, 1: 2.6V, 0: 2.2V

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_EnableBODRst

Prototype

```
void DrvSYS_EnableBODRst(int32_t i32Enable, BOD_CALLBACK bodcallbackFn);
```

Description

To enable Brown-out reset function or interrupt function.

Parameter

i32Enable [in]

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

bodcallbackFn [in]

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

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_EnableBODLowPowerMode

Prototype

```
void DrvSYS_EnableBODLowPowerMode(int32_t i32Enable);
```

Description

To enable Brown-out low power mode.

Parameter

i32Enable [in]

1: low power mode, 0: normal mode

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_EnableLowVoltRst

Prototype

```
void DrvSYS_EnableLowVoltRst(int32_t i32Enable);
```

Description

To enable LVR function reset the chip when input voltage is lower than LVR circuit.

Parameter

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

None.

DrvSYS_GetBODState

Prototype

uint32_t DrvSYS_GetBODState(void);

Description

To get BOD state.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

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

0: the detected voltage is higher than BOD threshold voltage

DrvSYS_EnableTempatureSensor

Prototype

void DrvSYS_EnableTempatureSensor(int32_t i32Enable);

Description

To enable temperature sensor function.

Parameters

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SetPORDisCode

Prototype

```
void DrvSYS_SetPORDisCode(uint32_t u32Code);
```

Description

To set POD DIS CODE for power on reset enable control

Parameters

u32Code [in]

POD DIS CODE

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_UnlockKeyAddr

Prototype

```
int32_t DrvSYS_UnlockKeyAddr(void);
```

Description

To unlock protected registers.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

DrvSYS_LockKeyAddr

Prototype

```
int32_t DrvSYS_LockKeyAddr(void);
```

Description

To lock protected registers.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

DrvSYS_SetRCAdjValue

Prototype

```
void     DrvSYS_SetRCAdjValue(uint32_t u32Adj);
```

Description

To set RC adjustment value.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SetIPClock

Prototype

```
void    DrvSYS_SetIPClock(E_SYS_IP_CLK eIpClk, int32_t i32Enable);
```

Description

To enable IP clock include Watch Dog, RTC, Timer0, Timer1, Timer2, Timer3, I2C0, I2C1, SPI0, SPI1, SPI2, SPI3, UART0, UART1, PWM01, PWM23, CAN0, CAN1, USBD, ADC, ACMP, PS2, PDMA and Flash ISP controller.

Parameter

eIpClk [in]

Enumeration for IP clock, reference the IP Clock Enable Control.

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SetHCLKSource

Prototype

```
int32_t DrvSYS_SetHCLKSource(uint8_t u8ClkSrcSel);
```

Description

To select HCLK clock source from external 12M crystal clock, external 32K crystal clock, PLL clock, internal 10K oscillator clock, or internal 22M oscillator clock.

Parameter

u8ClkSrcSel [in]

- 0: External 12M clock
- 1: External 32K clock
- 2: PLL clock
- 3: Internal 10K clock
- 4~7: Internal 22M clock

Include

Driver/DrvSYS.h

Return Value

- 0 Succeed
- < 0 Wrong parameter

DrvSYS_SetSysTickSource

Prototype

```
int32_t DrvSYS_SetSysTickSource(uint8_t u8ClkSrcSel);
```

Description

To select Cortex M0 Sys Tick clock source from external 32K crystal clock, external 12M crystal clock/2, HCLK/2, or internal 22M oscillator clock/2.

Parameter

u8ClkSrcSel [in]

- 1: External 32K clock

- 2: External 12M clock / 2
- 3: HCLK / 2
- 4~7: Internal 22M clock / 2

Include

Driver/DrvSYS.h

Return Value

- 0 Succeed
- < 0 Wrong parameter

DrvSYS_SetIPClockSource

Prototype

```
int32_t DrvSYS_SetIPClockSource(E_SYS_IP_CLKSRC eIpClkSrc, uint8_t u8ClkSrcSel);
```

Description

To select IP clock source include Watch Dog, ADC, Timer 0~3, UART, CAN, PWM10, and PWM32.

Parameter

eIpClkSrc [in]

E_SYS_WDG_CLKSRC / E_SYS_ADC_CLKSRC / E_SYS_TMR0_CLKSRC
E_SYS_TMR1_CLKSRC / E_SYS_TMR2_CLKSRC / E_SYS_TMR3_CLKSRC
E_SYS_UART_CLKSRC / E_SYS_CAN_CLKSRC / E_SYS_PWM10_CLKSRC
E_SYS_PWM32_CLKSRC.

u8ClkSrcSel [in]

IP's corresponding clock source.

Include

Driver/DrvSYS.h

Return Value

- 0 Succeed
- < 0 Wrong parameter

DrvSYS_SetClockDivider

Prototype

```
int32_t DrvSYS_SetClockDivider(E_SYS_IP_DIV eIpDiv , int32_t i32value);
```

Description

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

Parameter

eIpDiv [in]

E_SYS_ADC_DIV / E_SYS_CAN_DIV / E_SYS_UART_DIV
E_SYS_USB_DIV / E_SYS_HCLK_DIV

i32value [in]

Divide number

Include

Driver/DrvSYS.h

Return Value

0 Succeed
< 0 Wrong parameter

DrvSYS_SetOscCtrl

Prototype

int32_t DrvSYS_SetOscCtrl(E_SYS_OSC_CTRL eOscCtrl, int32_t i32Enable);

Description

To enable internal oscillator and external crystal include internal 10K and 22M oscillator, or external 32K and 12M crystal.

Parameter

eOscCtrl [in]

E_SYS_XTL12M / E_SYS_XTL32K / E_SYS_OSC22M / E_SYS_OSC10K.

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

0 Succeed
< 0 Wrong parameter

DrvSYS_EnablePWRWUInt

Prototype

```
void DrvSYS_EnablePWRWUInt(int32_t i32Enable, PWRWU_CALLBACK
pdwucallbackFn, int32_t i32enWUDelay);
```

Description

To enable power down wake up interrupt, and install its callback function if power down wake up is enable, and enable 64 clock cycle delay to wait the 12M crystal or 22M oscillator clock stable.

Parameter

i32Enable [in]

1: enable, 0: disable

pdwucallbackFn [in]

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

i32enWUDelay [in]

1: enable the 64 clock cycle delay, 0: disable the 64 clock cycle delay

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_EnablePowerDown

Prototype

```
void DrvSYS_EnablePowerDown(int32_t i32Enable);
```

Description

To enable or active system power down. If the PD_WAIT_CPU is set, the chip enters power down mode after PWR_DOWN_EN bit set; otherwise the chip keeps active until the CPU sleep mode is active and then the chip enters power down mode. When chip enters power down mode, the LDO, 12M crystal, and 22M oscillator will be disabled and the 32K and 10K won't be controlled.

Parameter

i32Enable [in]

1: Chip enters power down mode instant or wait CPU sleep command.

0: Chip operates in normal mode or CPU enters into idle mode.

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SetPowerDownWaitCPU

Prototype

```
void DrvSYS_SetPowerDownWaitCPU(int32_t i32Enable);
```

Description

To select the power down entry condition.

Parameter

i32Enable [in]

1: Chip enters power down mode when the both PWR_DOWN is set and CPU runs WFE/WFI instruction.

0: Chip enters power down mode when PWR_DOWN is set.

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SetPllSrc

Prototype

```
void DrvSYS_SetPllSrc(E_DRVSYS_SRC_CLK ePllSrc);
```

Description

To select PLL clock source include 22M oscillator and 12M crystal.

Parameter

ePllSrc [in]

E_DRVSYS_EXT_12M / E_DRVSYS_INT_22M

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_SetPLLPowerDown

Prototype

```
void DrvSYS_SetPLLPowerDown(int32_t i32Enable);
```

Description

To enable PLL power down mode.

Parameter

i32Enable [in]

1: PLL is in power down mode.

0: PLL is in normal mode.

Include

Driver/DrvSYS.h

Return Value

None

DrvSYS_GetEXTClock

Prototype

```
uint32_t DrvSYS_GetEXTClock(void);
```

Description

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The external crystal clock frequency

DrvSYS_GetPllContent

Prototype

```
uint32_t DrvSYS_GetPllContent(uint32_t u32ExtClockKHz, uint32_t u32PllClockKHz);
```

Description

To calculate the nearest PLL frequency to fit the target PLL frequency that is defined by u32PllClockKHz.

Parameter

u32ExtClockKHz [in]

The external clock frequency. The unit is in KHz.

u32PllClockKHz [in]

The target PLL clock frequency. The unit is in KHz.

Include

Driver/DrvSYS.h

Return Value

The PLL control register setting.

DrvSYS_GetPLLClock

Prototype

uint32_t DrvSYS_GetPLLClock(void);

Description

To get PLL clock frequency.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The PLL clock frequency in KHz

DrvSYS_GetHCLK

Prototype

uint32_t DrvSYS_GetHCLK(void);

Description

To get HCLK clock frequency.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The HCLK clock frequency in KHz

DrvSYS_Open

Prototype

```
int32_t DrvSYS_Open(uint32_t u32ExtClockKHz, uint32_t u32PllClockKHz);
```

Description

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

Parameter

u32ExtClockKHz [in]

The external clock frequency. The unit is in KHz.

u32PllClockKHz [in]

The target PLL clock frequency. The unit is in KHz.

Include

Driver/DrvSYS.h

Return Value

0 Succeed

3. DrvUART Introduction

3.1. UART Introduction

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

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

3.2. UART Feature

The UART includes following features:

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

4. DrvUART APIs Specification

4.1. Constant Definition

Table 4-1: UART driver constant definition.

Name	Value	Description
DRVUART_PORT0	0x000	UART port 0
DRVUART_PORT1	0x100000	UART port 1
DRVUART_LININT	0x100	LIN RX Break Field Detected Interrupt Enable
DRVUART_WAKEUPINT	0x40	Wake up interrupt enable
DRVUART_BUFERRINT	0x20	Buffer Error Interrupt Enable
DRVUART_TOUTINT	0x10	Time-out Interrupt.
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_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

4.2. Functions

DrvUART_Open

Prototype

```
int32_t
DrvUART_Open (
    UART_PORT port,
    UART_T *sParam
);
```

Description

The function is used to initialize UART

Parameter

Port [in]

Specify UART_PORT0/UART_PORT1

sParam [in]

Specify the property of UART. It includes

u32BaudRate: Baud rate

u8cParity: NONE/EVEN/ODD parity

u8cDataBits: DRVUART_DATA_BITS_5 , DRVUART_DATA_BITS_6,
DRVUART_DATA_BITS_7 or DRVUART_DATA_BITS_8

u8cStopBits: DRVUART_STOPBITS_1 , STOPBITS_1_5 or
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

The function is used disable UART clock, disable ISR and close UART function. .

Parameter

Port [in]

Specify UART_PORT0/UART_PORT1

Include

Driver/DrvUART.h

Return Value

None

DrvUART_EnableInt

Prototype

```
void    DrvUART_EnableInt (
    UART_PORT    u8Port
    uint32_t      u32InterruptFlag,
    PFN_DRVUART_CALLBACK pfncallback
);
```

Description

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

Parameter

u8Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

pfncallback [in]

Call back function pointer

Include

Driver/DrvUART.h

Return Value

None

Note

Use “/” to connect the interrupt flags to disable multiple interrupts simultaneously.

DrvUART_IsIntEnabled

Prototype

```
uint32_t
DrvUART_IsIntEnabled (
    UART_PORT    u16Port
    uint32_t      u32InterruptFlag
);
```

Description

The function is used to get the interrupt enable status

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

1: Enable.

0: Disable.

Note

It is recommended to query one interrupt at a time.

DrvUART_DisableInt

Prototype

```
void    DrvUART_DisableInt (
        UART_PORT    u16Port
        uint32_t      u32InterruptFlag
    );
```

Description

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

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

None

Note

Use “/” to connect the interrupt flags to disable multiple interrupts simultaneously.

DrvUART_ClearInt

Prototype

```
uint32_t
DrvUART_ClearInt (
    UART_PORT    u16Port
    uint32_t      u32InterruptFlag
);
```

Description

The function is used to clear UART Interrupt

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt.

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

E_SUCESS Success

DrvUART_GetIntStatus

Prototype

```
int8_t
DrvUART_GetIntStatus (
    UART_PORT    u16Port
    uint32_t      u32InterruptFlag
);
```

Description

The function is used to get the interrupt status

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt.

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

0: None.

1: Interrupt occurs.

Note

It is recommended to poll one interrupt at a time.

DrvUART_SetFIFOTriggerLevel

Prototype

```
void    DrvUART_SetFIFOTriggerLevel (
    UART_PORT    u16Port
    uint16_t      u32TriggerLevel
);
```

Description

The function is used to set Rx FIFO Trigger Level

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

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 (
    UART_PORT    u16Port,
    uint8_t      *pu8CTSValue,
    uint8_t      *pu8CTSChangeState
)
```

Description

The function is used to get CTS value and change state

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

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 (
    UART_PORT    u16Port,
    uint8_t      u8Value
)
```

Description

The function is used to set RTS information

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

u8Value [in]

Specify the RTS value

Include

Driver/DrvUART.h

Return Value

None

DrvUART_SetRxTimeOut

Prototype

```
void
DrvUART_SetRxTimeOut (
    UART_PORT    u16Port,
    uin8_t       u8TimeOut
)
```

Description

The function is used to set Rx Time Out Value

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

u8TimeOut [in]

Specify the Time out value

Include

Driver/DrvUART.h

Return Value

None

DrvUART_Read

Prototype

```
int32_t
DrvUART_Read (
    UART_PORT    u16Port
    uint8_t       *pu8RxBuf,
    uint32_t      u32ReadBytes
);
```

Description

The function is used to read Rx data from RX buffer

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

pu8RxBuf [out]

Specify the buffer to receive the data of receive FIFO.

u32ReadBytes [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

```
int32_t
DrvUART_Write(
    UART_PORT    u16Port
```

```
uint8_t      *pu8TxBuf,
uint32_t      u32WriteBytes
);
```

Description

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

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

pu8TxBuf [in]

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

u32WriteBytes [in]

Specify the byte number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success

E_DRVUART_TIMEOUT: FIFO polling timeout

DrvUART_SetPDMA

Prototype

```
void
DrvUART_SetPDMA (
    UART_PORT    u16Port
    uint16_t IsEnable
);
```

Description

The function is to Enable/Disable PDMA Channel

Parameter

u16Port [in]

Specify UART_PORT0/UART_PORT1

IsEnable [in]

Enable or Disable function.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success

DrvUART_OpenIRCR

Prototype

```
void
DrvUART_OpenIRCR (
    UART_PORT    u16Port
    STR_IRCR_T str_IRCR
);
```

Description

The function is to Set IRCR Control Register

Parameter

u16Port **[in]**
Specify UART_PORT0/UART_PORT1

str_IRCR **[in]**
The structure of IrDA
It includes of
u8cRXSelect : Select Rx function
u8cTXSelect : Select Tx function
u8cInvTX : Invert Tx signal
u8cInvRX : Invert Rx signal

Include

Driver/DrvUART.h

Return Value

None

DrvUART_OpenLIN

Prototype

```
void
DrvUART_OpenLIN (
```

```

UART_PORT    u16Port
uint16_t DIRECTION,
uint16_t BCNT
);

```

Description

The function is used to set LIN relative setting

Parameter

u16Port [in]
Specify UART_PORT0/UART_PORT1

DIRECTION [in]
Specify LIN direction : _MODE_TX or MODE_RX

BCNT [in]
Specify break count value

Include

Driver/DrvUART.h

Return Value

None

DrvUART_GetVersion

Prototype

```

int32_T
DrvUART_GetVersion (void);

```

Description

Return the current version number of driver.

Include

Driver/DrvUART.h

Return Value

Version number:

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

5. DrvTIMER Introduction

5.1. Timer Introduction

The timer module includes four channels, TIMER0~TIMER3 (TIMER0, TIMER1 at AHB1 and TIMER2 and TIMER3 at AHB2), 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.

5.2. Timer Feature

The timer processing unit includes following features:

- Compliant with the AMBA APB
- Each channel with a 8-bit pre-scale counter/24-bit counter and an interrupt request signal.
- Independent clock source for each channel (TCLK0,TCLK1,TCLK2,TCLK3)
- Maximum uninterrupted time = $(1 / 25 \text{ MHz}) * (2^8) * (2^{24} - 1)$ when TCLK = 25 MHz

6. DrvTIMER APIs Specification

6.1. Function

DrvTIMER_GetStatus

Prototype

```
int32_t DrvTIMER_GetStatus(TIMER_CHANNEL ch);
```

Description

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

Parameter

channel [in]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

Include

Driver/DrvTimer.h

Return Value

The data of register TISR.

DrvTIMER_SetTimerEvent

Prototype

```
int32_t DrvTIMER_SetTimerEvent(
    TIMER_CHANNEL channel,
    uint32_t uTimeTick,
    TIMER_CALLBACK pvFun ,
    uint32_t parameter
);
```

Description

This function is used to install a event to timer0, timer1, timer2, timer3.

Parameter

channel [in]

TMR0/ TMR1/ TMR2/ TMR3

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_t          uTimeEventNo
);
```

Description

This function is used to remove an installed event.

Parameter

channel [in]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

uTimeEventNo [in]

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

Include

Driver/DrvTimer.h

Return Value

None

DrvTIMER_ResetTicks

Prototype

```
Int32_t DrvTIMER_ResetTicks(TIMER_CHANNEL channel);
```

Description

This function is used to reset TIMER Tick..

Parameter

channel [in]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

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_t DrvTIMER_Open(
    TIMER_CHANNEL channel,
    uint32_t uTicksPerSecond,
    TIMER_OPMODE mode
);
```

Description

This function is used to set and start TIMER.

Parameter

channel [in]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

uTickPerSecond [in]

Tick per second.

Mode [in]

Operation Mode One-Shot / Periodic / Toggle. It could be ONESHOT_MODE, PERIODIC_MODE ,TOGGLE_MODE or UNINTERREUPT_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_t DrvTIMER_GetTicks(TIMER_CHANNEL channel);

Description

This function is used to return Timer ticks.

Parameter

channel [in]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

Include

Driver/DrvTimer.h

Return Value

Return the current ticks of TIMER0 , TIMER1 , TIMER2 or TIMER3.

DrvTIMER_Delay

Prototype

void DrvTIMER_Delay (uint32_t 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_t DrvTIMER_Ioctl(
    TIMER_CHANNEL channel,
    TIMER_CMD      uCmd,
    UINT32          uArg1,
);
```

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]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

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_t DrvTIMER_Close(TIMER_CHANNEL channel);
```

Description

The function is used to disable timer.

Parameter

channel [in]

TIMER channel TMR0/ TMR1/ TMR2/ TMR3.

Include

Driver/DrvTimer.h

Return Value

E_SUCCESS	Success.
E_DRVTIMER_CMD	Invalid timer channel.

DrvWDT_Open
Prototype

```
int32_t DrvWDT_Open(int32_t 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 LEVEL0 ~ 7 .

Include

Driver/DrvTimer.h

Return Value

E_SUCESS Success

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_Ioctl

Prototype

int32_T DrvWDT_Ioctl(int32_t hander ,WDT_CMD uCmd , uint32_t 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 Watch Dog Timer and Disable WDT Interrupt.

Parameter

None

Include

Driver/DrvTimer.h

Return Value

None

DrvTIMER_GetVersion

Prototype

```
uint32_t  
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

7. DrvGPIO Introduction

7.1. GPIO introduction

- 80 pins of General Purpose I/O are shared with special feature functions.
- The I/O type of each of I/O pins can be independently software configured as input, output, open-drain or quasi-bidirectional mode.
- 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_Open

Prototype

```
int32_t DrvGPIO_Open(
    DRVGPIO_PORT    port,
    int32_t          i32Bit,
    DRVGPIO_IO      mode,
);
```

Description

To configure the specified GPIO to use it.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

mode [in]

Set the IO to be IO_INPUT , IO_OUTPUT ,IO_OPENDRAIN or IO_QUASI.

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


```
int32_t DrvGPIO_Close(DRVGPIO_PORT port, int32_t i32Bit);
```

Description

To close the opened IO and reset its configurations.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_SetBit

Prototype

```
int32_t DrvGPIO_SetBit(DRVGPIO_PORT port, int32_t i32Bit);
```

Description

Set the specified IO bit to 1.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_ClrBit

Prototype

```
int32_t DrvGPIO_ClrBit(DRVGPIO_PORT port,int32_t i32Bit);
```

Description

Clear the specified IO bit to 0.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments.

DrvGPIO_GetBit

Prototype

```
int32_t DrvGPIO_GetBit(DRVGPIO_PORT port, int32_t i32Bit);
```

Description

Get the value of the specified IO bit.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified IO bit.

DrvGPIO_SetPortBits

Prototype

```
int32_t DrvGPIO_SetPortBits(DRVGPIO_PORT port, int32_t i32Data);
```

Description

Set the specified IO port.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE..

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 GPA, GPB , GPC , GPD , GPE.

Include

Driver/DrvGPIO.h

Return Value

The value of the IO port.

DrvGPIO_GetPortDoutBits

Prototype

int32_t 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 GPA, GPB , GPC , GPD , GPE.

Include

Driver/DrvGPIO.h

Return Value

The value of the GPIO DOUT register value.

E_DRVGPIO_ARGUMENT Wrong arguments.

DrvGPIO_EnableInt

Prototype

```
int32_t
DrvGPIO_EnableInt(
    DRVGPIO_PORT      port,
    INT32              i32Bit,
    DRVGPIO_INT_TYPE   triggerType,
    DRVGPIO_INT_MODE   mode
);
```

Description

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

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

triggerType [in]

Specified INT type. It could be IO_RISING, IO_FALLING and IO_BOTH_EDGE.

Mode [in]

Specified INT mode. It could be MODE_EDGE, IO_FALLING and MODE_LEVEL.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success.

E_DRVGPIO_ARGUMENT Wrong arguments

DrvGPIO_DisableInt

Prototype

```
int32_t DrvGPIO_DisableInt(DRVGPIO_PORT port,int32_t i32Bit);
```

Description

Disable the interrupt function of the specified IO bit

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_SetDebounceTime

Prototype

```
int32_t DrvGPIO_SetDebounceTime(int32_t i32DebounceClk, int8_t i8ClockSource));
```

Description

To set the debounce timing and select source.

Parameter

i32DebounceClk [in]

The debounce timing is $2^{(i32DebounceClk)} * \text{APB clock}$.

i8ClockSource [in]

The debounce clock source. It can be DBCLKSRC_HCLK or DBCLKSRC_10K.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_EnableDebounce

Prototype

```
int32_t DrvGPIO_EnableDebounce(
```

```

        DRVGPIOPORT port,
        int32_t i32Bit
    );
    
```

Description

To enable the debounce function of the specified GPIO interrupt.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIOPORT_ARGUMENT	Wrong arguments

DrvGPIO_DisableDebounce

Prototype

```

nt32_t DrvGPIO_DisableDebounce(DRVGPIOPORT port,int32_t i32Bit);
    
```

Description

To disable the debounce function of the specified GPIO interrupt.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIOPORT_ARGUMENT	Wrong arguments

DrvGPIO_GetDebounceTime

Prototype

```
int32_t DrvGPIO_GetDebounceTime(void);
```

Description

Get the debounce timing setting.

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

The debounce time setting..

DrvGPIO_GetIntStatus

Prototype

```
uint32_t DrvGPIO_GetIntStatus(void);
```

Description

This function is used to return a pointer to the GPIO interrupt status register.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

Include

Driver/DrvGPIO.h

Return Value

A value to the GPIO interrupt status register.

DrvGPIO_InitFunction

Prototype

```
int32_t DrvGPIO_InitFunction (DRVGPI_FUNC function);
```

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_PWM, FUNC_I2C0, FUNC_I2C1, FUNC_ADC,
 FUNC_EXTINT, FUNC_CPO, FUNC_TMR0, FUNC_TMR1, FUNC_TMR2,
 FUNC_TMR3, FUNC_UART0, FUNC_UART1, FUNC_COMP0,
 FUNC_COMP1, FUNC_CAN0, FUNC_CAN1, FUNC_SPI0, FUNC_SPI1,
 FUNC_SPI2, FUNC_SPI3.

option [in]

Select the different pin to output the specified function

Function	IO
FUNC_GPIO	All GPIO
FUNC_PWM	GPA12 ~ GPA15
FUNC_I2C0	GPA8 , GPA9
FUNC_I2C1	GPA10 , GPA11
FUNC_ADC	GPA0 ~ GPA7
FUNC_EXTINT	GPB14 , GPB15
FUNC_CPO	GPB12 , GPB13
FUNC_TMR0	GPB8
FUNC_TMR1	GPB9
FUNC_TMR2	GPB10
FUNC_TMR3	GPB11
FUNC_UART0	GPB0 ~ GPB3
FUNC_UART1	GPB4 ~ GPB7
FUNC_COMP0	GPC6 , GPC7
FUNC_COMP1	GPC14 ~ GPC15
FUNC_CAN0	GPD6 , GPD7
FUNC_CAN1	GPD14 , GPD15
FUNC_SPI0	GPC0 ~ GPC5
FUNC_SPI1	GPC8 ~ GPC13
FUNC_SPI2	GPD0 ~ GPD5
FUNC_SPI3	GPD8 ~ GPD13

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success.
 E_DRVGPIO_ARGUMENT Wrong arguments

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]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified IO bit

DrvGPIO_BitIsUsed

Prototype

int32_t DrvGPIO_BitIsUsed(DRVGPIO_PORT port, int32_t i32bit);

Description

The function is used to read flag to judge GPIO bit is used or not.

Parameter

None.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified IO bit

DrvGPIO_SetBitMask

Prototype

int32_t DrvGPIO_SetBitMask(DRVGPIO_PORT port, int32_t i32Bit);

Description

To set bits of GPIO port mask.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success.

DrvGPIO_ClrBitMask

Prototype

int32_t DrvGPIO_ClrBitMask(DRVGPIO_PORT port, int32_t i32Bit);

Description

To clear bits of GPIO port mask.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success.

DrvGPIO_SetPortMask

Prototype

int32_t DrvGPIO_SetPortMask(DRVGPIO_PORT port, uint32_t mask);

Description

To set GPIO port mask register.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

mask [in]

The data to mask to the specified IO port..

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS	Success.
E_DRVGPIO_ARGUMENT	Wrong arguments

DrvGPIO_ReadPortMask

Prototype

```
int32_t DrvGPIO_ReadPortMask(DRVGPIO_PORT port);
```

Description

To get current GPIO port mask setting.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

Include

Driver/DrvGPIO.h

Return Value

The value of Specified GPIO port mask register

DrvGPIO_InstallISR

Prototype

```
int32_t  
DrvGPIO_InstallISR(  
DRVGPIO_PORT port,  
int32_t i32Bit ,  
GPIO_CALLBACK pvFun ,  
uint32_t parameter);
```

Description

To install GPIO interrupt callback function.

Parameter

port [in]

Specified GPIO port. It could be GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

Specified bit of the IO port. It could be 0~15.

pvFun [in]

The event function pointer.

parameter [in]

he event number which contains this event.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS Success.
E_DRVGPIO_ARGUMENT Wrong arguments

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. DrvADC Introduction

9.1. ADC Introduction

The 10-bit Analog to Digital Converter (ADC) is a successive approximation type ADC with 8-channel inputs. 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.

UNITEC NUC100 series contain one 12-bit successive approximation analog-to-digital converters (SAR A/D converter) with 8 input channels. It takes 20 ADC clock cycles to convert one sample, and the maximum input clock to ADC is 20MHz at 5.0V. The A/D converter supports three operation modes: single, single-cycle scan and continuous scan mode. There are two kinds of scan mode: continuous mode and single cycle mode. The A/D converters can be started by software and external STADC/PB.8 pin.

Note that the analog input port pins must be configured as input type before ADC function is enabled.

9.2. ADC Feature

The Analog to Digital Converter includes following features:

- Analog input voltage range: 0~Vref (Max to 5.0V)
- 12-bits resolution and 10-bits accuracy is guaranteed
- Up to 8 analog input channels
- Maximum ADC clock frequency is 20MHz
- Three operating modes
 1. Single mode
 2. Single-cycle scan mode
 3. Continuous scan mode
- An A/D conversion can be started by
 1. Software write 1 to ADST bit
 2. External pin STADC
- Conversion result can be compared with specify value and user can select whether to generate an interrupt when conversion result matches the compare register setting
- The APIs include setting conditions and getting conversion data for ADC applications
- Channel 7 support 3 input source: external analog voltage, internal fixed bandgap voltage and internal temperature sensor output
- Support Self-calibration to minimum conversion error
- Support single end and differential input signal

10. DrvADC APIs Specification

10.1. Type Definition

Table 10-1: Type definition of ADC driver.

Type	Value	Description
ADC_INPUT_MODE	ADC_SINGLE_END (0)	ADC single end input
	ADC_DIFFERENTIAL (1)	ADC differential input
ADC_OPERATION_MODE	ADC_SINGLE_OP (0)	Single operation mode
	ADC_SINGLE_CYCLE_OP (1)	Single cycle scan mode
	ADC_CONTINUOUS_OP (2)	Continuous scan mode
ADC_CLK_SRC	EXT_12MHZ (0)	External 12MHz clock
	INT_PLL (1)	Internal PLL clock
	INT_RC22MHZ (2)	Internal 22MHz clock
ADC_EXT_TRI_COND	LOW_LEVEL (0)	Low level trigger
	HIGH_LEVEL (1)	High level trigger
	FALLING_EDGE (2)	Falling edge trigger
	RISING_EDGE (3)	Rising edge trigger
ADC_CH7_SRC	EXT_INPUT_SIGNAL (0)	External input signal
	INT_BANDGAP (1)	Internal bandgap voltage
	INT_TEMPERATURE_SENSOR (2)	Internal temperature sensor
ADC_COMP_CONDITION	LESS_THAN (0)	Less than compare data
	GREATER_OR_EQUAL (1)	Greater or equal to compare data

10.2. Macros

_DRVADC_CONV

Prototype

VOID _DRVADC_CONV (VOID);

Description

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

Include

Driver/DrvADC.h

Return Value

None.

10.3. Functions

DrvADC_Open

Prototype

```
void DrvADC_Open (
    ADC_INPUT_MODE InputMode,
    ADC_OPERATION_MODE OpMode,
    uint8_t u8ChannelSelBitwise,
    ADC_CLK_SRC ClockSrc,
    uint8_t u8AdcDivisor
);
```

Description

Enable the ADC function and complete the related settings.

Parameters

InputMode [in]

Specify the type of the analog input signal. It might be single-end or differential input.

OpMode [in]

Specify the operation mode. It might be single, single cycle scan or continuous scan mode.

u8ChannelSelBitwise [in]

Specify the input channels.

ClockSrc [in]

Specify the clock source of ADC clock.

u8AdcDivisor [in]

Determine the ADC clock frequency.

ADC clock frequency = ADC clock source frequency / (AdcDivisor + 1)

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_Close

Prototype

```
void DrvAdc_Close (void);
```

Description

Close ADC functions. Disable ADC, ADC engine clock and ADC interrupt.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_SetAdcChannel

Prototype

```
void DrvADC_SetAdcChannel (uint8_t u8ChannelSelBitwise);
```

Description

Set ADC input channels.

Parameters

u8ChannelSelBitwise [in]

Specify the analog input channels.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_ConfigAdcChannel7

Prototype

```
void DrvADC_ConfigAdcChannel7 (ADC_CH7_SRC Ch7Src);
```

Description

Select the input signal source of channel 7.

Parameters

Ch7Src [in]

Specify the analog input source.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_SetAdcInputMode

Prototype

```
void DrvADC_SetAdcInputMode (ADC_INPUT_MODE InputMode);
```

Description

Set the ADC input mode.

Parameters

InputMode [in]

Specify the input mode.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_SetAdcOperationMode

Prototype

```
void DrvADC_SetAdcOperationMode (ADC_OPERATION_MODE OpMode);
```

Description

Set the ADC operation mode.

Parameters

OpMode [in]

Specify the operation mode.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_SetAdcClkSrc

Prototype

```
void DrvADC_SetAdcClkSrc (ADC_CLK_SRC ClockSrc);
```

Description

Set the ADC clock source.

Parameters

ClockSrc [in]

Specify the ADC clock source.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_SetAdcDivisor

Prototype

```
void DrvADC_SetAdcDivisor (uint8_t u8AdcDivisor);
```

Description

Set the divisor value of ADC clock.

Parameters

u8AdcDivisor [in]

Specify the divisor value.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_EnableAdcInt

Prototype

```
void DrvADC_EnableAdcInt (
    DRVADC_ADC_CALLBACK Callback,
    uint32_t u32UserData
);
```

Description

Enable ADC 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

Return Value

None.

DrvADC_DisableAdcInt

Prototype

```
void DrvAdc_DisableAdcInt (void);
```

Description

Disable the ADC interrupt.

Parameters

None

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_EnableAdcmp0Int

Prototype

```
void DrvAdc_EnableAdcmp0Int (
    DRVADC_ADCMP0_CALLBACK Callback,
    uint32_t u32UserData
);
```

Description

Enable the ADC compare 0 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

Return Value

None.

DrvADC_DisableAdcmp0Int

Prototype

```
void DrvAdc_DisableAdcmp0Int (void);
```

Description

Disable the ADC compare 0 interrupt.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_EnableAdcmp1Int

Prototype

```
void DrvAdc_EnableAdcmp1Int (
    DRVADC_ADCMP1_CALLBACK Callback,
    uint32_t u32UserData
);
```

Description

Enable the ADC compare 1 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

Return Value

None.

DrvADC_DisableAdcmp1Int

Prototype

void DrvAdc_DisableAdcmp1Int (void);

Description

Disable the ADC compare 1 interrupt.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_GetConversionRate

Prototype

uint32_t DrvADC_GetConversionRate (void);

Description

To get the A/D conversion rate.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

Return the conversion rate.

DrvADC_ExtTriggerEnable

Prototype

```
void DrvADC_ExtTriggerEnable (ADC_EXT_TRI_COND TriggerCondition);
```

Description

Allow the external trigger pin (PB8) to be the trigger source of ADC.

Parameters

TriggerCondition [in]

Specify the trigger condition. The trigger condition could be low-level / high-level / falling-edge / positive-edge.

Include

Driver/DrvADC.h

Return Value

None

DrvADC_ExtTriggerDisable

Prototype

```
void DrvADC_ExtTriggerDisable (void);
```

Description

Prohibit the external ADC trigger.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_StartConvert

Prototype

```
void DrvADC_StartConvert(void);
```

Description

Start A/D converting.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_StopConvert

Prototype

void DrvADC_StopConvert(void);

Description

Stop A/D converting.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_IsConversionDone

Prototype

uint32_t DrvADC_IsConversionDone (void);

Description

Check whether the conversion action is finished or not.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

TURE	Conversion finished
FALSE	Under converting

DrvADC_GetConversionData

Prototype

```
uint32_t DrvADC_GetConversionData (uint8_t u8ChannelNum);
```

Description

To get the conversion result of the specified ADC channel.

Parameters

u8ChannelNum [in]

Specify the ADC channel.

Include

Driver/DrvADC.h

Return Value

12-bit conversion result.

If the channel number is out of range, returns E_DRVADC_CHANNELNUM.

DrvADC_PdmaEnable

Prototype

```
void DrvADC_PdmaEnable (void);
```

Description

Enable PDMA transfer.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None

DrvADC_PdmaDisable

Prototype

```
void DrvADC_PdmaDisable (void);
```

Description

Disable PDMA transfer.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None

DrvADC_IsDataValid

Prototype

UInt32_t DrvADC_IsDataValid (uint8_t u8ChannelNum);

Description

Check whether the conversion data is valid or not.

Parameters

u8ChannelNum [in]

Specify the ADC channel.

Include

Driver/DrvADC.h

Return Value

TURE	data is valid
FALSE	data is invalid

DrvADC_IsDataOverrun

Prototype

UInt32_t DrvADC_IsDataOverrun (uint8_t u8ChannelNum);

Description

Check whether the conversion data is overrun or not.

Parameters

u8ChannelNum [in]

Specify the ADC channel.

Include

Driver/DrvADC.h

Return Value

TURE	overrun
------	---------

FALSE non-overflow

DrvADC_Adcmp0Enable

Prototype

```
ERRCODE DrvADC_Adcmp0Enable (
    uint8_t u8CmpChannelNum,
    DC_COMP_CONDITION CmpCondition,
    uint16_t u16CmpData,
    uint8_t u8CmpMatchCount
);
```

Description

Enable the ADC result monitor 0. Configure the necessary settings and enable the result monitor function.

Parameters

u8CmpChannelNum [in]

Specify the channel number that wants to compare.

CmpCondition [in]

Specify the compare condition.

u16CmpData [in]

Specify the compare data.

u8CmpMatchCount [in]

Specify the compare match count.

Include

Driver/DrvADC.h

Return Value

E_SUCCESS	Success. The compare function is enabled.
E_DRVADC_ARGUMENT	One of the input arguments is out of the range

DrvADC_Adcmp0Disable

Prototype

```
void DrvADC_Adcmp0Disable (void);
```

Description

Disable the ADC result monitor 0.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_Adcmp1Enable

Prototype

```
ERRCODE DrvADC_Adcmp1Enable (
    uint8_t u8CmpChannelNum,
    DC_COMP_CONDITION CmpCondition,
    uint16_t u16CmpData,
    uint8_t u8CmpMatchCount
);
```

Description

Enable the ADC result monitor 1. Configure the necessary settings and enable the result monitor function.

Parameters

u8CmpChannelNum [in]

Specify the channel number that wants to compare.

CmpCondition [in]

Specify the compare condition.

u16CmpData [in]

Specify the compare data.

u8CmpMatchCount [in]

Specify the compare match count.

Include

Driver/DrvADC.h

Return Value

E_SUCCESS	Success. The compare function is enabled.
E_DRVADC_ARGUMENT	One of the input arguments is out of the range

DrvADC_Adcmp1Disable

Prototype

void DrvADC_Adcmp1Disable (void);

Description

Disable the ADC result monitor 1.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_SelfCalEnable

Prototype

void DrvADC_SelfCalEnable (void);

Description

Enable the self calibration function.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_IsCalDone

Prototype

uint32_t DrvADC_IsCalDone (void);

Description

Check whether the self calibration action is finished or not.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

TURE the self calibration action is finished.
FALSE the self calibration action is in progress.

DrvADC_SelfCalDisable

Prototype

void DrvADC_SelfCalDisable (void);

Description

Disable the self calibration function.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

DrvADC_GetVersion

Prototype

uint32_t DrvAdc_GetVersion (void);

Description

Return the current version number of driver.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

Version number:

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

11. DrvSPI Introduction

11.1. SPI Introduction

The Serial Peripheral Interface (SPI) is a synchronous serial data communication protocol which operates in full duplex mode. Devices communicate in master/slave mode with 4-wire bi-direction interface. NUC100 series contain four sets of SPI controller performing a serial-to-parallel conversion on data received from a peripheral device, and a parallel-to-serial conversion on data transmitted to a peripheral device. Each SPI set can drive up to 2 external peripherals. It also can be driven as the slave device when the SLAVE bit (CNTRL[18]) is set.

Each controller can generate an individual interrupt signal when data transfer is finished and can be cleared by writing 1 to the respective interrupt flag. The active level of device/slave select signal can be programmed to low active or high active on SSR[SS_LVL] bit, which depends on the connected peripheral. Writing a divisor into DIVIDER register can program the frequency of serial clock output when it is as the master. If the VARCLK_EN bit in SPI_CNTRL[23] is enabling, the serial clock can be set as two programmable frequencies which are defined in DIV and DIV2. The format of the variable frequency is defined in VARCLK.

This master/slave core contains two 32-bit transmit/receive buffers, and can provide burst mode operation. It supports variable length transfer and the maximum transmitted/received length can be up to 64 bits.

The controller also supports two bits data mode which is defined in the SPI_CNTRL[22]. When the TWOB bit, in SPI_CNTRL[22], is enabling, it can transmits and receives two bit serial data out/in the serial buffer. The 1st bit channel transmits the data from SPI_TX0 and receives the data into SPI_RX0. The 2nd bit channel transmits the data from SPI_TX1 and receives the data into SPI_RX1.

11.2. General Feature

- Four sets of SPI controller
- Support master/slave/Joystick mode operation
- Support 1, 2 bit serial data IN/OUT
- Configurable data length of transfer word up to 32 bits
- Variable output serial clock frequency in master mode
- Provide burst mode operation, transmit/receive can be executed up to two times in one transfer
- MSB or LSB first data transfer
- 2 slave/device select lines when it is set as the master mode, and 1 slave/device select line when it is set as slave mode
- Byte Sleep Mode

12. DrvSPI APIs Specification

12.1. Constant Definition

Type	Value	Description
E_DRVSPI_PORT	eDRVSPI_PORT0 (0)	SPI port 0
	eDRVSPI_PORT1 (1)	SPI port 1
	eDRVSPI_PORT2 (2)	SPI port 2
	eDRVSPI_PORT3 (3)	SPI port 3
E_DRVSPI_MODE	eDRVSPI_MASTER (0)	SPI Master Mode
	eDRVSPI_SLAVE (1)	SPI Slave Mode
	eDRVSPI_JOYSTICK (2)	SPI Joystick Mode
E_DRVSPI_TRANS_TYPE	eDRVSPI_TYPE0 (0)	SPI Transfer Type 0
	eDRVSPI_TYPE1 (1)	SPI Transfer Type 1
	eDRVSPI_TYPE2 (2)	SPI Transfer Type 2
	eDRVSPI_TYPE3 (3)	SPI Transfer Type 3
	eDRVSPI_TYPE4 (4)	SPI Transfer Type 4
	eDRVSPI_TYPE5 (5)	SPI Transfer Type 5
	eDRVSPI_TYPE6 (6)	SPI Transfer Type 6
	eDRVSPI_TYPE7 (7)	SPI Transfer Type 7
E_DRVSPI_ENDIAN	eDRVSPI_LSB_FIRST(0)	Send LSB First
	eDRVSPI_MSB_FIRST(1)	Send MSB First
E_DRVSPI_SSLTRIG	eDRVSPI_EDGE_TRIGGER (0)	Edge Trigger
	eDRVSPI_LEVEL_TRIGGER (1)	Level Trigger
E_DRVSPI_SS_ACT_TYPE	eDRVSPI_ACTIVE_LOW_FALLING (0)	Low-level/Falling-edge active
	eDRVSPI_ACTIVE_HIGH_RISING (1)	High-level/Rising-edge active
E_DRVSPI_SLAVE_SEL	eDRVSPI_NONE (0)	All slave select pins are de-selected
	eDRVSPI_SS0 (1)	SS0 active
	eDRVSPI_SS1 (2)	SS1 active
	eDRVSPI_SS0_SS1 (3)	Both SS0 and SS1 are selected

Type	Value	Description
E_DRVSPi_JOYSTiCK_INT_FLAG	eDRVSPi_JOYSTiCK_CS_ACTIVE (0)	Joystick CS active
	eDRVSPi_JOYSTiCK_DATA_READY (1)	Joystick 8-Byte Data Ready
	eDRVSPi_JOYSTiCK_CS_DEACT (2)	Joystick CS de-active
	eDRVSPi_JOYSTiCK_NONE (3)	No event in Joystick mode
E_DRVSPi_JOYSTiCK_RW_MODE	eDRVSPi_JOYSTiCK_TRANSMIT_MODE (0)	Joystick Transmit Mode
	eDRVSPi_JOYSTiCK_RECEIVE_MODE (1)	Joystick Receive Mode
E_DRVSPi_DMA_MODE	eDRVSPi_TX_DMA (0)	Tx DMA
	eDRVSPi_RX_DMA (1)	Rx DMA

12.2. Functions

DrvSPi_Open

Prototype

```

ERRCODE
DrvSPi_Open(
    E_DRVSPi_PORT eSpiPort,
    E_DRVSPi_MODE eMode,
    E_DRVSPi_TRANS_TYPE eType,
    int32_t i32BitLength
);

```

Description

This function is used to open SPI module. It decides the SPI to work on master or slave or Joystick mode, SPI bus timing and bit length per transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eMode [in]

To work in Master (eDRVSPi_MASTER) / Slave (eDRVSPi_SLAVE) / Joystick (eDRVSPi_JOYSTiCK) mode

eType [in]

Transfer types, i.e the bus timing. it could be eDRVSPi_TYPE0~eDRVSPi_TYPE7.

i32BitLength [in]

Bit length per transaction.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS	Success.
E_DRVSPi_ERR_INIT	The specified SPI port has been opened before.
E_DRVSPiMS_ERR_BIT_LENGTH	The bit length is out of range.
E_DRVSPiMS_ERR_BUSY	The specified SPI port is in busy status.

DrvSPi_Close

Prototype

```
void DrvSPi_Close (
    E_DRVSPi_PORT eSpiPort
);
```

Description

Close the specified SPI module and disable the SPI interrupt.

Parameters

eSpiPort [in]
Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPi_Set2BitSerialDataIOMode

Prototype

```
void DrvSPi_Set2BitSerialDataIOMode (
    E_DRVSPi_PORT eSpiPort,
    BOOL bEnable
);
```

Description

Set 2-bit serial data I/O mode.

Parameters

eSpiPort [in]
Specify the SPI port.

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_SetEndian

Prototype

```
void DrvSPI_SetEndian (
    E_DRVSPi_PORT eSpiPort,
    E_DRVSPi_ENDIAN eEndian
);
```

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eEndian [in]

Specify LSB first or MSB first.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_SetBitLength

Prototype

```
ERRCODE
DrvSPI_SetBitLength(
    E_DRVSPi_PORT eSpiPort,
    int32_t i32BitLength
);
```

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

i32BitLength [in]

Specify the bit length (1~32 bits).

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS

Success.

E_DRVSPI_ERR_BIT_LENGTH

The bit length is out of range.

DrvSPI_SetByteSleep

Prototype

ERRCODE

```
DrvSPI_SetByteSleep(
    E_DRVSPI_PORT eSpiPort,
    BOOL bEnable
);
```

Description

This function is used to enable/disable Byte Sleep function

Parameters

eSpiPort [in]

Specify the SPI port.

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS

Success.

E_DRVSPIMS_ERR_BIT_LENGTH

The bit length is not 32 bits.

DrvSPI_SetByteEndian

Prototype

```

ERRCODE
DrvSPI_SetByteEndian (
    E_DRVSPI_PORT eSpiPort,
    BOOL bEnable
);

```

Description

This function is used to enable/disable Byte Endian function

Parameters

eSpiPort [in]

Specify the SPI port.

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS	Success.
E_DRVSPIMS_ERR_BIT_LENGTH	The bit length MUST be 8/16/24/32.

DrvSPI_SetTriggerMode

Prototype

```

void DrvSPI_SetTriggerMode (
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_SSLTRIG eSSTriggerMode
);

```

Description

Set the trigger mode of slave select pin.

Parameters

eSpiPort [in]

Specify the SPI port.

eSSTriggerMode [in]

Specify the trigger mode. (eDRVSPI_EDGE_TRIGGER or eDRVSPI_LEVEL_TRIGGER)

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_SetSlaveSelectActiveLevel

Prototype

```
void DrvSPI_SetSlaveSelectActiveLevel (
    E_DRVSPi_PORT eSpiPort,
    E_DRVSPi_SS_ACT_TYPE eSSActType
);
```

Description

Set the active level of slave select.

Parameters

eSpiPort [in]

Specify the SPI port.

eSSActType [in]

Select the active type of slave select pin.

eDRVSPi_ACTIVE_LOW_FALLING:

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

eDRVSPi_ACTIVE_HIGH_RISING:

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

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_GetLevelTriggerStatus

Prototype

```
BOOL
DrvSPI_GetLevelTriggerStatus (
    E_DRVSPi_PORT eSpiPort
);
```

Description

This function is used to get the level-trigger transmission status

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

TRUE: The received number and received bits met the requirement which defines in TX_NUM and TX_BIT_LEN among one transaction.

FALSE: The transaction number or the received bit length of one transaction doesn't meet the requirement in one transfer.

DrvSPI_EnableAutoCS

Prototype

```
void DrvSPI_EnableAutoCS (
    E_DRV_SPI_PORT eSpiPort,
    E_DRV_SPI_SLAVE_SEL eSlaveSel
);
```

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 SPI 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.

Parameters

eSpiPort [in]

Specify the SPI port.

eSlaveSel [in]

Select the slave select pins which will be used.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_DisableAutoCS

Prototype

```
void DrvSPI_DisableAutoCS (
    E_DRVSPi_PORT eSpiPort
);
```

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.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_SetCS

Prototype

```
void DrvSPI_SetCS(
    E_DRVSPi_PORT eSpiPort,
    E_DRVSPi_SLAVE_SEL eSlaveSel
);
```

Description

Configure the slave select pins.

Parameters

eSpiPort [in]

Specify the SPI port.

eSlaveSel [in]

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

In manual slave select operation, the specified slave select pins will be set to active state. It could be eDRVSPi_NONE, eDRVSPi_SS0, eDRVSPi_SS1 or eDRVSPi_SS0_SS1.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_ClrCS

Prototype

```
void DrvSPI_ClrCS(
    E_DRV_SPI_PORT eSpiPort,
    E_DRV_SPI_SLAVE_SEL eSlaveSel
);
```

Description

Set the specified slave select pins to inactive state.

Parameters

eSpiPort [in]

Specify the SPI port.

eSlaveSel [in]

Specify slave select pins.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_Busy

Prototype

```
BOOL
DrvSPI_Busy(
    E_DRV_SPI_PORT eSpiPort
);
```

Description

Check the busy status of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

TURE: The SPI port is in busy.

FALSE: The SPI port is not in busy.

DrvSPI_BurstTransfer

Prototype

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

Description

Configure the burst transfer settings.

Parameters

eSpiPort [in]

Specify the SPI port.

i32BurstCnt [in]

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

i32Interval [in]

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

Include

Driver/DrvSPI.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.

DrvSPI_SetClock

Prototype

```
uint32_t
DrvSPI_SetClock(
```

```
E_DRV_SPI_PORT eSpiPort,
uint32_t u32Clock1,
uint32_t u32Clock2
);
```

Description

Configure the SPI clock.

Parameters

eSpiPort [in]

Specify the SPI port.

u32Clock1 [in]

Specify the SPI clock rate in Hz. It's the clock rate of SPI base clock or variable clock 1.

u32Clock2 [in]

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

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

The actual value of divisor 1 is returned. The actual clock may be different to the target SPI clock due to hardware limitation.

DrvSPI_GetClock1

Prototype

```
uint32_t
DrvSPI_GetClock1(
    E_DRV_SPI_PORT eSpiPort
);
```

Description

Get the SPI engine clock rate in Hz.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

The current SPI bus clock frequency in Hz.

DrvSPI_GetClock2

Prototype

```
uint32_t
DrvSPI_SetClock2(
    E_DRV_SPI_PORT eSpiPort
);
```

Description

Get the clock rate of variable clock 2 in Hz.

Parameters

eSpiPort [in]
Specify the SPI port.

Include

Driver/DrvSPI.h
Driver/DrvSYS.h

Return Value

The frequency of variable clock 2. (Hz.)

DrvSPI_SetVariableClockPattern

Prototype

```
void
DrvSPI_SetVariableClockPattern (
    E_DRV_SPI_PORT eSpiPort,
    uint32_t u32Pattern
);
```

Description

If the bit pattern of VARCLK is '0', the output frequency of SPICLK is according to the value of DIVIDER.

If the bit pattern of VARCLK is '1', the output frequency of SPICLK is according to the value of DIVIDER2.

Parameters

eSpiPort [in]

Specify the SPI port.

u32Pattern [in]

Specify the variable clock pattern.

Include

Driver/DrvSPI.h

Return Value

None.

DrvSPI_SetVariableClockFunction

Prototype

```
void
DrvSPI_SetVariableClockFunction (
    E_DRV_SPI_PORT eSpiPort,
    BOOL bEnable
);
```

Description

Set the variable clock function.

Parameters

eSpiPort [in]

Specify the SPI port.

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvSPI.h

Return Value

None.

DrvSPI_EnableInt

Prototype

```
void DrvSPI_EnableInt(
    E_DRV_SPI_PORT eSpiPort,
    PFN_DRV_SPI_CALLBACK pfnCallback,
    uint32_t u32UserData
);
```

Description

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

Parameters

u16Port [in]

Specify the SPI port.

pfnCallback [in]

The callback function of the corresponding SPI interrupt.

u32UserData [in]

The parameter which will be passed to the callback function.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_DisableInt

Prototype

```
void DrvSPI_DisableInt(
    E_DRV_SPI_PORT eSpiPort
);
```

Description

Disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_SingleRead

Prototype

```
BOOL
DrvSPI_SingleRead(
```

```
E_DRV_SPI_PORT eSpiPort,
uint32_t *pu32Data
);
```

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

pu32Data [out]

Store the data got from the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data is valid.

FALSE: The data stored in pu32Data is invalid.

DrvSPI_SingleWrite

Prototype

```
BOOL
DrvSPI_SingleWrite (
    E_DRV_SPI_PORT eSpiPort,
    uint32_t *pu32Data
);
```

Description

Write data to SPI bus and trigger SPI to start transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

pu32Data [in]

Store the data which will be transmitted through the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data has been transferred.

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

DrvSPI_BurstRead

Prototype

```

BOOL
DrvSPI_BurstRead (
    E_DRV_SPI_PORT eSpiPort,
    uint32_t *pu32Buf
);
    
```

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

pu32Buf [out]

Store the data got from the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf is valid.

FALSE: The data stored in pu32Buf is invalid.

DrvSPI_BurstWrite

Prototype

```

BOOL
DrvSPI_BurstWrite (
    E_DRV_SPI_PORT eSpiPort,
    uint32_t *pu32Buf
);
    
```

Description

Write two words of data to SPI bus and then trigger SPI to start transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

pu32Buf [in]

Store the data which will be transmitted through the SPI bus.

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf has been transferred.

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

DrvSPI_DumpRxRegister

Prototype

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

Description

Read data from Rx registers. This function will not trigger another data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

pu32Buf [out]

Store the data got from Rx registers.

u32DataCount [in]

The count of data read from Rx registers.

Include

Driver/DrvSPI.h

Return Value

The count of data actually read from Rx registers.

DrvSPI_SetTxRegister

Prototype

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

Description

Write data to Tx registers. This function will not trigger another data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

pu32Buf [in]

Store the data which will be written to Tx registers.

u32DataCount [in]

The count of data written to Tx registers.

Include

Driver/DrvSPI.h

Return Value

The count of data actually written to Tx registers.

DrvSPI_SetGo

Prototype

```
void DrvSPI_SetGo (
    E_DRVSPi_PORT eSpiPort
);
```

Description

Set the GO_BUSY bit to trigger a SPI data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_GetJoyStickIntType

Prototype

```
E_DRVSPi_JOYSTICK_INT_FLAG
DrvSPI_GetJoyStickIntType (
    E_DRVSPi_PORT eSpiPort
);
```

Description

Get interrupt flag of JOYSTICK mode.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

eDRVSPI_JOYSTICK_DATA_READY: 8-byte data available in the buffer.

eDRVSPI_JOYSTICK_CS_ACTIVE: Chip Select is activated.

eDRVSPI_JOYSTICK_CS_DEACT: Chip Select is de-activated.

eDRVSPI_JOYSTICK_NONE: None.

DrvSPI_SetJoyStickStatus

Prototype

```
void DrvSPI_SetJoyStickStatus (
    E_DRVSPI_PORT eSpiPort,
    BOOL bReady
);
```

Description

Set the JoyStick status to ready or not ready.

Parameters

eSpiPort [in]

Specify the SPI port.

bReady [in]

TRUE: The SPI is ready to transfer data.

FALSE: The SPI is not ready to transfer data.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_GetJoyStickMode

Prototype

```
E_DRVSPI_JOYSTICK_RW_MODE
DrvSPI_GetJoyStickMode (
    E_DRVSPI_PORT eSpiPort
```

```
);
```

Description

Get the JoyStick operation mode.

Parameters

eSpiPort [in]

Specify the SPI port.

Include

Driver/DrvSPI.h

Return Value

The Joystick operation mode:

eDRVSPI_JOYSTICK_TRANSMIT_MODE: Master writes data to slave.

eDRVSPI_JOYSTICK_RECEIVE_MODE: Master read data from slave.

DrvSPI_StartPMDA

Prototype

```
void DrvSPI_StartPDMA (
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_DMA_MODE eDmaMode,
    BOOL bEnable
);
```

Description

Configure the DMA settings.

Parameters

eSpiPort [in]

Specify the SPI port.

eDmaMode [in]

Specify the DMA mode.

eEnable [in]

True: Enable DMA.

False: Disable DMA.

Include

Driver/DrvSPI.h

Return Value

None

DrvSPI_GetVersion

Prototype

```
uint32_t
DrvSPI_GetVersion (void);
```

Description

Get the version number of NUC100 SPI driver.

Parameters

None.

Include

Driver/DrvSPI.h

Return Value

Version number:

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

13. DrvI2C Introduction

13.1. 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.

13.2. Feature

The I2C includes following features:

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

14. DrvI2C APIs Specification

14.1. Functions

DrvI2C_Open

Prototype

```
int32_t DrvI2C_Open(E_I2C_PORT port, uint32_t clock_Hz, uint32_t baudrate_Hz);
```

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

clock_Hz [in]

To input I2C source clock. The unit is Hz.

baudrate_Hz [in]

To configure I2C source clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_Close

Prototype

```
int32_t DrvI2C_Close(E_I2C_PORT port);
```

Description

To close the I2C hardware.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_SetClock

Prototype

```
int32_t DrvI2C_SetClock(E_I2C_PORT port, uint32_t clock_Hz, uint32_t baudrate_Hz);
```

Description

To configure the I2C bus clock.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

clock_Hz [in]

To input I2C source clock. The unit is Hz.

baudrate_Hz [in]

To configure I2C source clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_GetClock

Prototype

```
uint32_t DrvI2C_GetClock(E_I2C_PORT port, uint32_t u32clock);
```

Description

To get the I2C bus clock.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

u32clock [in]

To input I2C source clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

I2C bus clock

DrvI2C_SetAddress

Prototype

```
int32_t DrvI2C_SetAddress(E_I2C_PORT port, uint8_t slaveNo, uint8_t slave_addr,
uint8_t GC_Flag);
```

Description

To set the I2C slave address.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

slaveNo [in]

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

slave_addr [in]

To set 7-bit slave address.

GC_Flag [in]

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_SetAddressMask

Prototype

```
int32_t DrvI2C_SetAddressMask(E_I2C_PORT port, uint8_t slaveNo, uint8_t
slaveAddrMask);
```

Description

To set the I2C slave address mask.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

slaveNo [in]

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

slaveAddrMask [in]

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_GetStatus

Prototype

uint32_t DrvI2C_GetStatus(E_I2C_PORT port);

Description

To get the I2C status. There are 26 status code.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

I2C status code

DrvI2C_WriteData

Prototype

void DrvI2C_WriteData(E_I2C_PORT port, uint8_t u8data);

Description

To set byte data to send.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

u8data [in]

Byte data.

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_ReadData

Prototype

uint8_t DrvI2C_ReadData(E_I2C_PORT port);

Description

To read the last data from I2C bus.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

Last byte data

DrvI2C_Ctrl

Prototype

void DrvI2C_Ctrl(E_I2C_PORT port, uint8_t start, uint8_t stop, uint8_t intFlag, uint8_t ack);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

start [in]

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

stop [in]

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

intFlag [in]

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

ack [in]

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

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_GetIntFlag

Prototype

uint8_t DrvI2C_GetIntFlag(E_I2C_PORT port);

Description

To get I2C interrupt flag status.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

Interrupt status (1 or 0)

DrvI2C_ClearIntFlag

Prototype

void DrvI2C_ClearIntFlag(E_I2C_PORT port);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

None

DrvI2C_EnableInt

Prototype

```
int32_t DrvI2C_EnableInt(E_I2C_PORT port);
```

Description

To enable I2C interrupt bit and corresponding bit of NVIC.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_DisableInt

Prototype

```
int32_t DrvI2C_DisableInt(E_I2C_PORT port);
```

Description

To disable I2C interrupt bit and corresponding bit of NVIC.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_InstallCallBack

Prototype

```
int32_t DrvI2C_InstallCallBack(E_I2C_PORT port, E_I2C_CALLBACK_TYPE Type,  
I2C_CALLBACK callbackfn);
```

Description

To install I2C call back function in I2C interrupt handler.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Type [in]

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

I2CFUNC: For normal I2C condition

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

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

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

callbackfn [in]

Call back function name.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

<0 Failed

DrvI2C_UninstallCallBack

Prototype

int32_t DrvI2C_UninstallCallBack(E_I2C_PORT port, E_I2C_CALLBACK_TYPE Type);

Description

To uninstall I2C call back function in I2C interrupt handler.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Type [in]

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

I2CFUNC: For normal I2C condition

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

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

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed
<0 Failed

DrvI2C_EnableTimeoutCount

Prototype

```
int32_t DrvI2C_EnableTimeoutCount(E_I2C_PORT port, int32_t i32enable, uint8_t
u8div4);
```

Description

To enable 14-bit time-out counter.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

i32enable [in]

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

u8div4 [in]

To enable DIV4 function. The counter will be divider by 4 if counter enabled.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

DrvI2C_ClearTimeoutFlag

Prototype

```
void DrvI2C_ClearTimeoutFlag(E_I2C_PORT port);
```

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

None

15. DrvRTC Introduction

15.1. 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.

15.2. 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

16. DrvRTC APIs Specification

16.1. Constant Definition

Table 16-1: The constant definitions of RTC driver.

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

16.2. Functions

DrvRTC_SetFrequencyCompensation

Prototype

```
int32_t
DrvRTC_SetFrequencyCompensation (
    float fnumber;
);
```

Description

Set Frequency Compensation Data

Parameter

fnumber [in]

Specify the Compensation value.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FCR_VALUE: Wrong Compensation value

DrvRTC_WriteEnable

Prototype

```
int32_t
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

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

Description

Initial RTC

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_Open

Prototype

```
int32_t  
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

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

Description

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

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

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

Description

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

Parameter

eTime [in]

Specify the current/alarm time to be written.

DRVRTC_CURRENT_TIME: Current time

DRVRTC_ALARM_TIME: Alarm time

*sPt [in]

Specify the data to write to RTC. It includes

u8cClockDisplay : DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24

u8cAmPm : DRVRTC_AM / DRVRTC_PM

u32cSecond : Second value

u32cMinute : Minute value

u32cHour : Hour value

u32cDayOfWeek : Day of week

u32cDay : Day value

u32cMonth : Month value

u32Year : Year value

pfnAlarmCallBack : The alarm call back function pointer

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

DrvRTC_IocI

Prototype

```
int32_t
DrvRTC_IocI (
    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 current time

DRVRTC_IOC_SET_ALARM_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

int32_t

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

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

Description

Return the current version number of driver.

Include

Driver/DrvRTC.h

Return Value

Version number :

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

17. DrvCAN Introduction

17.1. CAN Introduction

The Controller Area Network (CAN) is a serial communications protocol which is multi-master and it efficiently supports distributed real-time control with very high level of security. In CAN systems, a node does not make use of any information about the system configuration (station addresses). Any Nodes can be added to the CAN network without requiring any change in the software or hardware of any node.

17.2. CAN Feature

The CAN processing unit includes following features:

- CAN 2.0B protocol compatibility
- AMBA APB bus interface compatible
- Multi-master node
- Support 11-bit identifier as well as 29-bit identifier
- Bit rates up to 1Mbits/s
- NRZ bit coding
- Error detection: bit error, stuff error, form error, 15-bit CRC detection, and acknowledge error
- Listen only mode (no acknowledge, no active error flags)
- Acceptance filter extension (4-byte code, 4-byte mask)
- Error interrupt for each CAN-bus error
- Extended receive buffer (8-byte FIFO)
- Wakeup function

18. DrvCAN APIs Specification

18.1. Function

DrvCAN_Open

Prototype

```
int32_t DrvCAN_Open(CAN_PORT port,int32_t Clock );
```

Description

The function is used to open CAN intial setting.

Parameter

port [in]

DRVCAN_PORT0 / DRVCAN_PORT1

Clock [in]

BITRATE_100K_6M , BITRATE_500K_6M ,BITRATE_1000K_6M
 BITRATE_100K_12M,BITRATE_500K_12M,BITRATE_1000K_12M
 BITRATE_100K_24M,BITRATE_500K_24M,BITRATE_1000K_24M
 BITRATE_100K_48M,BITRATE_500K_48M,BITRATE_1000K_48M
 Or Userself configure value

Include

Driver/DrvCAN.h

Return Value

E_SUCEESS

DrvCAN_DisableInt

Prototype

```
int32_t DrvCAN_DisableInt (
    CAN_PORTL    port,
    int32_t      u32InterruptFlag
);
```

Description

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

Parameter

port [in]

CAN_PORT0 / CAN_PORT1

u32InterruptFlag [in]

INT_BEI/INT_ALI/INT_WUI/INT_TI/INT_RI.

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS

DrvCAN_EnableInt

Prototype

```
int32_t DrvCAN_EnableInt (
    CAN_PORT      port,
    u32InterruptFlag  INT_BEI/INT_ALI/INT_WUI/INT_TI/INT_RI
    PFN_DRVCAN_CALLBACK  pfncallback
);
```

Description

The function is used to enable CAN Interrupt and install the call back function.

Parameter

port [in]

CAN channel CAN_PORT0 / CAN_PORT1.

u32InterruptFlag [in]

Interrupt Flag INT_BEI/INT_ALI/INT_WUI/INT_TI/INT_RI.

pfncallback [in]

Callback function pointer.

Include

Driver/DrvCAN.h

Return Value

None

DrvCAN_GetErrorStatus

Prototype

```
int32_t DrvCAN_GetErrorStatus (
    CAN_PORT      port,
    DRVCAN_ERRFLAG u32ErrorFlag
)
```

Description

The function is used to Get CAN Error Status.

Parameter

port [in]

CAN channel CAN_PORT0 / CAN_PORT1.

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS Success

DrvCAN_ReadMsg

Prototype

```
STR_CAN_T DrvCAN_ReadMsg(CAN_PORT port);
```

Description

The function is used to get CAN RX information.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

CAN structure

DrvCAN_SetAcceptanceFilter

Prototype

```
int32_t DrvCAN_SetAcceptanceFilter (
    CAN_PORT      port,
```

```
int32_t id_Filter
);
```

Description

The function is used to Set a Accept ID filter.

Parameter

port [in]

port CAN_PORT0 / CAN_PORT1

id_Filter [in]

The data to write to the specified ID Filter

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS Success.

DrvCAN_SetMaskFilter

Prototype

```
uint32_t DrvCAN_SetMaskFilter (CAN_PORT port,int32_t id_Filter );
```

Description

The function is used to set mask ID filter.

Parameter

port [in]

port CAN_PORT0 / CAN_PORT1

id_Filter [in]

The data to write to the specified Mask Filter

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS Success

DrvCAN_WaitReady

Prototype

```
int32_t DrvCAN_WaitReady (CAN_PORT port);
```

Description

The function is used to check bus is idle

Parameter

port [in]

port CAN_PORT0 / CAN_PORT1

Include

Driver/DrvCAN.h

Return Value

None

DrvCAN_WriteMsg

Prototype

```
int32_t DrvCAN_WriteMsg(CAN_PORT port,STR_CAN_T *Msg);
```

Description

The function is set CAN information and send to CAN BUS

Parameter

port [in]

port CAN_PORT0 / CAN_PORT1

Msg [in]

Specify the property of CAN. It includes

id: 18 bit or 29 bit identifier

u32cData[2]: Transfer data field

u8cLen: Length of data field in bytes

u8cFormat: STANDARD or EXTENDED IDENTIFIER

u8cType: FRAME or REMOTE FRAME

u8OverLoad: Disable or Enable

Include

Driver/DrvCAN.h

Return Value

None

DrvCAN_GetVersion

Prototype

iint32_t

DrvCAN_GetVersion (void);

Description

Return the current version number of driver.

Include

Driver/DrvCAN.h

Return Value

Version number :

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

19. DrvPWM Introduction

19.1. PWM Introduction

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

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.

20. DrvPWM APIs Specification

20.1. Constant Definition

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

20.2. Functions

DrvPWM_IsTimerEnabled

Prototype

```
int32_t DrvPWM_IsTimerEnabled(uint8_t u8Timer);
```

Description

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

Parameter

u8Timer [in]

Specify the timer.

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

Include

Driver/DrvPWM.h

Return Value

1: The specified timer is enabled.

0: The specified timer is disabled.

DrvPWM_SetTimerCounter

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer.

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_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_t DrvPWM_GetTimerCounter(uint8_t u8Timer);
```

Description

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

Parameter

u8Timer [in]

Specify the timer.

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

Include

Driver/DrvPWM.h

Return Value

The specified timer counter value.

DrvPWM_EnableInt

Prototype

```
void DrvPWM_EnableInt(uint8_t u8Timer, uint8_t u8Int, PFN_DRV_PWM_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

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

or the capture.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

u8Int [in]

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

DRV_PWM_CAP_RISING_INT: The capture rising interrupt.

DRV_PWM_CAP_FALLING_INT: The capture falling interrupt.

DRV_PWM_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_t u8Timer);
```

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV PWM_TIMER2: PWM timer 2.

DRV PWM_TIMER3: PWM timer 3.

or the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

u8Int [in]

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

DRV PWM_CAP_RISING_INT: The capture rising interrupt.

DRV PWM_CAP_FALLING_INT: The capture falling interrupt.

DRV PWM_CAP_ALL_INT: All capture interrupt.

pfncallback [in]

The call back function for specified timer / capture.

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_ClearInt

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV PWM_TIMER0: PWM timer 0.

DRV PWM_TIMER1: PWM timer 1.

DRV PWM_TIMER2: PWM timer 2.

DRV PWM_TIMER3: PWM timer 3.

or the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_GetIntFlag

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

or the capture.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

Include

Driver/DrvPWM.h

Return Value

1: The specified interrupt occurs.

0: The specified interrupt doesn't occur.

DrvPWM_GetRisingCounter

Prototype

```
uint16_t DrvPWM_GetRisingCounter(uint8_t u8Capture);
```

Description

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

Parameter

u8Capture [in]

Specify the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_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_t DrvPWM_GetFallingCounter(uint8_t u8Capture);
```

Description

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

Parameter

u8Capture [in]

Specify the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_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

```
int32_t DrvPWM_GetCaptureIntStatus(uint8_t u8Capture, uint8_t u8IntType);
```

Description

Check if there's a rising / falling transition

Parameter

u8Capture [in]

Specify the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

u8IntType [in]

Specify the capture.

DRV PWM_CAP_RISING_FLAG: The capture rising interrupt flag.

DRV PWM_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_t u8Capture, uint8_t u8IntType);
```

Description

Clear the rising / falling transition interrupt flag

Parameter

u8Capture [in]

Specify the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

u8IntType [in]

Specify the capture.

DRV_PWM_CAP_RISING_FLAG: The capture rising interrupt flag.

DRV_PWM_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_t u8Timer, uint8_t u8Length, int32_t i32EnableDeadZone);

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

u8Length [in]

Specify Dead Zone Length : 0~255.

i32EnableDeadZone [in]

Enable DeadZone (1) / Disable DeadZone (0)

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_Enable

Prototype

```
void DrvPWM_Enable(uint8_t u8Timer, int32_t i32Enable);
```

Description

This function is used to enable PWM timer / capture function

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

or the capture.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

i32Enable [in]

Enable (1) / Disable (0)

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_SetTimerClk

Prototype

```
uint32_t DrvPWM_SetTimerClk(uint8_t u8Timer, S_DRVPWM_TIME_DATA_T *sPt);
```

Description

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

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 (1) / Inverter Disable (0)

u8ClockSelector: Clock Selector

DRVPWM_CLOCK_DIV_1:

DRVPWM_CLOCK_DIV_2:

DRVPWM_CLOCK_DIV_4:

DRVPWM_CLOCK_DIV_8:

DRV_PWM_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 = DRV_PWM_CAP0/DRV_PWM_CAP1/DRV_PWM_CAP2/DRV_PWM_CAP3)

Include

Driver/DrvPWM.h

Return Value

- 1: The specified interrupt occurs.
- 0: 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_t u8Timer, int32_t i32Enable);
```

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

or the capture.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

i32Enable [in]

Enable (1) / Disable (0)

Include

Driver/DrvPWM.h

Return Value

None

DrvPWM_SelectClockSource

Prototype

```
void DrvPWM_SelectClockSource(uint8_t u8Timer, uint8_t u8ClockSourceSelector);
```

Description

This function is used to select PWM0/PWM1 and PWM2/PWM3 engine clock source.

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0/DRV_PWM_TIMER1: PWM timer 0 or PWM timer 1.

DRV_PWM_TIMER2/DRV_PWM_TIMER3: PWM timer 2 or PWM timer 3.

u8ClockSourceSelector [in]

DRV_PWM_EXT_12M/DRV_PWM_EXT_32K/DRV_PWM_HCLK/DRV_PWM_INTERNAL_22M

DRV_PWM_EXT_12M: external 12 MHz crystal clock

DRV_PWM_EXT_32K: external 32 KHz crystal clock

DRV_PWM_HCLK: HCLK

DRV_PWM_INTERNAL_22M: internal 22 MHz crystal clock

Include

Driver/DrvPWM.h

Return Value

None

21. DrvPS2 Introduction

21.1. PS2 Introduction

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

Because PS2 device controller is very simple, we recommend using macro as much as possible for speed consideration. Because no Rx FIFO, so DrvPS2_Read only read one byte; but DrvPS2_Write can write any length bytes to host

Default PS2 interrupt handler has been implemented, it's PS2_IRQHandler. User can install interrupt call back function using function DrvPS2_EnableInt and uninstall using DrvPS2_DisableInt

21.2. PS2 Feature

The PS2 device controller includes following features:

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

22. DrvSP2 APIs Specification

22.1. Macro

DRVPS2_OVERRIDE

Prototype

```
void DRVPS2_OVERRIDE(bool state);
```

Description

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

Parameter

state [in]

Specify software override or not. 1 means to enable software override PS2 CLK/DATA pin state, 0 means to disable it.

Include

Driver/DrvPS2.h

Return Value

None.

DRVPS2_PS2CLK

Prototype

```
void DRVPS2_PS2CLK(bool state);
```

Description

This macro can force PS2CLK high or low regardless of the internal state of the device controller if DRVPS2_OVERRIDE called. 1 means high, 0 means low

Parameter

state [in]

Specify PS2CLK line high or low

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_PS2DATA

Prototype

void DRVPS2_PS2DATA(bool state);

Description

This macro can force PS2DATA high or low regardless of the internal state of the device controller if DRVPS2_OVERRIDE called. 1 means high, 0 means low.

Parameter

u16Port [in]

Specify PS2DATA line high or low

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_CLR_FIFO

Prototype

void DRVPS2_CLR_FIFO();

Description

The macro is used to clear tx fifo.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_ACKNOTALWAYS

Prototype

```
void DRVPS2_ACKNOTALWAYS(bool state);
```

Description

The macro is used to enable ack always or not. If parity error or stop bit is not received correctly, acknowledge bit will not be sent to host at 12th clock, If state=1; else always send acknowledge to host at 12th clock for host to device communication

Parameter

state [in]

Specify enable or disable

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_RXINTENABLE

Prototype

```
void DRVPS2_RXINTENABLE();
```

Description

The macro is used to enable Rx interrupt. When acknowledge bit is sent for Host to device communication, Rx interrupt will happen

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_RXINTDISABLE

Prototype

```
void DRVPS2_RXINTDISABLE();
```

Description

The macro is used to disable Rx interrupt.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXINTENABLE

Prototype

void DRVPS2_TXINTENABLE();

Description

The macro is used to enable Tx interrupt. When STOP bit is transmitted, Tx interrupt will happen.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXINTDISABLE

Prototype

void DRVPS2_TXINTDISABLE ();

Description

The macro is used to disable Tx interrupt.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_PS2ENABLE

Prototype

```
void RVPS2_PS2ENABLE ();
```

Description

The macro is used to enable PS2 device controller.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_PS2DISABLE

Prototype

```
void RVPS2_PS2DISABLE ();
```

Description

The macro is used to disable PS2 device controller.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXFIFO

Prototype

```
void DRVPS2_TXFIFO();
```

Description

The macro is used to set Tx fifo depth. The range is [0,15]

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_SWOVERRIDE
Prototype

```
void DRVPS2_SWOVERRIDE(bool data, bool clk);
```

Description

The macro is used to set PS2DATA and PS2CLK line by software override. It's equal to these macros:

```
DRVPS2_PS2DATA(data);
```

```
DRVPS2_PS2CLK(clk);
```

```
DRVPS2_OVERRIDE(1);
```

Parameter
data [in]

Specify PS2DATA line high or low

clk [in]

Specify PS2CLK line high or low

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_INTCLR
Prototype

```
void DRVPS2_INTCLR(uint8_t intclr);
```

Description

The macro is used to clear interrupt status.

Parameter
intclr [in]

Specify to clear Tx or Rx interrupt. Intclr=0x1 for clear Rx interrupt; Intclr=0x2 for clear Tx interrupt; Intclr=0x3 for clear Rx and Tx interrupt

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_RXDATA
Prototype

```
uint8_t DRVPS2_RXDATA();
```

Description

Reads 1 byte from the receive register.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

One byte data received.

DRVPS2_TXDATAWAIT
Prototype

```
void DRVPS2_TXDATAWAIT(uint32_t data, uint32_t len);
```

Description

The macro is used to wait TX FIFO EMPTY, set Tx fifo depth and fill Tx fifo 0-3. Data is sent if bus is in IDLE state immediately. The range of len is [0, 15]

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1

Parameter
data [in]

Specify the data sent

len [in]

Specify the length of the data sent. Unit is byte. Range is [0, 15]

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXDATA

Prototype

```
void DRVPS2_TXDATA(uint32_t data, uint32_t len);
```

Description

The macro is used to set Tx fifo depth and fill Tx fifo 0-3. But not wait TX FIFO EMPTY. Data is sent if bus is in IDLE state immediately. The range of len is [0, 15]

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data sent

len [in]

Specify the length of the data sent. Unit is byte. Range is [0, 15]

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXDATA0

Prototype

```
void DRVPS2_TXDATA0(uint32_t data);
```

Description

The macro is used to fill Tx fifo 0-3. But not wait TX FIFO EMPTY and not set Tx fifo depth. Data is sent if bus is in IDLE state immediately.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXDATA1

Prototype

```
void DRVPS2_TXDATA1(uint32_t data);
```

Description

The macro is used to fill Tx fifo 4-7. But not wait TX FIFO EMPTY and not set Tx fifo depth.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXDATA2

Prototype

```
void DRVPS2_TXDATA2(uint32_t data);
```

Description

The macro is used to fill Tx fifo 8-11. But not wait TX FIFO EMPTY and not set Tx fifo depth.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_TXDATA3

Prototype

void DRVPS2_TXDATA3(uint32_t data);

Description

The macro is used to fill Tx fifo 12-15. But not wait TX FIFO EMPTY and not set Tx fifo depth.

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

data [in]

Specify the data that will be sent.

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_ISTXEMPTY

Prototype

void DRVPS2_ISTXEMPTY();

Description

The macro is used to check Tx fifo whether or not empty

When transmitted data byte number is equal to FIFODEPTH then TXEMPTY bit is set to 1.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_ISFRAMEERR

Prototype


```
void DRVPS2_ISFRAMEERR();
```

Description

The macro is used to check whether or not frame error happen. For host to device communication, if STOP bit is not received it is a frame error. If frame error occurs, DATA line may keep at low state after 12th clock. At this moment, software override PS2CLK to send clock till PS2DATA release to high state. After that, device sends a “Resend” command to host

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

DRVPS2_ISRXBUSY

Prototype

```
void DRVPS2_ISRXBUSY();
```

Description

The macro is used to check whether or not Rx busy. If busy it indicates that PS2 device is currently receiving data

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None.

22.2. Functions

DrvPS2_Open

Prototype

```
int32_t DrvPS2_Open();
```

Description

This function is used to init PS2 IP. it includes enable PS2 clock, enable PS2 controller, clear FIFO, set Tx Fifo depth to default value zero

Parameter

None

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS.

DrvPS2_Close

Prototype

void DrvPS2_Close();

Description

This function is used to disable PS2 controller and disable PS2 clock.

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None

DrvPS2_EnableInt

Prototype

```
int32_t DrvPS2_EnableInt (
    uint32_t  u32InterruptFlag,
    PFN_DRVPS2_CALLBACK pfncallback
);
```

Description

This function is used to enable Tx/Rx interrupt and install interrupt call back function.

Parameter

u32InterruptFlag [in]

specify Tx/Rx interrupt flag that will be enable. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT

pfncallback [in]

specify the interrupt call back function. When PS2 interrupt happen, this function will be call

Include

Driver/ DrvPS2.h

Return Value

E_SUCCESS

DrvPS2_DisableInt

Prototype

void DrvPS2_DisableInt(uint32_t u32InterruptFlag);

Description

This function is used to disable Tx/Rx interrupt and uninstall interrupt call back function..

Parameter

u32InterruptFlag [in]

specify Tx/Rx interrupt flag that will be disabled. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT.

Include

Driver/ DrvPS2.h

Return Value

None

DrvPS2_IsIntEnabled

Prototype

uint32_t DrvPS2_IsIntEnabled(uint32_t u32InterruptFlag);

Description

This function is used to check whether or not interrupt be enabled.

Parameter

u32InterruptFlag [in]

specify Tx/Rx interrupt flag that will be checked. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT.

Include

Driver/ DrvPS2.h

Return Value

None

DrvPS2_ClearIn

Prototype

```
uint32_t DrvPS2_ClearInt(uint32_t u32InterruptFlag);
```

Description

This function is used to clear interrupt status.

Parameter

U32InterruptFlag [in]

specify Tx/Rx interrupt flag that will be cleared. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS Success.

DrvPS2_GetIntStatus

Prototype

```
int8_t DrvPS2_GetIntStatus(uint32_t u32InterruptFlag);
```

Description

This function is used to check interrupt status. If interrupt that be checked happens it will return TRUE

Parameter

U32InterruptFlag [in]

specify Tx/Rx interrupt flag that will be checked. It can be DRVPS2_TXINT or DRVPS2_RXINT

Include

Driver/ DrvPS2.h

Return Value

TRUE: interrupt that be checked happens

FALSE: interrupt that be checked doesn't happen

DrvPS2_SetTxFIFODepth

Prototype

```
void DrvPS2_SetTxFIFODepth(uint16_t u16TxFIFODepth);
```

Description

This function is used to set Tx fifo depth. The function will call macro DRVPS2_TXFIFO to set Tx fifo depth

Parameter

u16TxFIFODepth [in]

specify Tx fifo depth. The range can be [0, 15]

Include

Driver/ DrvPS2.h

Return Value

None

DrvPS2_Read
Prototype

```
int32_t DrvPS2_Read(uint8_t *pu8RxBuf);
```

Description

The function is used to read one byte to the buffer of pu8RxBuf. The function will call macro DRVPS2_RXDATA to receive data

Parameter

pu8RxBuf [out]

the buffer is used to contain byte received. The size of buffer needs one byte only

Include

Driver/ DrvPS2.h

Return Value

E_SUCCESS Success.

DrvPS2_Write
Prototype

```
int32_t
DrvPS2_Write(
    uint32_t *pu32TxBuf,
    uint32_t u32WriteBytes
);
```

Description

The function is used to write the buffer of pu32TxBuf and the length of u32WriteBytes to host. if data count sent is less than 16 bytes, please use macro DRVPS2_TXDATAxxx for speed

Parameter

pu32TxBuf [in]

the data that will be sent to host.

u32WriteBytes [in]

the length of data that will be sent to host.

Include

Driver/ DrvPS2.h

Return Value

E_SUCCESS Success.

DrvPS2_GetVersion

Prototype

int32_t DrvPS2_GetVersion(void);

Description

Return the current version number of driver.

Include

Driver/ DrvPS2.h

Return Value

Version number :

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

23. DrvFMC Introduction

23.1. Introduction

NUC100 series equips with 128/64/32k bytes on chip embedded flash for application program memory (APROM) that can be updated. NUC100 series also provide additional 4k bytes data flash for user to store some application depended data before chip power off. For 128k bytes device, the data flash is shared with 128k program memory and its shared address is defined by user in Config1. The data flash size is defined by user depends on user application request.

23.2. Feature

The FMC includes following features:

- 128/64/32kB application program memory (APROM).
- 4kB in system programming loader program memory (LDROM).
- 4k data flash with 512 bytes page erase unit.
- Programmable data flash start address and memory size for 128 program memory.

24. DrvFMC APIs Specification

24.1. Functions

DrvFMC_EnableISP

Prototype

```
void DrvFMC_EnableISP(int32_t i32Enable);
```

Description

To enable ISP function.

Parameter

i32Enable [in]

1:enable, 0:disable

Include

Driver/DrvFMC.h

Return Value

None

DrvFMC_BootSelect

Prototype

```
void DrvFMC_BootSelect(E_FMC_BOOTSELECT boot);
```

Description

To select next booting from APROM or LDROM.

Parameter

boot [in]

Specify APROM or LDROM.

Include

Driver/DrvFMC.h

Return Value

None

DrvFMC_GetBootSelect

Prototype

E_FMC_BOOTSELECT DrvFMC_GetBootSelect(void);

Description

To get current boot select setting.

Parameter

None.

Include

Driver/DrvFMC.h

Return Value

APROM The current boot select setting is in APROM

LDROM The current boot select setting is in LDROM

DrvFMC_EnableLDUpdate

Prototype

void DrvFMC_EnableLDUpdate(int32_t i32Enable);

Description

To enable LDROM update function.

Parameter

i32Enable [in]

1:enable, 0:disable

Include

Driver/DrvFMC.h

Return Value

None

DrvFMC_EnablePowerSaving

Prototype

void DrvFMC_EnablePowerSaving(int32_t i32Enable);

Description

To enable flash access power saving function.

Parameter

i32Enable [in]

1:enable, 0:disable

Include

Driver/DrvFMC.h

Return Value

None

DrvFMC_ReadCID

Prototype

```
int32_t DrvFMC_ReadCID(uint32_t * u32data);
```

Description

To read company ID.

Parameter

u32data [in]

The data to store company ID.

Include

Driver/DrvFMC.h

Return Value

0 Succeed

<0 Failed

DrvFMC_ReadDID

Prototype

```
int32_t DrvFMC_ReadDID(uint32_t * u32data);
```

Description

To read device ID.

Parameter

u32data [in]

The data to store device ID.

Include

Driver/DrvFMC.h

Return Value

0 Succeed
<0 Failed

DrvFMC_Write

Prototype

int32_t DrvFMC_Write(uint32_t u32addr, uint32_t u32data);

Description

To write word data into APROM, LDROM, Data Flash or Config.

Parameter

u32addr [in]

Word address of flash.

u32data [in]

Word data to program into flash.

Include

Driver/DrvFMC.h

Return Value

0 Succeed
<0 Failed

DrvFMC_Read

Prototype

int32_t DrvFMC_Read(uint32_t u32addr, uint32_t * u32data);

Description

To read data from APROM, LDROM, Data Flash or Config.

Parameter

u32addr [in]

Word address of flash.

u32data [in]

The data to store data from flash.

Include

Driver/DrvFMC.h

Return Value

0 Succeed

<0 Failed

DrvFMC_Erase

Prototype

int32_t DrvFMC_Erase(uint32_t u32addr);

Description

To page erase flash or Config. The flash page erase unit is 512 bytes.

Parameter

u32addr [in]

Flash page base address or Config0 address.

Include

Driver/DrvFMC.h

Return Value

0 Succeed

<0 Failed

DrvFMC_WriteConfig

Prototype

int32_t DrvFMC_WriteConfig(uint32_t u32data0, uint32_t u32data1);

Description

To erase Config and write data into Config0 and Config1.

Parameter

u32data0 [in]

Word data for Config0.

u32data1 [in]

Word data for Config1.

Include

Driver/DrvFMC.h

Return Value

0	Succeed
<0	Failed

DrvFMC_ReadDataFlashBaseAddr

Prototype

```
uint32_t DrvFMC_ReadDataFlashBaseAddr(void);
```

Description

To read data flash base address.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

Data Flash base address

25. DrvUSB Introduction

25.1. 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.

25.2. Feature

- Conform to USB2.0 Full speed, 12Mbps.
- Provide 1 interrupt source with 4 interrupt events.
- Support Control, Bulk, Interrupt, and Isochronous transfers.
- Suspend when no bus signaling for 3 ms.
- Provide 6 endpoints for configuration.
- Include 512 bytes internal SRAM as USB buffer.
- Provide remote wake-up capability.

25.3. Call Flow

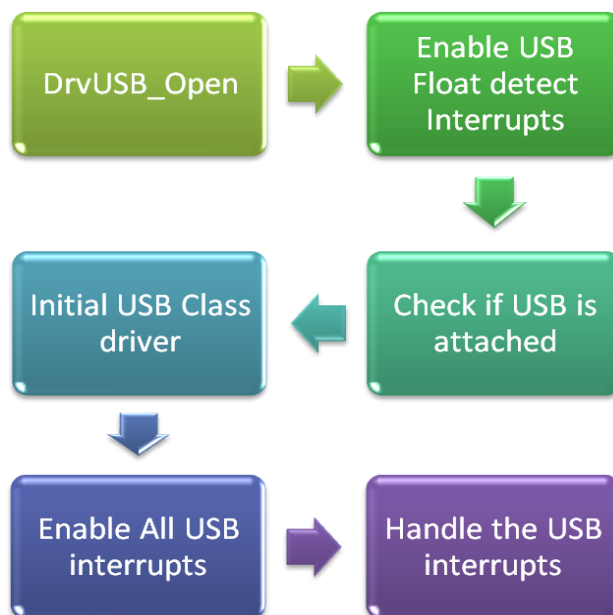


Figure 25-1: USB Driver Call Flow

26. DrvUSB APIs Specification

26.1. Macro Functions

_DRVUSB_ENABLE_MISC_INT

Prototype

```
static __inline
void _DRVUSB_ENABLE_MISC_INT (
    uint32_t    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_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_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_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_EP_READY_AND_TRIG_STALL

Prototype

static __inline

void _DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL (

```
uint32_t    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_t    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_t    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_t    u32EPNum,
    uint32_t    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_t u32EPNum,
    uint32_t 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_t
_DRVUSB_GET_EP_DATA_SIZE (
    uint32_t    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_t    u32EPNum,
    int32_t     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_t    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_t
__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_t    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_t    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_t    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_t    u32CFGNum,
    uint32_t    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_t
_DRVUSB_GET_CFG (
    uint32_t    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_t    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_t
_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_t    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_t
_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_t    u8CFGPNum,
    uint32_t    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_t
_DRVUSB_GET_CFGP (
    uint32_t    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_t    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_t    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_t    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_t    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_t    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_t    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

26.2. Functions

DrvUSB_GetVersion

Prototype

```
uint32_t
DrvUsb_GetVersion (void);
```

Description

Get this module's version.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Version number :

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

DrvUSB_Open

Prototype

```
int32_t
DrvUsb_Open (
    void *    pVoid
)
```

Description

This function is used to reset USB controller, initial the USB endpoints, interrupts, and USB driver structures. It also used to call the relative handler when the USB is attached before USB driver opened. The user must provide the materials before they can call DrvUSB_Open, including sEpDescription, g_sBusOps.

sEpDescription:

The structure type of sEpDescription is as follows:

```
typedef struct
{
    //bit7 is directory bit, 1: input; 0: output
    uint32_t  u32EPAAddr;
    uint32_t  u32MaxPacketSize;
    uint8_t * u8SramBuffer;
}S_DRVUSB_EP_CTRL;
```

This structure is used to set the endpoint number, maximum packet size, and buffer of specified endpoint hardware. There are 6 endpoints hardware available in NUC100 series USB controller.

g_sBusOps:

The structure type of g_sBusOps is as follows:

```
typedef struct
{
    PFN_DRVUSB_CALLBACK    apfnCallback;
    void *                  apCallbackArgu;
}S_DRVUSB_EVENT_PROCESS
```

It is used to install the USB bus event handler, such as follows:

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
```

```

        {NULL, NULL}, /* attach event callback */
        {NULL, NULL}, /* detach event callback */
        {DrvUSB_BusResetCallback, &g_HID_sDevice}, /* bus reset event callback */
        {NULL, NULL}, /* bus suspend event callback */
        {NULL, NULL}, /* bus resume event callback */
        {DrvUSB_CtrlSetupAck, &g_HID_sDevice}, /* setup event callback */
    };

```

Parameter

pVoid

NULL	None
Callback function	If the pVoid is not NULL, it will be the callback function of USB interrupt and it is called after DrvUSB_PreDispatchEvent in USB interrupt handler.

Include

Driver/DrvUsb.h

Return Value

E_SUCCESS: Succeed

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

int32_t DrvUSB_IsData0(uint32_t u32EpId)

Description

To check if the current DATA is DATA0. If it is false, then it should be DATA1.

Parameter

u32EpId The hardware endpoint id. The id could be 0~5.

Include

Driver/DrvUSB.h

Return Value

TRUE The current data packet is DATA0

FALSE The current data packet is DATA1

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_t DrvUSB_GetEpIdentity(uint32_t u32EpNum, uint32_t u32EpAttr)
```

Description

To get endpoint index base on endpoint number and direction. The endpoint id is used to identify the hardware endpoint resource. The range of endpoint index could be 0 ~ 5. The endpoint number is assigned by software and it could be 0 ~ 15 according to USB standard. Host will access the device through relative endpoint number.

Parameter

u32EpNum The endpoint number

u32EpAttr The endpoint number attribute. It could be EP_INPUT or EP_OUTPUT.

Include

Driver/DrvUSB.h

Return Value

0~5 The endpoint id of specified endpoint address.

otherwise Can't get relative endpoint id according to the input endpoint address.

DrvUSB_GetEpId
Prototype

```
uint32_t DrvUSB_GetEpId(uint32_t u32EpNum)
```

Description

Get endpoint index base on endpoint address. This argument “u32EpNum” is different from DrvUSB_GetEPNumber's because its argument includes direction bit (bit 7). eg: 0x81. If the bit 7 is high, it indicates this is EP_INPUT, otherwise it is EP_OUTPUT.

Parameter

u32EpNum The endpoint address with direction information at bit 7.

Include

Driver/DrvUSB.h

Return Value

0~5 The endpoint index of specified endpoint address.
otherwise Can't get relative endpoint id according to the input endpoint address.

DrvUSB_DataOutTrigger

Prototype

int32_t DrvUSB_DataOutTrigger(uint32_t u32EpNum, uint32_t u32Size)

Description

Trigger data out ready flag by write MXPLD register. It indicates the relative endpoint buffer is ready to receive data out packet.

Parameter

u32EpNum The endpoint number.
u32Size Maximum size want to receive from USB

Include

Driver/DrvUSB.h

Return Value

0 Succeed
<0 Can't get relative endpoint id according to the input endpoint address.

DrvUSB_GetOutData

Prototype

uint8_t * DrvUSB_GetOutData(uint32_t u32EpNum, uint32_t *u32Size)

Description

This function will return the buffer pointer of u32EpNum 's out USB SRAM buffer. User can use this pointer to get the data payload of current data out packet.

Parameter

u32EpNum The endpoint number.
u32Size Data size received from USB

Include

Driver/DrvUSB.h

Return Value

To return USB SRAM address.

DrvUSB_DataIn

Prototype

```
int32_t DrvUSB_DataIn(uint32_t u32EpNum, const uint8_t * u8Buffer, uint32_t u32Size)
```

Description

Trigger ready flag for sending data after receive IN token from host, USB will send the data. if u8Buffer == NULL && u32Size == 0 then send DATA1 always else DATA0 and DATA1 by turns.

Parameter

u32EpNum	The endpoint number.
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(void * pVoid)
```

Description

Bus reset handler. After receiving bus reset event, this handler will be called. It will reset USB address, accept SETUP packet and initial the endpoints.

Parameter

pVoid	Parameter passed by g_sBusOps[].
-------	----------------------------------

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_InstallClassDevice

Prototype

```
void * DrvUSB_InstallClassDevice(S_DRVUSB_CLASS *sUsbClass)
```

Description

Register USB class device to USB driver.

Parameter

sUsbClass USB class structure pointer.

Include

Driver/DrvUSB.h

Return Value

Return USB driver pointer

DrvUSB_InstallCtrlHandler

Prototype

```
int32_t DrvUSB_InstallCtrlHandler(
    void *                *device,
    S_DRVUSB_CTRL_CALLBACK_ENTRY *psCtrlCallbackEntry,
    uint32_t              u32RegCnt
)
```

Description

Register ctrl pipe handler including SETUP ACK , IN ACK, OUT ACK handle for Standard/Vendor/Class command.

Parameter

device USB driver device pointer.

psCtrlCallbackEntryHandler 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(void * pArgu)
```

Description

When SETUP ack interrupt happen, this function will be called. It will call SETUP handler that DrvUSB_InstallCtrlHandler registered base on command category and command.

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlDataInAck
Prototype

```
void DrvUSB_CtrlDataInAck(void * pArgu)
```

Description

When IN ack interrupt happen, this function will be called. It will call IN ACK handler that DrvUSB_InstallCtrlHandler registered base on command category and command.

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlDataOutAck
Prototype

```
void DrvUSB_CtrlDataOutAck(void * pArgu)
```

Description

When OUT ack interrupt happen, this function will be called. It will call OUT handler that DrvUSB_RegisterCtrl registered base on command category and command.

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlDataInDefault

Prototype

void DrvUSB_CtrlDataInDefault(void * pVoid)

Description

IN ACK default handler. It is used to return ACK for next OUT token.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler.

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlDataOutDefault

Prototype

void DrvUSB_CtrlDataOutDefault(void * pVoid)

Description

OUT ACK default handler. It is used to return zero data length packet when next IN token.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler.

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_Reset

Prototype

void DrvUSB_Reset(uint32_t u32EpNum)

Description

Restore the specified CFGx and CFGPx registers according the endpoint number.

Parameter

u32EpNum The endpoint number to reset

Include

Driver/DrvUSB.h

Return Value

None

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);

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_t 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_t DrvUSB_GetFreeSram(void)
```

Description

Get free USB SRAM buffer address after EP assign base on sEpDescription[i].u32MaxPacketSize in DrvUSB_Open. User can get this for dual buffer.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Free USB SRAM address

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

int32_t DrvUSB_IsSelfPowerEnabled(int32_t * 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

```
int32_t DrvUSB_IsRemoteWakeupEnabled (int32_t * 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_t DrvUSB_SetMaxPower(uint32_t u32MaxPower)

Description

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

Parameter

u32MaxPower	Maximum power value
-------------	---------------------

Include

Driver/DrvUSB.h

Return Value

E_SUCCESS	Successful
<0	Wrong maximum value

DrvUSB_GetMaxPower

Prototype

int32_t DrvUSB_GetMaxPower(void)

Description

Get current max power. The unit is in 2mA,i.e 0x32 is 100mA.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Return the maximum power. (2mA unit)

DrvUSB_EnableUsb

Prototype

```
void DrvUSB_EnableUsb(S_DRVUSB_DEVICE *psDevice)
```

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(void * 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_CtrlSetupClearSetFeature(void * pVoid)
```

Description

Setup ACK handler for Clear feature command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupGetConfiguration

Prototype

void DrvUSB_CtrlSetupGetConfiguration(void * pVoid)

Description

Setup ACK handler for Get configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupGetStatus

Prototype

void DrvUSB_CtrlSetupGetStatus(void * pVoid)

Description

Setup ACK handler for Get status command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupGetInterface

Prototype

void DrvUSB_CtrlSetupGetInterface(void * pVoid)

Description

Setup ACK handler for Get interface command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlSetupSetConfiguration

Prototype

void DrvUSB_CtrlSetupSetConfiguration(void * pVoid)

Description

Setup ACK handler for Set configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

DrvUSB_CtrlDataInSetAddress

Prototype

void DrvUSB_CtrlDataInSetAddress(void * pVoid)

Description

Setup ACK handler for Set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

27. DrvPDMA Introduction

27.1. PDMA Introduction

The NUC100 contains a peripheral direct memory access (PDMA) controller that transfers data to and from memory or transfer data to and from APB. The PDMA has nine channels of DMA (Peripheral-to-Memory or Memory-to-Peripheral or Memory-to-Memory). For each PDMA channel (PDMA CH0~CH8), there is one word buffer to do transfer buffer between the Peripherals APB IP and Memory.

Software can stop the PDMA operation by disable PDMA [PDMACEN]. The CPU can recognize the completion of a PDMA operation by software polling or when it receives an internal PDMA interrupt. The NUC101 PDMA controller can increment source or destination address and fixed them as well.

27.2. PDMA Feature

The PDMA includes following features:

- AMBA AHB master/slave interface compatible, for data transfer and register read/write.
- PDMA support 32-bit source and destination addressing range address increment and fixed.

28. DrvPDMA APIs Specification

28.1. Functions

DrvPDMA_Init

Prototype

```
int32_t  
DrvPDMA_Init (void);
```

Description

The function is used to initialize PDMA

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

DrvPDMA_Close

Prototype

```
void DrvPDMA_Close (void);
```

Description

The function is disable all PDMA channel clock and AHB PDMA clock

Parameter

None

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

DrvPDMA_CHEnableTransfer

Prototype

```
int32_t
DrvPDMA_CHEnableTransfer(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

Description

The function is used to enable PDMA data read or write transfer

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

DrvPDMA_CHSoftwareReset

Prototype

```
int32_t
DrvPDMA_CHSoftwareReset(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

Description

The function is used to software reset Channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

DrvPDMA_Open

Prototype

```
int32_t
DrvPDMA_Open(
    E_DRVPDMA_CHANNEL_INDEX sChannel,
    STR_PDMA_T *sParam
);
```

Description

The function is configure PDMA setting

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

sParam [in]

the struct parameter to configure PDMA.It includes

sSrcAddr: Source Address

sDestAddr: Destination Address

u8TransWidth: Transfer Width

u8Mode: Operation Mode

i32ByteCnt: Byte Count

Include

Driver/DrvPDMA.h

Return Value

E_DRVPDMA_ERR_PORT_INVALID.	Wrong Port parameter
E_SUCCESS.	Success

DrvPDMA_ClearInt

Prototype

```
void
DrvPDMA_ClearInt(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_FLAG eIntFlag
);
```

Description

The function is used to clear interrupt status for channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntFlag [in]

Interrupt source : eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

None

DrvPDMA_PollInt

Prototype

int32_t

DrvPDMA_PollInt(

E_DRVPDMA_CHANNEL_INDEX eChannel,

E_DRVPDMA_INT_FLAG eIntFlag

);

Description

The function is used to polling channel interrupt status

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntFlag [in]

Interrupt source : eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

True: Interrupt status is set.

False: Interrupt status is clear.

DrvPDMA_SetAPBTransferWidth

Prototype

```
int32_t
DrvPDMA_SetAPBTransferWidth(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_TRANSFER_WIDTH eTransferWidth
);
```

Description

The function is used to set APB transfer width for channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eTransferWidth [in]

eDRVPDMA_WIDTH_32BITS

eDRVPDMA_WIDTH_8BITS

eDRVPDMA_WIDTH_16BITS

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS Success

DrvPDMA_SetCHForAPBDevice

Prototype

```
int32_t
DrvPDMA_SetCHForAPBDevice(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_APB_DEVICE        eDevice,
    E_DRVPDMA_APB_RW            eRWAPB
);
```

Description

The function is used to select PDMA channel for APB device

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eDevice [in]

CH for APB device. It includes of

eDRVPDMA_SPI0~3,eDRVPDMA_UART0~1,
eDRVPDMA_USB,eDRVPDMA_ADC

eRWAPB [in]

eDRVPDMA_WRITE_APB / eDRVPDMA_READ_APB

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS	Success
E_DRVPDMA_FALSE_INPUT	Wrong parameter

DrvPDMA_DisableInt

Prototype

```
void
DrvPDMA_DisableInt(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_ENABLE eIntSource
);
```

Description

The function is used to disable Interrupt for channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]

Interrupt source : eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

None

DrvPDMA_EnableInt

Prototype

```
int32_t
DrvPDMA_EnableInt(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_ENABLE eIntSource
);
```

Description

The function is used to enable Interrupt for channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]

Interrupt source : eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

DrvPDMA_GetAPBTransferWidth

Prototype

```
int32_t
DrvPDMA_GetAPBTransferWidth(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_TRANSFER_WIDTH* peTransferWidth
);
```

Description

The function is used to get transfer width from channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

peTransferWidth [in]

A data pointer to save APB transfer width

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

DrvPDMA_GetCHForAPBDevice

Prototype

```
E_DRVPDMA_CHANNEL_INDEX
DrvPDMA_GetCHForAPBDevice(
    E_DRVPDMA_APB_DEVICE eDevice,
    E_DRVPDMA_APB_RW eRWAPB
);
```

Description

The function is used to get PDMA channel for APB device

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eDevice [in]

CH for APB device. It includes of
eDRVPDMA_SPI0~3,eDRVPDMA_UART0~1,
eDRVPDMA_USB,eDRVPDMA_ADC

eRWAPB [in]

CH for APB device. It includes of

Include

Driver/DrvPDMA.h

Return Value

Channel Number

E_DRVPDMA_FALSE_INPUT Wrong parameter

DrvPDMA_GetCurrentDestAddr

Prototype

```
uint32_t
DrvPDMA_GetCurrentDestAddr(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

Description

The function is used to get current destination address from channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

current destination address

DrvPDMA_GetCurrentSourceAddr
Prototype

uint32_t

```
DrvPDMA_GetCurrentSourceAddr(
    E_DRVPDMA_CHANNEL_INDEX eChannel
)
```

Description

The function is used to get current source address from channelx.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Current source address register

DrvPDMA_GetCurrentTransferCount
Prototype

uint32_t

```
DrvPDMA_GetCurrentTransferCount(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

Description

The function is used to get current transfer count from channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Current transfer count register

DrvPDMA_GetInternalBufPointer

Prototype

uint32_t

DrvPDMA_GetInternalBufPointer(

E_DRVPDMA_CHANNEL_INDEX eChannel

);

Description

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

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVUART_TIMEOUT: FIFO polling timeout

DrvPDMA_GetSharedBufData

Prototype

uint32_t

DrvPDMA_GetSharedBufData(

E_DRVPDMA_CHANNEL_INDEX eChannel,

)

Description

The function is used to get shared buffer content from channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVUART_TIMEOUT: FIFO polling timeout

DrvPDMA_GetTransferLength

Prototype

int32_t

```
DrvPDMA_GetTransferLength(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    uint32_t* pu32TransferLength
);
```

Description

The function is used to get channel transfer length setting

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

pu32TransferLength [in]

The data pointer to save transfer length

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

DrvPDMA_InstallCallBack

Prototype

```
int32_t
DrvPDMA_InstallCallBack(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_ENABLE eIntSource,
    PFN_DRVPDMA_CALLBACK pfncallback
);
```

Description

The function is used to install call back function for Channelx & Interrupt source

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]

Interrupt source eDRVPDMA_TABORT/eDRVPDMA_BLKD

pfncallback [in]

The callback function pointer

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

DrvPDMA_IsCHBusy

Prototype

```
int32_t
DrvPDMA_IsCHBusy(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

Description

The function is used to Get Channel Enable/Disable status

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

TRUE: The channel is busy

FALSE: The channel is un-used.

DrvPDMA_IsIntEnabled

Prototype

int32_t

```
DrvPDMA_IsIntEnabled(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_ENABLE eIntSource
);
```

Description

The function is used to check if the specified interrupt source is enabled in Channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]

Interrupt source: eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

TRUE: The interrupt source is enable.

FALSE: The interrupt source is disable.

DrvPDMA_IsIntEnabled

Prototype

```
int32_t
DrvPDMA_IsIntEnabled(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_ENABLE eIntSource
);
```

Description

The function is used to check if the specified interrupt source is enabled in Channelx

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]

Interrupt source: eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

TRUE: The interrupt source is enable.

FALSE: The interrupt source is disable.

DrvPDMA_GetVersion

Prototype

```
int32_T
DrvPDMA_GetVersion (void);
```

Description

Return the current version number of driver.

Include

Driver/DrvPDMA.h

Return Value

Version number:

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

29. Revision History

Version	Date	Description
V1.00.001	Jan. 8, 2009	<ul style="list-style-type: none"> Created

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.