

NUC1xx 驱动参考 指南 V1.00.001

发布日期: 1. 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 介绍.....	14
1.1. 介绍14	
2. DrvSYS APIs 说明	15
2.1. 静态定义.....	15
2.1.1. IP 初始化.....	15
2.1.2. IP 时钟使能控制.....	15
2.2. 函数16	
DrvSYS_ReadProductID	16
DrvSYS_GetRstSrc	17
DrvSYS_ClearRstSrc	17
DrvSYS_ResetIP	18
DrvSYS_ResetCPU	18
DrvSYS_ResetChip	18
DrvSYS_EnableBOD	19
DrvSYS_SelectBODVolt.....	19
DrvSYS_EnableBODRst.....	20
DrvSYS_EnableBODLowPowerMode.....	20
DrvSYS_EnableLowVoltRst	21
DrvSYS_GetBODState.....	21
DrvSYS_EnableTempatureSensor.....	21
DrvSYS_SetPORDisCode	22
DrvSYS_UnlockKeyAddr	22
DrvSYS_LockKeyAddr.....	23
DrvSYS_SetRCAdjValue	23
DrvSYS_SetIPClock.....	24
DrvSYS_SetHCLKSource.....	24
DrvSYS_SetSysTickSource.....	25
DrvSYS_SetIPClockSource.....	25
DrvSYS_SetClockDivider	26
DrvSYS_SetOscCtrl	27
DrvSYS_EnablePWRWUInt	27
DrvSYS_EnablePowerDown.....	28
DrvSYS_SetPowerDownWaitCPU	28
DrvSYS_SetPllSrc	29
DrvSYS_SetPLLPowerDown.....	29
DrvSYS_GetEXTClock.....	30
DrvSYS_GetPllContent.....	30
DrvSYS_GetPLLClock	31
DrvSYS_GetHCLK	31

DrvSYS_Open	31
3. DrvUART 介绍.....	33
3.1. 串口介绍.....	33
3.2. 串口特性.....	33
4. DrvUART APIs 说明.....	34
4.1. 静态定义.....	34
4.2. 函数35	
DrvUART_Open.....	35
DrvUART_Close	36
DrvUART_EnableInt.....	36
DrvUART_IsIntEnabled	37
DrvUART_DisableInt.....	38
DrvUART_ClearInt	39
DrvUART_GetIntStatus	40
DrvUART_SetFIFOTriggerLevel.....	41
DrvUART_GetCTS	41
DrvUART_SetRTS.....	42
DrvUART_SetRxTimeOut	43
DrvUART_Read	43
DrvUART_Write	44
DrvUART_SetPDMA.....	45
DrvUART_OpenIRCR	45
DrvUART_OpenLIN	46
DrvUART_GetVersion.....	47
5. DrvTIMER 介绍	48
5.1. 定时器介绍.....	48
5.2. 定时器特性.....	48
6. DrvTIMER APIs 说明.....	49
6.1. 函数49	
DrvTIMER_GetStatus	49
DrvTIMER_SetTimerEvent.....	49
DrvTIMER_ClearTimerEvent	50
DrvTIMER_ResetTicks	51
DrvTIMER_Init	51
DrvTIMER_Open	51
DrvTIMER_GetTicks	52
DrvTIMER_Delay	53
DrvTIMER_Ioctl	53
DrvTIMER_Close.....	54

DrvWDT_Open	54
DrvWDT_ResetCount	55
DrvWDT_Ioclt.....	55
DrvWDT_Close.....	56
DrvTIMER_GetVersion	56
7. DrvGPIO 介绍.....	58
7.1. GPIO 介绍	58
8. DrvGPIO APIs 说明	59
8.1. 函数59	
DrvGPIO_Open	59
DrvGPIO_Close.....	60
DrvGPIO_SetBit.....	60
DrvGPIO_ClrBit.....	61
DrvGPIO_GetBit	61
DrvGPIO_SetPortBits.....	62
DrvGPIO_GetPortBits	62
DrvGPIO_GetPortDoutBits	62
DrvGPIO_EnableInt	63
DrvGPIO_DisableInt	64
DrvGPIO_SetDebounceTime	64
DrvGPIO_EnableDebounce.....	65
DrvGPIO_DisableDebounce	65
DrvGPIO_GetDebounceTime	66
DrvGPIO_GetIntStatus.....	66
DrvGPIO_InitFunction	67
DrvGPIO_GetDoutBit	68
DrvGPIO_SetBitMask	68
DrvGPIO_ClrBitMask	69
DrvGPIO_SetPortMask	69
DrvGPIO_ReadPortMask	70
DrvGPIO_InstallSR	70
DrvGPIO_GetVersion	71
9. DrvADC 介绍	73
9.1. ADC 介绍	73
9.2. ADC 特性	73
10. DrvADC APIs 说明.....	74
10.1. 类型定义.....	74
10.2. 宏 74	
_DRVADC_CONV	74

10.3. 函数75

DrvADC_Open	75
DrvADC_Close.....	76
DrvADC_SetAdcChannel.....	76
DrvADC_ConfigAdcChannel7	76
DrvADC_SetAdcInputMode	77
DrvADC_SetAdcOperationMode.....	77
DrvADC_SetAdcClkSrc	78
DrvADC_SetAdcDivisor	78
DrvADC_EnableAdcInt.....	79
DrvADC_DisableAdcInt	79
DrvADC_EnableAdcmp0Int	80
DrvADC_DisableAdcmp0Int	80
DrvADC_EnableAdcmp1Int	81
DrvADC_DisableAdcmp1Int	81
DrvADC_GetConversionRate	82
DrvADC_ExtTriggerEnable	82
DrvADC_ExtTriggerDisable	82
DrvADC_StartConvert	83
DrvADC_StopConvert	83
DrvADC_IsConversionDone.....	84
DrvADC_GetConversionData	84
DrvADC_PdmaEnable	84
DrvADC_PdmaDisable	85
DrvADC_IsDataValid	85
DrvADC_IsDataOverrun.....	86
DrvADC_Adcmp0Enable.....	86
DrvADC_Adcmp0Disable.....	87
DrvADC_Adcmp1Enable.....	87
DrvADC_Adcmp1Disable.....	88
DrvADC_SelfCalEnable.....	89
DrvADC_IsCalDone.....	89
DrvADC_SelfCalDisable.....	89
DrvADC_GetVersion	90

11. DrvSPI 介绍.....91

11.1. SPI 介绍..... 91

11.2. 特性91

12. DrvSPI APIs 说明92

12.1. 静态定义..... 92

12.2. 函数93

DrvSPI_Open.....	93
DrvSPI_Close	94
DrvSPI_Set2BitSerialDataIOMode	94
DrvSPI_SetEndian	95
DrvSPI_SetBitLength	95

DrvSPI_SetByteSleep	96
DrvSPI_SetByteEndian	97
DrvSPI_SetTriggerMode	97
DrvSPI_SetSlaveSelectActiveLevel	98
DrvSPI_GetLevelTriggerStatus	99
DrvSPI_EnableAutoCS	99
DrvSPI_DisableAutoCS	100
DrvSPI_SetCS	100
DrvSPI_ClrCS	101
DrvSPI_Busy	102
DrvSPI_BurstTransfer	102
DrvSPI_SetClock	103
DrvSPI_GetClock1	104
DrvSPI_GetClock2	104
DrvSPI_SetVariableClockPattern	105
DrvSPI_SetVariableClockFunction	105
DrvSPI_EnableInt	106
DrvSPI_DisableInt	107
DrvSPI_SingleRead	107
DrvSPI_SingleWrite	108
DrvSPI_BurstRead	108
DrvSPI_BurstWrite	109
DrvSPI_DumpRxRegister	109
DrvSPI_SetTxRegister	110
DrvSPI_SetGo	111
DrvSPI_GetJoyStickIntType	111
DrvSPI_SetJoyStickStatus	112
DrvSPI_GetJoyStickMode	112
DrvSPI_StartPMDA	113
DrvSPI_GetVersion	114

13. DrvI2C 介绍115

13.1. 介绍 115

13.2. 特性 115

14. DrvI2C APIs 说明116

14.1. 函数 116

DrvI2C_Open	116
DrvI2C_Close	116
DrvI2C_SetClock	117
DrvI2C_GetClock	117
DrvI2C_SetAddress	118
DrvI2C_SetAddressMask	118
DrvI2C_GetStatus	119
DrvI2C_WriteData	119
DrvI2C_ReadData	120
DrvI2C_Ctrl	120
DrvI2C_GetIntFlag	121
DrvI2C_ClearIntFlag	121

DrvI2C_EnableInt.....	122
DrvI2C_DisableInt	122
DrvI2C_InstallCallBack	123
DrvI2C_UninstallCallBack.....	123
DrvI2C_EnableTimeoutCount.....	124
DrvI2C_ClearTimeoutFlag	125
15. DrvRTC 介绍	126
15.1.RTC 控制器介绍.....	126
15.2.RTC 特性.....	126
16. DrvRTC APIs 说明	127
16.1.静态定义.....	127
16.2.函数128	
DrvRTC_SetFrequencyCompenation	128
DrvRTC_WriteEnable	128
DrvRTC_Init.....	129
DrvRTC_Open.....	129
DrvRTC_Read	130
DrvRTC_Write	131
DrvRTC_Ioctl	132
DrvRTC_Close	133
DrvRTC_GetVersion	134
17. DrvCAN 介绍	135
17.1.CAN 介绍	135
17.2.CAN 特性	135
18. DrvCAN APIs 说明.....	136
18.1.函数136	
DrvCAN_Open	136
DrvCAN_DisableInt	136
DrvCAN_EnableInt	137
DrvCAN_GetErrorStatus.....	138
DrvCAN_ReadMsg	138
DrvCAN_SetAcceptanceFilter	138
DrvCAN_SetMaskFilter	139
DrvCAN_WaitReady.....	140
DrvCAN_WriteMsg	140
DrvCAN_GetVersion	141
19. DrvPWM 介绍.....	142

19.1.PWM 介绍	142
20. DrvPWM APIs 说明	143
20.1.静态定义.....	143
20.2.函数 144	
DrvPWM_IsTimerEnabled	144
DrvPWM_SetTimerCounter	144
DrvPWM_GetTimerCounter	145
DrvPWM_EnableInt	145
DrvPWM_DisableInt	146
DrvPWM_ClearInt	147
DrvPWM_GetIntFlag	148
DrvPWM_GetRisingCounter	148
DrvPWM_GetFallingCounter	149
DrvPWM_GetCaptureIntStatus	150
DrvPWM_ClearCaptureIntStatus	150
DrvPWM_Open	151
DrvPWM_Close	151
DrvPWM_EnableDeadZone	152
DrvPWM_Enable	152
DrvPWM_SetTimerClk	153
DrvPWM_SetTimerIO	154
DrvPWM_SelectClockSource	155
DrvPWM_GetVersion	156
21. DrvPS2 介绍	157
21.1.PS2 介绍	157
21.2.PS2 特性	157
22. DrvSP2 APIs 说明	158
22.1.宏 158	
DRVPS2_OVERRIDE	158
DRVPS2_PS2CLK	158
DRVPS2_PS2DATA	159
DRVPS2_CLRFIFO	159
DRVPS2_ACKNOTALWAYS	160
DRVPS2_RXINTENABLE	160
DRVPS2_RXINTDISABLE	160
DRVPS2_TXINTENABLE	161
DRVPS2_TXINTDISABLE	161
DRVPS2_PS2ENABLE	162
DRVPS2_PS2DISABLE	162
DRVPS2_TXFIFO	163
DRVPS2_SWOVERRIDE	163
DRVPS2_INTCLR	164

DRVPS2_RXDATA.....	164
DRVPS2_TXDATAWAIT.....	165
DRVPS2_TXDATA.....	165
DRVPS2_TXDATA0.....	166
DRVPS2_TXDATA1.....	166
DRVPS2_TXDATA2.....	167
DRVPS2_TXDATA3.....	167
DRVPS2_ISTXEMPTY.....	168
DRVPS2_ISFRAMEERR.....	168
DRVPS2_ISRXBUSY.....	169

22.2. 函数 169

DrvPS2_Open.....	169
DrvPS2_Close.....	169
DrvPS2_EnableInt.....	170
DrvPS2_DisableInt.....	170
DrvPS2_IsIntEnabled.....	171
DrvPS2_ClearIn.....	171
DrvPS2_GetIntStatus.....	172
DrvPS2_SetTxFIFODepth.....	172
DrvPS2_Read.....	173
DrvPS2_Write.....	173
DrvPS2_GetVersion.....	174

23. DrvFMC 介绍.....175

23.1. 介绍 175

23.2. 特性 175

24. DrvFMC APIs 说明.....176

24.1. 函数 176

DrvFMC_EnableISP.....	176
DrvFMC_BootSelect.....	176
DrvFMC_GetBootSelect.....	177
DrvFMC_EnableLDUpdate.....	177
DrvFMC_EnablePowerSaving.....	178
DrvFMC_ReadCID.....	178
DrvFMC_ReadDID.....	178
DrvFMC_Write.....	179
DrvFMC_Read.....	179
DrvFMC_Erase.....	180
DrvFMC_WriteConfig.....	180
DrvFMC_ReadDataFlashBaseAddr.....	181

25. DrvUSB 介绍.....182

25.1. 介绍 182

25.2. 特性 182

25.3.Call Flow.....	183
---------------------	-----

26. DrvUSB APIs 说明.....184

26.1.宏 184

_DRVUSB_ENABLE_MISC_INT	184
_DRVUSB_ENABLE_WAKEUP	184
_DRVUSB_DISABLE_WAKEUP	185
_DRVUSB_ENABLE_WAKEUP_INT.....	185
_DRVUSB_DISABLE_WAKEUP_INT.....	186
_DRVUSB_ENABLE_FLD_INT	186
_DRVUSB_DISABLE_FLD_INT	187
_DRVUSB_ENABLE_USB_INT	187
_DRVUSB_DISABLE_USB_INT	187
_DRVUSB_ENABLE_BUS_INT	188
_DRVUSB_DISABLE_BUS_INT	188
_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL.....	189
_DRVUSB_CLEAR_EP_READY.....	189
_DRVUSB_SET_SETUP_BUF	190
_DRVUSB_SET_EP_BUF.....	190
_DRVUSB_TRIG_EP	191
_DRVUSB_GET_EP_DATA_SIZE	192
_DRVUSB_SET_EP_TOG_BIT	192
_DRVUSB_SET_EVF	193
_DRVUSB_GET_EVF.....	194
_DRVUSB_CLEAR_EP_STALL	194
_DRVUSB_TRIG_EP_STALL.....	195
_DRVUSB_CLEAR_EP_DSQ	195
_DRVUSB_SET_CFG	196
_DRVUSB_GET_CFG.....	196
_DRVUSB_SET_FADDR.....	197
_DRVUSB_GET_FADDR.....	197
_DRVUSB_SET_STS	198
_DRVUSB_GET_STS	198
_DRVUSB_SET_CFGP	199
_DRVUSB_GET_CFGP	200
_DRVUSB_ENABLE_USB.....	200
_DRVUSB_DISABLE_USB.....	201
_DRVUSB_DISABLE_PHY	201
_DRVUSB_ENABLE_SE0.....	201
_DRVUSB_DISABLE_SE0.....	202
_DRVUSB_SET_CFGP0	202
_DRVUSB_SET_CFGP1	203
_DRVUSB_SET_CFGP2	203
_DRVUSB_SET_CFGP3	204
_DRVUSB_SET_CFGP4	204
_DRVUSB_SET_CFGP5	205

26.2.函数205

DrvUSB_GetVersion	228
DrvUSB_Open.....	205
DrvUSB_Close	207
DrvUSB_PreDispatchEvent.....	207

DrvUSB_Isr_PreDispatchEvent	208
DrvUSB_DispatchEvent	208
DrvUSB_IsData0	208
DrvUSB_GetUsbState	209
DrvUSB_SetUsbState	209
DrvUSB_GetEpIdentity	210
DrvUSB_GetEpId	210
DrvUSB_DataOutTrigger	211
DrvUSB_GetOutData	211
DrvUSB_DataIn	212
DrvUSB_BusResetCallback	212
DrvUSB_InstallClassDevice	213
DrvUSB_InstallCtrlHandler	213
DrvUSB_CtrlSetupAck	214
DrvUSB_CtrlDataInAck	214
DrvUSB_CtrlDataOutAck	215
DrvUSB_CtrlDataInDefault	215
DrvUSB_CtrlDataOutDefault	215
DrvUSB_Reset	216
DrvUSB_ClrCtrlReady	216
DrvUSB_ClrCtrlReadyAndTrigStall	217
DrvUSB_GetSetupBuffer	217
DrvUSB_GetFreeSram	217
DrvUSB_EnableSelfPower	218
DrvUSB_DisableSelfPower	218
DrvUSB_IsSelfPowerEnabled	219
DrvUSB_EnableRemoteWakeup	219
DrvUSB_DisableRemoteWakeup	219
DrvUSB_IsRemoteWakeupEnabled	220
DrvUSB_SetMaxPower	220
DrvUSB_GetMaxPower	221
DrvUSB_EnableUsb	221
DrvUSB_DisableUsb	222
DrvUSB_PreDispatchWakeupEvent	222
DrvUSB_PreDispatchFdtEvent	222
DrvUSB_PreDispatchBusEvent	223
DrvUSB_PreDispatchEPEvent	223
DrvUSB_DispatchWakeupEvent	224
DrvUSB_DispatchMiscEvent	224
DrvUSB_DispatchEPEvent	224
DrvUSB_CtrlSetupSetAddress	225
DrvUSB_CtrlSetupClearSetFeature	225
DrvUSB_CtrlSetupGetConfiguration	226
DrvUSB_CtrlSetupGetStatus	226
DrvUSB_CtrlSetupGetInterface	226
DrvUSB_CtrlSetupSetConfiguration	227
DrvUSB_CtrlDataInSetAddress	227

27. DrvPDMA 介绍.....229

27.1. PDMA 介绍..... 229

27.2. PDMA 特性..... 229

28. DrvPDMA APIs 说明230

28.1. 函数 230

DrvPDMA_Init	230
DrvPDMA_Close	230
DrvPDMA_CHEnableTransfer	231
DrvPDMA_CHSoftwareReset	231
DrvPDMA_Open	232
DrvPDMA_ClearInt	232
DrvPDMA_PollInt	233
DrvPDMA_SetAPBTransferWidth	234
DrvPDMA_SetCHForAPBDevice	234
DrvPDMA_DisableInt	235
DrvPDMA_EnableInt	236
DrvPDMA_GetAPBTransferWidth	236
DrvPDMA_GetCHForAPBDevice	237
DrvPDMA_GetCurrentDestAddr	237
DrvPDMA_GetCurrentSourceAddr	238
DrvPDMA_GetCurrentTransferCount	238
DrvPDMA_GetInternalBufPointer	239
DrvPDMA_GetSharedBufData	239
DrvPDMA_GetTransferLength	240
DrvPDMA_InstallCallBack	241
DrvPDMA_IsCHBusy	241
DrvPDMA_IsIntEnabled	242
DrvPDMA_GetVersion	243

29. Revision History244

1. DrvSYS 介绍

1.1. 介绍

系统管理模块包含下面的功能：

- 系统内存映射
- 系统中断映射
- 产品 ID 寄存器
- 系统管理寄存器，可用于芯片和各个功能模块初始化以及多功能引脚控制.
- Brown-Out 和芯片各种其它的控制.
- 时钟发生器
- 系统时钟和外设时钟
- Power down 模式

2. DrvSYS APIs 说明

2.1. 静态定义

2.1.1. IP 复位

Table 2-1: IP reset

名字	值	描述
E_SYS_GPIO_RST	1	GPIO 复位
E_SYS_TMR0_RST	2	定时器 0 复位
E_SYS_TMR1_RST	3	定时器 1 复位
E_SYS_TMR2_RST	4	定时器 2 复位
E_SYS_TMR3_RST	5	定时器 3 复位
E_SYS_I2C0_RST	8	I2C0 复位
E_SYS_I2C1_RST	9	I2C1 复位
E_SYS_SPI0_RST	12	SPI0 复位
E_SYS_SPI1_RST	13	SPI1 复位
E_SYS_SPI2_RST	14	SPI2 复位
E_SYS_SPI3_RST	15	SPI3 复位
E_SYS_UART0_RST	16	UART0 复位
E_SYS_UART1_RST	17	UART1 复位
E_SYS_PWM_RST	20	PWM 复位
E_SYS_ACMP_RST	22	模拟比较器复位
E_SYS_PS2_RST	23	PS2 复位
E_SYS_CAN0_RST	24	CAN0 复位
E_SYS_CAN1_RST	25	CAN1 复位
E_SYS_USBD_RST	27	USB 设备复位
E_SYS_ADC_RST	28	ADC 复位
E_SYS_PDMA_RST	32	PDMA 复位

2.1.2. IP 时钟使能控制

Table 2-2: IP Clock Enable

名字	值	描述
E_SYS_WD_CLK	0	Watch Dog 时钟使能
E_SYS_RTC_CLK	1	RTC 时钟使能
E_SYS_TMR0_CLK	2	定时器 0 时钟使能
E_SYS_TMR1_CLK	3	定时器 1 时钟使能
E_SYS_TMR2_CLK	4	定时器 2 时钟使能
E_SYS_TMR3_CLK	5	定时器 3 时钟使能
E_SYS_I2C0_CLK	8	I2C0 时钟使能
E_SYS_I2C1_CLK	9	I2C1 时钟使能
E_SYS_SPI0_CLK	12	SPI0 时钟使能
E_SYS_SPI1_CLK	13	SPI1 时钟使能
E_SYS_SPI2_CLK	14	SPI2 时钟使能
E_SYS_SPI3_CLK	15	SPI3 时钟使能
E_SYS_UART0_CLK	16	UART0 时钟使能
E_SYS_UART1_CLK	17	UART1 时钟使能
E_SYS_PWM01_CLK	20	PWM01 时钟使能
E_SYS_PWM23_CLK	21	PWM23 时钟使能
E_SYS_CAN0_CLK	24	CAN0 时钟使能
E_SYS_CAN1_CLK	25	CAN1 时钟使能
E_SYS_USBD_CLK	27	USB 设备时钟使能
E_SYS_ADC_CLK	28	ADC 时钟使能 I
E_SYS_ACMP_CLK	30	模拟比较器时钟使能
E_SYS_PS2_CLK	31	PS2 时钟使能
E_SYS_PDMA_CLK	33	PDMA 时钟使能
E_SYS_ISP_CLK	34	Flash ISP 控制器时钟使能

2.2. 函数

DrvSYS_ReadProductID

原型

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

描述

读取产品 ID.

参数

无

头文件

Driver/DrvSYS.h

返回值

产品 ID

DrvSYS_GetRstSrc

原型

```
uint32_t DrvSYS_GetRstSrc(void);
```

描述

取得最后一次”复位信号”的出处，也就是由哪个 IP 发出的”复位信号”

参数

无

头文件

Driver/DrvSYS.h

返回值

RSTSRC 寄存器的值.比特定义如下:

6	5	4	3	2	1	0
PMU	MCU	BOD	LVR	WDG	PAD	POR

DrvSYS_ClearRstSrc

原型

```
uint32_t DrvSYS_ClearRstSrc(uint32_t u32Src);
```

描述

写 0 清除 RSTSRC 寄存器相应指示标志.

参数

u32Src [in]

要清除的比特

头文件

Driver/DrvSYS.h

返回值

0 成功

DrvSYS_ResetIP

原型

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

描述

复位 IP，包括 GPIO, Timer0, Timer1, Timer2, Timer3, I2C0, I2C1, SPI0, SPI1, SPI2, SPI3, UART0, UART1, PWM, ACMP, PS2, CAN0, CAN1, USB0, ADC, 和 PDMA.

参数

eIpRst [in]

要复位的 IP，参考 E_SYS_IP_RST 的定义.在头文件 Driver/DrvSYS.h 中

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_ResetCPU

原型

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

描述

复位 CPU.

参数

无

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_ResetChip

原型

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

描述

复位整个芯片.

参数

无

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_EnableBOD

原型

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

描述

使能 Brown-Out 功能.

参数

i32Enable [in]

1:enable, 0:disable

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SelectBODVolt

原型

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

描述

选择 BOD 极限电压

参数

u8Volt [in]

可能的值: 3= 4.5V, 2= 3.8V, 1= 2.6V, 0= 2.2V

头文件

Driver/DrvSYS.h

返回值

无.

DrvSYS_EnableBODRst

原型

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

描述

配置当 Brow-out 探测到电压低于极限电压时，发送复位信号还是中断.

参数

i32Enable [in]

1: 使能 Brow-out 复位功能, 0: 使能 Brow-out 中断功能

bodcallbackFn [in]

如果中断功能使能的话，安装中断回调函数.

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_EnableBODLowPowerMode

原型

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

描述

使能 Brow-out low power 模式.

参数

i32Enable [in]

1: low power 模式, 0: normal 模式

头文件

Driver/DrvSYS.h

返回值

无.

DrvSYS_EnableLowVoltRst

原型

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

描述

当输入电压低于 LVR 电路电压时，使能 LVR 复位芯片功能.

参数

i32Enable [in]

1: enable, 0: disable

头文件

Driver/DrvSYS.h

返回值

无.

DrvSYS_GetBODState

原型

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

描述

取得 BOD 状态.

参数

无

头文件

Driver/DrvSYS.h

返回值

1: 检测到的电压低于 BOD 极限电压.

0: 检测到的电压高于 BOD 极限电压

DrvSYS_EnableTempatureSensor

原型

```
void DrvSYS_EnableTempatureSensor(int32_t i32Enable);
```

描述

使能温度传感器功能.

参数

i32Enable [in]

1: enable, 0: disable

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SetPORDisCode

原型

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

描述

写 PORCR 寄存器。这个寄存器是用来控制上电复位信号的。系统上电的时候 POR 电路会产生一个复位信号，但是电源杂讯可能会导致复位信号误发。**PORCR** 寄存器说是用来避免电源 noise, 导致误发 reset signal。POR 电路发 reset signal 的时候会先检测这个寄存器的值？如果是 0x5AA5 就不发 reset signal。那会再次发送 reset signal 的条件是啥为 power on reset 功能使能控制，设定 POR DIS CODE (power-on reset disable code)

参数

u32Code [in]

POD DIS CODE

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_UnlockKeyAddr

原型

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

描述

开启被保护的寄存器. 为了安全考虑, 一些寄存器被加锁, 要写这些寄存器需要先开锁。

参数

无

头文件

Driver/DrvSYS.h

返回值

0 成功

<0 失败

DrvSYS_LockKeyAddr

原型

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

描述

锁上被保护的寄存器.

参数

无

头文件

Driver/DrvSYS.h

返回值

0 成功

<0 失败

DrvSYS_SetRCAdjValue

原型

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

描述

设定电阻电容震荡器(RC oscillator)的调整值.

参数

无

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SetIPClock

原型

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

描述

使能/关闭 IP 时钟，包括看门狗，实时时钟，定时器 0，定时器 1，定时器 2，定时器 3，I2C0, I2C1, SPI0, SPI1, SPI2, SPI3, UART0, UART1, PWM01, PWM23, CAN0, CAN1, USB0, ADC, ACMP, PS2, PDMA 和 Flash ISP 控制器。

参数

eIpClk [in]

要设定时钟的 IP，参考 E_SYS_IP_CLK 的定义。

i32Enable [in]

1: enable, 0: disable

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SetHCLKSource

原型

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

描述

选择 HCLK 时钟源，时钟源可以是外部 12M crystal 时钟，外部 32K crystal 时钟，PLL 时钟，内部 10K oscillator 时钟，或者内部 22M oscillator 时钟。

参数

u8ClkSrcSel [in]

0: 外部 12M 时钟

1: 外部 32K 时钟

- 2: PLL 时钟
- 3: 内部 10K 时钟
- 4~7: 内部 22M 时钟

头文件

Driver/DrvSYS.h

返回值

- 0 成功
- < 0 参数错误

DrvSYS_SetSysTickSource

原型

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

描述

设定 M0 SysTick 时钟源, 可以是外部 32K crystal 时钟, 外部 12M crystal 时钟/2, HCLK/2, 或者 内部 22M oscillator 时钟/2.

参数

u8ClkSrcSel [in]

- 1: 外部 32K 时钟
- 2: 外部 12M 时钟/ 2
- 3: HCLK / 2
- 4~7: 内部 22M 时钟/ 2

头文件

Driver/DrvSYS.h

返回值

- 0 成功
- < 0 参数错误

DrvSYS_SetIPClockSource

原型

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

描述

设定 IP 时钟源，包括看门狗，模数转换器，定时器 0~3, UART, CAN, PWM10, 和 PWM32.

参数

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]

相应 IP 的时钟源，请参考寄存器 CLKSEL1.

头文件

Driver/DrvSYS.h

返回值

0 成功
< 0 参数错误

DrvSYS_SetClockDivider

原型

int32_t DrvSYS_SetClockDivider(E_SYS_IP_DIV eIpDiv , int32_t i32value);

描述

设定 IP 时钟源的除频值.

参数

eIpDiv [in]

E_SYS_ADC_DIV / E_SYS_CAN_DIV / E_SYS_UART_DIV
E_SYS_USB_DIV / E_SYS_HCLK_DIV

i32value [in]

除频值

头文件

Driver/DrvSYS.h

返回值

0 成功
< 0 参数错误

DrvSYS_SetOscCtrl

原型

```
int32_t DrvSYS_SetOscCtrl(E_SYS_OSC_CTRL eOscCtrl, int32_t i32Enable);
```

描述

使能内部 oscillator，外部 crystal，包括内部 10K 和 22M oscillator，外部 32K 和 12M crystal.

参数

eOscCtrl [in]

E_SYS_XTL12M / E_SYS_XTL32K / E_SYS_OSC22M / E_SYS_OSC10K.

i32Enable [in]

1: enable, 0: disable

头文件

Driver/DrvSYS.h

返回值

0 成功
< 0 参数错误

DrvSYS_EnablePWRWUInt

原型

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

描述

使能/关闭唤醒中断。如果 wake up 中断使能的话，将安装回调函数；并且可以使能 64 个时钟延迟来等待 12M crystal 或者 22M oscillator 时钟状态稳定

参数

i32Enable [in]

1: enable, 0: disable

pdwucallbackFn [in]

如果唤醒中断使能的话，安装唤醒中断回调函数

i32enWUDelay [in]

1: 使能 64 个时钟延迟, 0: 关闭 64 个时钟延迟

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_EnablePowerDown

原型

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

描述

使能或者激活系统 power down 功能. 如果 DrvSYS_SetPowerDownWaitCPU(0)被调用, 那么芯片将马上进入 power down 模式; 如果 DrvSYS_SetPowerDownWaitCPU(1) 被调用, 那么 CPU 保持 active 直到 CPU sleep 模式也被激活之后, 芯片才进入 power down 模式. 当芯片进入 power down 模式之后, LDO, 12M crystal, 和 22M oscillator 将被关闭, 但是 32K 和 10K 不受影响.

参数

i32Enable [in]

- 1: 芯片立即进入 power down 模式或者等待 CPU sleep 命令.
- 0: 芯片正常操作.

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SetPowerDownWaitCPU

原型

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

描述

设定 CPU 进入 power down 的条件.

参数

i32Enable [in]

1: DrvSYS_EnablePowerDown(1)被调用，并且 CPU 运行 WFE/WFI 指令之后，芯片才进入 power down 模式.

0: DrvSYS_EnablePowerDown(1)被调用之后，芯片立即进入 power down 模式.

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SetPllSrc

原型

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

描述

选择 PLL 时钟源，可以是内部 22M oscillator 和 外部 12M crystal.

参数

ePllSrc [in]

E_DRVSYS_EXT_12M / E_DRVSYS_INT_22M

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_SetPLLPowerDown

原型

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

描述

使能/关闭 PLL power down 模式.

参数

i32Enable [in]

1: PLL 进入 power down 模式.

0: PLL 进入正常模式.

头文件

Driver/DrvSYS.h

返回值

无

DrvSYS_GetEXTClock

原型

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

描述

取得外部 crystal 时钟频率.单位是 KHz.

参数

无

头文件

Driver/DrvSYS.h

返回值

外部 crystal 时钟频率

DrvSYS_GetPllContent

原型

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

描述

根据外部时钟频率计算最接近的 PLL 时钟频率，然后返回 PLLCON 寄存器的设定值. 用户只要把这个值填到 PLLCON 寄存器，PLL 电路就会振出目标时钟频率了

参数

u32ExtClockKHz [in]

外部时钟频率. 单位是 KHz.

u32PllClockKHz [in]

目标 PLL 时钟频率. 单位是 KHz.

头文件

Driver/DrvSYS.h

返回值

PLLCON 寄存器的设定值.

DrvSYS_GetPLLClock

原型

```
uint32_t DrvSYS_GetPLLClock(void);
```

描述

取得 PLL 输出的时钟频率.

参数

无

头文件

Driver/DrvSYS.h

返回值

PLL 时钟频率, 单位 KHz

DrvSYS_GetHCLK

原型

```
uint32_t DrvSYS_GetHCLK(void);
```

描述

取得 HCLK 时钟频率.

参数

无

头文件

Driver/DrvSYS.h

返回值

HCLK 时钟频率, 单位 KHz

DrvSYS_Open

原型

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

描述

根据外部时钟和目标 PLL 时钟频率，配置 PLLCON 寄存器。由于硬件的限制，实际的 PLL 时钟可能跟目标 PLL 时钟略有不同

参数

u32ExtClockKHz [in]

外部时钟频率. 单位 KHz.

u32PllClockKHz [in]

目标 PLL 时钟频率. 单位 KHz.

头文件

Driver/DrvSYS.h

返回值

0 成功

3. DrvUART 介绍

3.1. 串口介绍

串行异步收发器(UART)从外设收到数据的时候实现串到并转换, 从 CPU 收到数据的时候实现并到串转换.

细节请参考芯片说明书 UART 章节.

3.2. 串口特性

串口包含下面的特性:

- 用作收/发数据缓冲的 64 字节(UART0)/16 字节(UART1) 缓冲区
- 支持自动流控/流控功能(CTS, RTS).
- 完全可编程的串口特性:
 - 5-, 6-, 7-, 或者 8 比特字符
 - 奇、偶或者无校验比特产生和探测
 - 1-, 1&1/2, 或者 2 比特停止位
 - 波特率发生器
 - 错误的起始位探测.
- 用于内部测试的回送模式
- 支持 IrDA SIR 功能
- 支持 LIN 主模式.
- 可编程波特率发生器, 允许时钟除以可编程的分频值

4. DrvUART APIs 说明

4.1. 静态定义

Table 4-1: UART driver constant definition.

名字	值	描述
DRVUART_PORT0	0x000	UART 端口 0
DRVUART_PORT1	0x100000	UART 端口 1
DRVUART_LININT	0x100	LIN RX Break Field Detected 中断使能
DRVUART_WAKEUPINT	0x40	Wake up 中断使能
DRVUART_BUFERRINT	0x20	Buffer Error 中断使能
DRVUART_TOUTINT	0x10	超时中断.
DRVUART_MOSINT	0x8	MODEM 中断
DRVUART_RLSNT	0x4	Receive Line 中断
DRVUART_THREINT	0x2	Transmit Holding Register Empty 中断
DRVUART_RDAINT	0x1	Receive Data Available Interrupt and Time-out 中断
DRVUART_DATABITS_5	0x0	字符长度: 5 比特.
DRVUART_DATABITS_6	0x1	字符长度: 6 比特.
DRVUART_DATABITS_7	0x2	字符长度: 7 比特.
DRVUART_DATABITS_8	0x3	字符长度: 8 比特.
DRVUART_PARITY_EVEN	0x18	使能偶校验
DRVUART_PARITY_ODD	0x08	使能奇校验
DRVUART_PARITY_NONE	0x00	无校验
DRVUART_PARITY_MARK	0x28	Parity mask
DRVUART_PARITY_SPACE	0x38	Parity space
DRVUART_STOPBITS_1	0x000	停止位长度: 1 比特.
DRVUART_STOPBITS_1_5	0x4	停止位长度: 当字符长度是 5 比特时, 停止位长度 1.5 比特
DRVUART_STOPBITS_2	0x4	停止位长度: 当字符长度是 6, 7, 8 比特时, 停止位长度 2 比特
DRVUART_FIFO_1BYTES	0x00	接收缓冲区中断触发级别是 1 个字节
DRVUART_FIFO_4BYTES	0x10	接收缓冲区中断触发级别是 4 个字节
DRVUART_FIFO_8BYTES	0x20	接收缓冲区中断触发级别是 8 个字节
DRVUART_FIFO_14BYTES	0x30	接收缓冲区中断触发级别是 14 个字节
DRVUART_FIFO_30BYTES	0x40	接收缓冲区中断触发级别是 30 个字节
DRVUART_FIFO_46BYTES	0x50	接收缓冲区中断触发级别是 46 个字节

DRVUART_FIFO_62BYTES	0x60	接收缓冲区中断触发级别是 62 个字节
----------------------	------	---------------------

4.2. 函数

DrvUART_Open

原型

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

描述

初始化串口

参数

Port [in]

说明串口: UART_PORT0/UART_PORT1

sParam [in]

说明串口的特性。包括

u32BaudRate: 波特率

u8cParity: 无/奇/偶校验

u8cDataBits: DRVUART_DATA_BITS_5 , DRVUART_DATA_BITS_6, DRVUART_DATA_BITS_7 或者 DRVUART_DATA_BITS_8

u8cStopBits: DRVUART_STOPBITS_1 , STOPBITS_1_5 或者 DRVUART_STOPBITS_2

u8RxTriggerLevel: LEVEL_1_BYTE 到 LEVEL_62_BYTES

u8TimeOut: 超时时间

头文件

Driver/DrvUART.h

返回值

E_SUCCESS: 成功.

E_DRVUART_ERR_PORT_INVALID: 端口错误

E_DRVUART_ERR_PARITY_INVALID: 校验设定错误

E_DRVUART_ERR_DATA_BITS_INVALID: 数据比特错误

E_DRVUART_ERR_STOP_BITS_INVALID: 停止位设定错误

E_DRVUART_ERR_TRIGGERLEVEL_INVALID: 缓冲区触发级别错误

DrvUART_Close

原型

```
void DrvUART_Close (
    UART_PORT    Port
);
```

描述

关闭串口时钟，中断和串口功能。

参数

Port [in]

说明串口：UART_PORT0/UART_PORT1

头文件

Driver/DrvUART.h

返回值

无

DrvUART_EnableInt

原型

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

描述

使能串口中断并且安装中断回调函数

参数

u8Port [in]

说明串口： UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected 中断

DRVUART_BUFERRINT : Buffer Error 中断

DRVUART_WAKEINT : Wakeup 中断.

DRVUART_MOSINT : MODEM Status 中断.

DRVUART_RLSNT : Receive Line Status 中断.

DRVUART_THREINT : Transmit Holding Register Empty 中断.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out 中断

DRVUART_TOUTINT : Time-out 中断.

pfncallback [in]

回调函数指针

头文件

Driver/DrvUART.h

返回值

无

Note

使用“/”运算符可以同时使能多个中断.

DrvUART_IsIntEnabled

原型

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

描述

取得中断使能标志

参数

u16Port [in]

说明串口: UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected 中断

DRVUART_BUFERRINT : Buffer Error 中断

DRVUART_WAKEINT : Wakeup 中断.

DRVUART_MOSINT : MODEM Status 中断.

DRVUART_RLSNT : Receive Line Status 中断.

DRVUART_THREINT : Transmit Holding Register Empty 中断.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out 中断

DRVUART_TOUTINT : Time-out 中断

头文件

Driver/DrvUART.h

返回值

是 1 的比特表示相应中断是使能的, 否则是关闭的

Note

推荐一次只查询一个中断.

DrvUART_DisableInt

原型

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

描述

关闭中断并且卸载相应的中断回调函数

参数

u16Port [in]

说明串口: UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected 中断

DRVUART_BUFERRINT : Buffer Error 中断

DRVUART_WAKEINT : Wakeup 中断.

DRVUART_MOSINT : MODEM Status 中断.

DRVUART_RLSNT : Receive Line Status 中断.

DRVUART_THREINT : Transmit Holding Register Empty 中断.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out 中断

DRVUART_TOUTINT : Time-out 中断.

头文件

Driver/DrvUART.h

返回值

无

Note

使用“/” 运算符可以同时关闭多个中断.

DrvUART_ClearInt

原型

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

描述

用来清除串口中断状态

参数

u16Port [in]

说明串口: UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected 中断

DRVUART_BUFERRINT : Buffer Error 中断

DRVUART_WAKEINT : Wakeup 中断.

DRVUART_MOSINT : MODEM Status 中断.

DRVUART_RLSNT : Receive Line Status 中断.

DRVUART_THREINT : Transmit Holding Register Empty 中断.

DRVUART_RDAINT : Receive Data Available 中断.

DRVUART_TOUTINT : Time-out 中断.

头文件

Driver/DrvUART.h

返回值

E_SUCESS 成功

DrvUART_GetIntStatus

原型

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

描述

这个函数用来取得中断状态

参数

u16Port [in]

说明串口: UART_PORT0/UART_PORT1

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected 中断

DRVUART_BUFERRINT : Buffer Error 中断

DRVUART_WAKEINT : Wakeup 中断.

DRVUART_MOSINT : MODEM Status 中断.

DRVUART_RLSNT : Receive Line Status 中断.

DRVUART_THREINT : Transmit Holding Register Empty 中断.

DRVUART_RDAINT : Receive Data Available 中断.

DRVUART_TOUTINT : Time-out 中断.

头文件

Driver/DrvUART.h

返回值

0: 无.

1: 有中断发生.

Note

一次只能查询一个中断.

DrvUART_SetFIFOTriggerLevel

原型

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

描述

这个函数可以用来设定接收 FIFO 触发级别

参数

u16Port [in]

说明串口：UART_PORT0/UART_PORT1

u32TriggerLevel [in]

接收缓冲区中断触发级别.

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.

头文件

Driver/DrvUART.h

返回值

无

DrvUART_GetCTS

原型

```
void
DrvUART_GetCTS (
    UART_PORT    u16Port,
    uint8_t      *pu8CTSValue,
    uint8_t      *pu8CTSChangeState
}
```

描述

这个函数可以用来取得 CTS 值并且改变状态

参数

u16Port [in]

说明串口：UART_PORT0/UART_PORT1

pu8CTSValue [in]

说明存放 CTS 值的缓存地址

pu8CTSChangeState [in]

说明存放 CTS 改变状态的缓存地址

头文件

Driver/DrvUART.h

返回值

无

DrvUART_SetRTS

原型

```
void
DrvUART_GetCTS (
    UART_PORT    u16Port,
    uint8_t      u8Value
)
```

描述

这个函数可以用来设定 RTS 信息

参数

u16Port [in]

说明串口：UART_PORT0/UART_PORT1

u8Value [in]

说明 RTS 值

头文件

Driver/DrvUART.h

返回值

无

DrvUART_SetRxTimeOut

原型

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

描述

这个函数可以用来设定接收超时时间

参数

u16Port [in]

说明串口：UART_PORT0/UART_PORT1

u8TimeOut [in]

说明超时时间，单位是 1/波特率秒

头文件

Driver/DrvUART.h

返回值

无

DrvUART_Read

原型

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

描述

这个函数用来从接收缓存中读取数据

参数

u16Port [in]

说明串口：UART_PORT0/UART_PORT1

pu8RxBuf [out]

说明存放接收到的数据的缓存指针.

u32ReadBytes [in]

说明要接收的字节数

头文件

Driver/DrvUART.h

返回值

E_SUCCESS: 成功.

E_DRVUART_TIMEOUT: 超过轮询次数仍没有收到足够的字符.

DrvUART_Write

原型

```
int32_t
DrvUART_Write(
    UART_PORT    u16Port
    uint8_t       *pu8TxBuf,
    uint32_t      u32WriteBytes
);
```

描述

这个函数可用来写数据到发送缓存，然后由串口发送出去

参数

u16Port [in]

说明串口：UART_PORT0/UART_PORT1

pu8TxBuf [in]

说明要发送的数据指针

u32WriteBytes [in]

要发送的字节数.

头文件

Driver/DrvUART.h

返回值

E_SUCCESS: 成功

E_DRVUART_TIMEOUT: 超过轮询次数发送缓冲仍不为空

DrvUART_SetPDMA

原型

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

描述

使能/关闭 PDMA 通道

参数

u16Port **[in]**
说明串口: UART_PORT0/UART_PORT1

IsEnable **[in]**
使能/关闭

头文件

Driver/DrvUART.h

返回值

E_SUCCESS: 成功

DrvUART_OpenIRCR

原型

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

描述

这个函数用来设定 IRCR 控制寄存器

参数

u16Port [in]

说明串口: UART_PORT0/UART_PORT1

str_IRCR [in]

IrDA 结构体包括

u8cRXSelect : Select Rx function

u8cTXSelect : Select Tx function

u8cInvTX : Invert Tx signal

u8cInvRX : Invert Rx signal

头文件

Driver/DrvUART.h

返回值

无

DrvUART_OpenLIN

原型

```
void
DrvUART_OpenLIN (
    UART_PORT    u16Port
    uint16_t DIRECTION,
    uint16_t BCNT
);
```

描述

这个函数用来设定与 LIN 相关的设定

参数

u16Port [in]

说明串口: UART_PORT0/UART_PORT1

DIRECTION [in]

说明 LIN 方向 : _MODE_TX , _MODE_RX

BCNT [in]

说明 break count value

头文件

Driver/DrvUART.h

返回值

无

DrvUART_GetVersion

原型

int32_T
DrvUART_GetVersion (void);

描述

返回当前版本号.

头文件

Driver/DrvUART.h

返回值

版本号:

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

5. DrvTIMER 介绍

5.1. 定时器介绍

定时器模块包含 4 个通道：TIMER0~TIMER3 (TIMER0, TIMER1 在 AHB1 总线上，TIMER2，TIMER3 在 AHB2 总线上)。用户利用它们可以很容易的实现计数功能、频率管理、事件计数、间隔管理、时钟发生器、延迟等功能。定时器有一些特性：分辨率可调、计数周期可编程等。定时器在超时的时候可以发生中断或者提供计数寄存器的值。定时器模块也提供看门狗功能来处理系统崩溃事件。

5.2. 定时器特性

定时器包含下面的特性：

- 与 AMBA APB 总线兼容
- 每个通道有一个 8 比特预分频和一个中断请求信号
- 每个通道有独立的时钟源(TCLK0,TCLK1,TCLK2,TCLK3)
- 最大不中断时间= $(1 / 25 \text{ MHz}) * (2^8) * (2^{24} - 1)$ 当 TCLK = 25 MHz

6. DrvTIMER APIs 说明

6.1. 函数

DrvTIMER_GetStatus

原型

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

描述

这个函数用来取得定时器的中断状态.

参数

channel [in]

定时器通道: TMR0/ TMR1/ TMR2/ TMR3.

头文件

Driver/DrvTimer.h

返回值

- 1: 相应的定时器通道发生了中断.
- 0: 相应的定时器通道没有中断发生.

DrvTIMER_SetTimerEvent

原型

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

描述

这个函数可以用来安装一个定时处理事件到 timer0, timer1, timer2, timer3 通道.

参数

channel [in]

TMR0/ TMR1/ TMR2/ TMR3

uTimeTick [in]

执行事件的间隔。单位：定时器 tick

pvFun [in]

事件处理函数指针。

parameter [in]

传给事件处理函数的参数。

头文件

Driver/DrvTimer.h

返回值

事件索引。可以是 0 ~ TIMER_EVENT_COUNT-1

DrvTIMER_ClearTimerEvent

原型

```
void DrvTIMER_ClearTimerEvent(
    TIMER_CHANNEL  channel,
    uint32_t        uTimeEventNo
);
```

描述

这个函数可以用来移除安装的定时处理事件。

参数

channel [in]

定时器通道 TMR0/ TMR1/ TMR2/ TMR3。

uTimeEventNo [in]

事件索引。可以是 0 ~ TIMER_EVENT_COUNT-1。

头文件

Driver/DrvTimer.h

返回值

无

DrvTIMER_ResetTicks

原型

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

描述

这个函数可以用来复位定时器 tick 计数.

参数

channel [in]

定时器通道 TMR0/ TMR1/ TMR2/ TMR3.

头文件

Driver/DrvTimer.h

返回值

E_SUCCESS 成功

E_DRVTIMER_CHANNEL 不支持的定时器通道

DrvTIMER_Init

原型

```
void DrvTIMER_Init(void);
```

描述

这个函数可以用来初始化定时器.

参数

无

头文件

Driver/DrvTimer.h

返回值

无

DrvTIMER_Open

原型

```
int32_t DrvTIMER_Open(
    TIMER_CHANNEL channel,
    uint32_t              uTicksPerSecond,
```

```
TIMER_OPMODE    mode
);
```

描述

这个函数可以用来设定定时器 tick 周期并且启动定时器.

参数

channel [in]

定时器通道 TMR0/ TMR1/ TMR2/ TMR3.

uTickPerSecond [in]

每秒 tick 数.

Mode [in]

操作模式: One-Shot / Periodic / Toggle. 可以是 ONESHOT_MODE, PERIODIC_MODE ,TOGGLE_MODE 或者 UNINTERREUPT_MODE.

头文件

Driver/DrvTimer.h

返回值

E_SUCCESS	成功.
E_DRVTIMER_CMD	命令错误.
E_DRVTIMER_EIO	定时器没有被 DrvTIMER_Init()初始化.

DrvTIMER_GetTicks

原型

```
uint32_t DrvTIMER_GetTicks(TIMER_CHANNEL channel);
```

描述

这个函数可以用来取得当前定时器 tick 数.

参数

channel [in]

定时器通道: TMR0/ TMR1/ TMR2/ TMR3.

头文件

Driver/DrvTimer.h

返回值

返回定时器 TIMER0, TIMER1, TIMER2 , TIMER3 当前的 tick 数.

DrvTIMER_Delay

原型

```
void DrvTIMER_Delay (uint32_t uTicks);
```

描述

这个函数可以用来设定一个延迟. 这个函数使用定时器 **TIMER0**, 所以它应该先被初始化并且打开.

参数

uTicks [in]
延迟时间 tick 数, 单位是定时器 **Timer0** 的 tick.

头文件

```
Driver/DrvTimer.h
```

返回值

无

DrvTIMER_Ioctl

原型

```
int32_t DrvTIMER_Ioctl(  
    TIMER_CHANNEL channel,  
    TIMER_CMD      uCmd,  
    UINT32          uArg1,  
);
```

描述

处理定时器的一般性控制。下面的表格列出有效的命令，参数和命令相关描述.

米几年工龄	参数	描述
TIMER_IOC_START_COUNT	无	开始计数
TIMER_IOC_STOP_COUNT	无	停止计数
TIMER_IOC_ENABLE_INT	无	使能定时器中断
TIMER_IOC_DISABLE_INT	无	关闭定时器中断
TIMER_IOC_RESET_TIMER	无	复位计数器并停止计数
TIMER_IOC_SET_PRESCALE	uArg1	uArg1 是计数器的预分频值. 这个值可以是 0 ~ 255 , 可以将计数器的时钟除以 1 ~ 256.
TIMER_IOC_SET_INITIAL_COUNT	uArg1	这个命令可以用来说明计数器的初始值. 由于计数器是 16 比特的, 所以 uArg1 的范围是 0 ~ 65535. 当开始计数的时候, 计数器将从这个初始值开始递减到 0.

参数

channel [in]

定时器通道 TMR0/ TMR1/ TMR2/ TMR3.

uCmd [in]

命令, e.x. TIMER_IOC_START_COUNT.

uArg1 [in]

特别命令的第一个参数.

头文件

Driver/DrvTimer.h

返回值

E_SUCCESS	成功.
E_DRVTIMER_CMD	命令无效.

DrvTIMER_Close

原型

```
int32_t DrvTIMER_Close(TIMER_CHANNEL channel);
```

描述

这个函数用来关闭定时器.

参数

channel [in]

定时器通道 TMR0/ TMR1/ TMR2/ TMR3.

头文件

Driver/DrvTimer.h

返回值

E_SUCCESS	成功.
E_DRVTIMER_CMD	定时器通道无效.

DrvWDT_Open

原型

```
int32_t DrvWDT_Open(int32_t handler ,WDT_INTERVAL level);
```

描述

这个函数可用来设定看门狗间隔并且启动看门狗功能.

参数

hander [in]

预留.

level [in]

看门狗超时级别. 可以是 LEVEL0 ~ 7 .

头文件

Driver/DrvTimer.h

返回值

E_SUCESS 成功

DrvWDT_ResetCount

原型

void DrvWDT_ResetCount(void);

描述

这个函数可以用来复位看门狗，避免超时重启系统.

参数

无

头文件

Driver/DrvTimer.h

返回值

无

DrvWDT_Ioctl

原型

int32_T DrvWDT_Ioctl(int32_t hander ,WDT_CMD uCmd , uint32_t uArg1);

描述

这个函数用来控制看门狗定时器.

参数

hander [in]

预留.

uCmd [in]

WDT_IOCTL 命令.

uArg1 [in]

命令的第一个参数.

头文件

Driver/DrvTimer.h

返回值

E_SUCCESS	成功
E_DRVTIMER_CMD	无效的 I/O 命令

DrvWDT_Close

原型

void DrvWDT_Close(void);

描述

这个函数用来停止看门狗定时器并且关闭看门狗中断

参数

无

头文件

Driver/DrvTimer.h

返回值

无

DrvTIMER_GetVersion

原型

uint32_t
DrvTimer_GetVersion (void);

描述

返回当前驱动版本号.

头文件

Driver/DrvTimer.h

返回值

版本号 :

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

7. DrvGPIO 介绍

7.1. GPIO 介绍

- GPIO 和其它某些功能共享 80 个引脚.
- 每个 GPIO 引脚可以单独编程为输入、输出、open-drain 或者 quasi-bidirectional 模式.
- 所有的 GPIO 功能都可以通过软件编程达到.

8. DrvGPIO APIs 说明

8.1. 函数

DrvGPIO_Open

原型

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

描述

配置指定的 GPIO 端口.

参数

port [in]

配置指定的 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE

i32Bit [in]

指定 GPIO 端口的某个比特. 可以是 0~15.

mode [in]

设定 GPIO 端口为 IO_INPUT , IO_OUTPUT ,IO_OPENDRAIN 或者 IO_QUASI.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS	成功.
E_DRVGPIO_ARGUMENT	参数错误.
E_DRVGPIO_BUSY	IO 已经被用了.

DrvGPIO_Close

原型

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

描述

这个函数用来关闭 GPIO 端口，并且复位配置信息。

参数

port [in]

指定 GPIO 端口。可以是 GPA, GPB , GPC , GPD , GPE

i32Bit [in]

指定 GPIO 端口比特。可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS	成功.
E_DRVGPIO_ARGUMENT	参数错误.

DrvGPIO_SetBit

原型

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

描述

设定指定的比特输出 1.

参数

port [in]

指定 GPIO 端口。可以是 GPA, GPB , GPC , GPD , GPE

i32Bit [in]

指定 GPIO 端口的比特。可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS	成功.
E_DRVGPIO_ARGUMENT	参数错误.

DrvGPIO_ClrBit

原型

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

描述

让指定的 GPIO 比特输出 0.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定 GPIO 端口比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS	成功.
E_DRVGPIO_ARGUMENT	参数错误.

DrvGPIO_GetBit

原型

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

描述

取得指定的 GPIO 比特的值.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定的比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

指定的比特值.

DrvGPIO_SetPortBits

原型

```
int32_tDrvGPIO_SetPortBits(DRVGPIO_PORT port, int32_tData);
```

描述

写数据到指定的 GPIO 输出寄存器.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE..

i32Data [in]

写到 GPIO 端口的数据.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS	成功.
E_DRVGPIO_ARGUMENT	参数错误.

DrvGPIO_GetPortBits

原型

```
INT32 DrvGPIO_GetPortBits(DRVGPIO_PORT port);
```

描述

取得指定 GPIO 端口的数据, 它反映了各个引脚的状态.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

头文件

Driver/DrvGPIO.h

返回值

GPIO 端口的数据.

DrvGPIO_GetPortDoutBits

原型

```
int32_t DrvGPIO_GetPortDoutBits(DRVGPIO_PORT port);
```

描述

取得指定端口的 DOUT 寄存器的值.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

头文件

Driver/DrvGPIO.h

返回值

GPIO DOUT 寄存器的值.

E_DRVGPIO_ARGUMENT 参数错误.

DrvGPIO_EnableInt

原型

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

描述

使能指定端口的指定比特的中断功能。.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定比特. 可以是 0~15.

tiggerType [in]

说明中断触发类型. 可以是 IO_RISING, IO_FALLING 和 IO_BOTH_EDGE.

Mode [in]

说明中断模式. 可以是 MODE_EDGE, IO_FALLING 和 MODE_LEVEL.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_DisableInt

原型

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

描述

关闭指定端口的指定比特的中断功能

参数

port [in]
指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.
i32Bit [in]
指定比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_SetDebounceTime

原型

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

描述

设定 debounce timing 并且选择 debounce 时钟源.

参数

i32DebounceClk [in]
debounce timing = $2^{(i32DebounceClk)} * \text{APB 时钟}$.
i8ClockSource [in]

debounce 时钟源. 可以是 DBCLKSRC_HCLK or DBCLKSRC_10K.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_EnableDebounce

原型

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

描述

使能指定 GPIO 端口的指定比特的 debounce 功能

参数

port [in]
指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.
i32Bit [in]
指定比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_DisableDebounce

原型

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

描述

关闭指定 GPIO 端口的指定比特的 debounce 功能

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.

E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_GetDebounceTime

原型

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

描述

取得 debounce timing 设定值.

参数

无

头文件

Driver/DrvGPIO.h

返回值

debounce timing 设定值.

DrvGPIO_GetIntStatus

原型

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

描述

这个函数可以用来返回指定端口的中断状态寄存器的值

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

头文件

Driver/DrvGPIO.h

返回值

指定 GPIO 端口的中断状态寄存器的值

DrvGPIO_InitFunction

原型

```
int32_t DrvGPIO_InitFunction (DRVGPIFUNC function);
```

描述

给多功能引脚指定功能

参数

function [in]

指定多功能引脚的功能。可以是：

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.

每个功能对应的 GPIO 引脚说明如下：

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

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_GetDoutBit

原型

INT32 DrvGPIO_GetDoutBit (DRVGPIO_PORT port, INT32 i32Bit);

描述

取得指定端口的指定比特的输出值（读 GPIO Dout 寄存器）.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

制定比特的输出值

DrvGPIO_SetBitMask

原型

int32_t DrvGPIO_SetBitMask(DRVGPIO_PORT port, int32_t i32Bit);

描述

设定指定 GPIO 端口的指定比特输出掩码。相应的 DOUT 比特将被保护，对这些比特写将被忽略

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.

DrvGPIO_ClrBitMask

原型

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

描述

清除指定端口的指定比特的掩码.

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定比特. 可以是 0~15.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.

DrvGPIO_SetPortMask

原型

```
int32_t DrvGPIO_SetPortMask(DRVGPIO_PORT port, uint32_t mask);
```

描述

设定指定 GPIO 端口的掩码寄存器. 与 DrvGPIO_SetBitMask 的区别只在于, DrvGPIO_SetBitMask 只能设定一个 bit, 而 DrvGPIO_SetPortMask 可以设定整个寄存器

参数

port [in]

指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

mask [in]

写到指定 GPIO 端口的掩码寄存器的值.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_ReadPortMask

原型

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

描述

取得指定 GPIO 端口的掩码值.

参数

port [in]
指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

头文件

Driver/DrvGPIO.h

返回值

指定 GPIO 端口的掩码值

DrvGPIO_InstallISR

原型

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

描述

安装指定 GPIO 端口的中断回调函数.

参数

port [in]
指定 GPIO 端口. 可以是 GPA, GPB , GPC , GPD , GPE.

i32Bit [in]

指定比特. 可以是 0~15.

pvFun [in]

回调函数指针.

parameter [in]

传给回调函数的参数.

头文件

Driver/DrvGPIO.h

返回值

E_SUCCESS 成功.
E_DRVGPIO_ARGUMENT 参数错误

DrvGPIO_GetVersion

原型

UINT32
DrvGPIO_GetVersion (VOID);

描述

返回驱动版本号.

头文件

Driver/DrvGPIO.h

返回值

版本号:

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

9. DrvADC 介绍

9.1. ADC 介绍

10 比特模数转换器(ADC) 是一个连续逼近型的 ADC，有 8 个通道的输入。ADC 可以运行在两个模式下：一个是正常的 ADC 模式；另一个是录音模式。两个模式不可以同时工作。

转换一个采样需要 20 个 ADC 时钟，ADC 最大输入时钟是 20M (5V)。A/D 转换支持三种操作模式：单个模式、单循环扫描模式和连续扫描模式。A/D 转换可以由软件启动，也可以由外部的 STADC/PB.8 引脚来启动。

注意：在 ADC 功能使能之前，模拟输入引脚必须配置成输入类型。

9.2. ADC 特性

模数转换器包含下面的特性：

- 模拟输入电压范围: 0~Vref (Max to 5.0V)
- 12 比特分辨率，10 比特精度
- 8 个模拟输入通道
- 最大 ADC 时钟频率是 20MHz
- 三种操作模式
 1. 单个通道模式
 2. 所有使能的通道扫描一次模式
 3. 所有使能的通道依次循环扫描模式
- A/D 转换可以由下列动作启动
 1. 软件写 1 到 ADST 比特
 2. 外部引脚 STADC
- 转换结果可以跟指定的值比较，如果匹配，用户可以选择是否产生一个中断
- 通道 7 支持 3 个输入源：外部模拟电压，内部固定带隙电压和内部温度传感输出
- 支持自校正以减小转换错误
- 支持 single end 和 differential 输入信号

10. DrvADC APIs 说明

10.1. 类型定义

Table 10-1: Type definition of ADC driver.

类型	值	描述
ADC_INPUT_MODE	ADC_SINGLE_END (0)	ADC single end 输入
	ADC_DIFFERENTIAL (1)	ADC differential 输入
ADC_OPERATION_MODE	ADC_SINGLE_OP (0)	单个通道模式
	ADC_SINGLE_CYCLE_OP (1)	所有使能的通道扫描一次模式
	ADC_CONTINUOUS_OP (2)	所有使能的通道依次循环扫描模式
ADC_CLK_SRC	EXT_12MHZ (0)	外部 12MHz 时钟
	INT_PLL (1)	内部 PLL 时钟
	INT_RC22MHZ (2)	内部 22MHz 时钟
ADC_EXT_TRI_COND	LOW_LEVEL (0)	Low level 触发
	HIGH_LEVEL (1)	High level 触发
	FALLING_EDGE (2)	Falling edge 触发
	RISING_EDGE (3)	Rising edge 触发
ADC_CH7_SRC	EXT_INPUT_SIGNAL (0)	外部输入信号 I
	INT_BANDGAP (1)	内部 bandgap 电压
	INT_TEMPERATURE_SENSOR (2)	内部温度传感器
ADC_COMP_CONDITION	LESS_THAN (0)	小于比较数据
	GREATER_OR_EQUAL (1)	大于等于比较数据

10.2. 宏

_DRVADC_CONV

原型

```
VOID _DRVADC_CONV (VOID);
```

描述

通知 ADC 开始转换输入电压到数字值。

头文件

Driver/DrvADC.h

返回值

无.

10.3. 函数

DrvADC_Open

原型

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

描述

使能 ADC 功能并且完成相关设定.

参数

InputMode [in]

说明模拟信号输入类型. 可以是 single-end 或者 differential 输入.

OpMode [in]

说明操作模式. 可以是 single, single cycle scan 或者 continuous scan mode.

u8ChannelSelBitwise [in]

指定输入通道.

ClockSrc [in]

指定 ADC 时钟源

u8AdcDivisor [in]

决定 ADC 时钟频率.

$\text{ADC 时钟频率} = \text{ADC 时钟源频率} / (\text{AdcDivisor} + 1)$

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_Close

原型

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

描述

关闭 ADC 功能. 关闭 ADC 时钟和 ADC 中断.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_SetAdcChannel

原型

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

描述

设定 ADC 输入通道.

参数

u8ChannelSelBitwise [in]

指定模拟信号输入通道.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_ConfigAdcChannel7

原型

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

描述

选择通道 7 的信号源。有三种选择：外部模拟电压，内部固定带隙电压和内部温度传感器输出

参数

Ch7Src [in]

指定模拟信号输入源.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_SetAdcInputMode

原型

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

描述

设定 ADC 输入模式.

参数

InputMode [in]

说明 ADC 输入模式. 可以是 single-end 或者 differential 输入

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_SetAdcOperationMode

原型

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

描述

设定 ADC 操作模式.

参数

OpMode [in]

说明操作模式.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_SetAdcClkSrc

原型

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

描述

设定 ADC 时钟源.

参数

ClockSrc [in]

说明 ADC 时钟源.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_SetAdcDivisor

原型

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

描述

设定 ADC 时钟的除数进行除频.

参数

u8AdcDivisor [in]

说明除数值.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_EnableAdcInt

原型

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

描述

使能 ADC 中断并且安装中断回调函数.

参数

Callback [in]

回调函数指针.

u32UserData [in]

传给回调函数的参数.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_DisableAdcInt

原型

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

描述

关闭 ADC 中断.

参数

无

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_EnableAdcmp0Int

原型

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

描述

使能 ADC 比较器 0 的比较匹配中断，并且安装中断回调函数.

参数

Callback [in]

回调函数指针.

u32UserData [in]

传给回调函数的参数.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_DisableAdcmp0Int

原型

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

描述

关闭 ADC 比较器 0 的比较匹配中断.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_EnableAdcmp1Int

原型

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

描述

使能 ADC 比较器 1 的比较匹配中断，并且安装中断回调函数。

参数

Callback [in]

回调函数指针。

u32UserData [in]

传给回调函数的参数。

头文件

Driver/DrvADC.h

返回值

无。

DrvADC_DisableAdcmp1Int

原型

```
void DrvAdc_DisableAdcmp1Int (void);
```

描述

关闭 ADC 比较器 1 的比较匹配中断。

参数

无。

头文件

Driver/DrvADC.h

返回值

无。

DrvADC_GetConversionRate

原型

```
uint32_t DrvADC_GetConversionRate (void);
```

描述

取得 A/D 转换的频率.

参数

无.

头文件

Driver/DrvADC.h

返回值

返回转换频率.

DrvADC_ExtTriggerEnable

原型

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

描述

使能外部触发引脚(PB8) 做 ADC 的触发源.

参数

TriggerCondition [in]

说明触发条件. 触发条件可以是 low-level / high-level / falling-edge / positive-edge.

头文件

Driver/DrvADC.h

返回值

无

DrvADC_ExtTriggerDisable

原型

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

描述

禁止外部 ADC 触发.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_StartConvert

原型

void DrvADC_StartConvert(void);

描述

开始 A/D 转换.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_StopConvert

原型

void DrvADC_StopConvert(void);

描述

停止 A/D 转换.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_IsConversionDone

原型

```
UInt32_t DrvADC_IsConversionDone (void);
```

描述

检查转换是否完成.

参数

无.

头文件

Driver/DrvADC.h

返回值

TURE 转换完成

FALSE 正在转换

DrvADC_GetConversionData

原型

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

描述

取得指定 ADC 通道的转换结果数据.

参数

u8ChannelNum [in]

说明 ADC 通道.

头文件

Driver/DrvADC.h

返回值

12 比特转换结果.

E_DRVADC_CHANNELNUM: 通道号错误.

DrvADC_PdmaEnable

原型

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

描述

使能 PDMA 传输.

参数

无.

头文件

Driver/DrvADC.h

返回值

无

DrvADC_PdmaDisable

原型

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

描述

关闭 PDMA 传输.

参数

无.

头文件

Driver/DrvADC.h

返回值

无

DrvADC_IsDataValid

原型

```
Uin32_t DrvADC_IsDataValid (uint8_t u8ChannelNum);
```

描述

检查 A/D 转换的数据是否有效.

参数

u8ChannelNum [in]

说明 A/D 通道号.

头文件

Driver/DrvADC.h

返回值

TURE	数据有效
FALSE	数据无效

DrvADC_IsDataOverrun

原型

```
UInt32_t DrvADC_IsDataOverrun (uint8_t u8ChannelNum);
```

描述

检查转换的数据是否溢出

参数

u8ChannelNum [in]

说明 ADC 通道号.

头文件

Driver/DrvADC.h

返回值

TURE	溢出
FALSE	没有溢出

DrvADC_Adcmp0Enable

原型

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

描述

使能 ADC 转换结果监控功能 0. 转换的结果将与比较寄存器的值比较。并且配置相关寄存器.

参数

u8CmpChannelNum [in]

说明想要比较的通道号.

CmpCondition [in]

说明比较条件（大于等于(>=)还是小于(<)）.

u16CmpData [in]

说明比较数据.

u8CmpMatchCount [in]

说明比较匹配总数。如果匹配数达到设定值，ADC 硬件将设定 CMPF 比特

头文件

Driver/DrvADC.h

返回值

E_SUCCESS 成功. 比较功能被打开.

E_DRVADC_ARGUMENT 参数错误

DrvADC_Adcmp0Disable

原型

void DrvADC_Adcmp0Disable (void);

描述

关闭 ADC 转换结果比较功能 0.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_Adcmp1Enable

原型

```
ERRCODE DrvADC_Adcmp1Enable (
    uint8_t u8CmpChannelNum,
    DC_COMP_CONDITION CmpCondition,
    uint16_t u16CmpData,
```

```
uint8_t u8CmpMatchCount
);
```

描述

使能 ADC 转换结果监控功能 1. 转换的结果将与比较寄存器的值比较。并且配置相关寄存器。

参数

u8CmpChannelNum [in]

说明想要比较的通道号。

CmpCondition [in]

说明比较条件（大于等于(>=)还是小于(<)）。

u16CmpData [in]

说明比较数据。

u8CmpMatchCount [in]

说明比较匹配总数。如果匹配数达到设定值，ADC 硬件将设定 CMPF 比特

头文件

Driver/DrvADC.h

返回值

E_SUCCESS	成功. 比较功能被打开.
E_DRVADC_ARGUMENT	参数错误

DrvADC_Adcmp1Disable

原型

```
void DrvADC_Adcmp1Disable (void);
```

描述

关闭 ADC 转换结果比较功能 1.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_SelfCalEnable

原型

```
void DrvADC_SelfCalEnable (void);
```

描述

使能自校正功能.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_IsCalDone

原型

```
UInt32_t DrvADC_IsCalDone (void);
```

描述

检查是否自校正功能已经完成.

参数

无.

头文件

Driver/DrvADC.h

返回值

TURE	自校正功能已经完成.
FALSE	自校正功能正在进行中.

DrvADC_SelfCalDisable

原型

```
void DrvADC_SelfCalDisable (void);
```

描述

关闭自校正功能.

参数

无.

头文件

Driver/DrvADC.h

返回值

无.

DrvADC_GetVersion

原型

uint32_t DrvAdc_GetVersion (void);

描述

返回驱动版本号.

参数

无.

头文件

Driver/DrvADC.h

返回值

版本号:

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

11. DrvSPI 介绍

11.1. SPI 介绍

串行外设接口(SPI)是一个同步串行数据通讯协议，全双工。设备通讯使用主/从接口，4线，双向模式。NUC1xx 系列有 4 组 SPI 控制器，当从外设收到数据的时候实现串到并的转换，当发送数据到外设的时候实现并到串的转换。每个 SPI 控制器可以驱动 2 个外设。SLAVE 比特 (CNTRL[18])被设定之后，NUC1xx 也能作为从设备工作

当数据传输完成的时候，每个控制器可以产生一个独立的中断信号，写 1 可以清除中断标志。从设备的选中信号激活级别可以是低/高 (SSR[SS_LVL] 比特)，具体设定依靠连接的外设。作为主设备的时候，可以写一个除数到 DIVIDER 寄存器来编程 SPI 时钟频率。如果 SPI_CNTRL[23]中的 VARCLK_EN 比特被设成 1，串行时钟可以被设成两个可编程的频率，除数定义在 DIV 和 DIV2 中。可变频率的选择定义在寄存器 VARCLK 中。

主/从核心包含两个 32 比特的收/发缓冲，支持突发模式，可变长度传输，最大收/发长度是 64 比特。

控制器也支持 2 比特数据模式，由寄存器 SPI_CNTRL[22]控制。如果 TWOB 比特使能，可以收/发 2 比特串行数据出/入串行缓冲。第一个比特从寄存器 SPI_TX0 发送，同时接收第一个比特到寄存器 SPI_RX0 中；第二个比特从寄存器 SPI_TX1 发送，并且接收第二个比特到寄存器 SPI_RX1 中。

11.2. 特性

- 四组SPI控制器
- 支持主/从模式
- 支持1,2 比特串行数据 IN/OUT
- 数据传输长度可配，最大32比特
- 主模式时，输出串行时钟频率可变
- 支持突发模式，一次传输最多可以执行两次收/发
- 支持大端/小端优先数据传输
- 作为主设备的时候，支持2个从设备选择线；作为从设备的时候，支持一个从设备选择线
- 字节休眠模式

12. DrvSPI APIs 说明

12.1. 静态定义

类型	值	描述
E_DRVSPi_PORT	eDRVSPi_PORT0 (0)	SPI 端口 0
	eDRVSPi_PORT1 (1)	SPI 端口 1
	eDRVSPi_PORT2 (2)	SPI 端口 2
	eDRVSPi_PORT3 (3)	SPI 端口 3
E_DRVSPi_MODE	eDRVSPi_MASTER (0)	SPI 主模式
	eDRVSPi_SLAVE (1)	SPI 从模式
	eDRVSPi_JOYSTICK (2)	SPI Joystick 模式
E_DRVSPi_TRANS_TYPE	eDRVSPi_TYPE0 (0)	SPI 传输类型 0
	eDRVSPi_TYPE1 (1)	SPI 传输类型 1
	eDRVSPi_TYPE2 (2)	SPI 传输类型 2
	eDRVSPi_TYPE3 (3)	SPI 传输类型 3
	eDRVSPi_TYPE4 (4)	SPI 传输类型 4
	eDRVSPi_TYPE5 (5)	SPI 传输类型 5
	eDRVSPi_TYPE6 (6)	SPI 传输类型 6
	eDRVSPi_TYPE7 (7)	SPI 传输类型 7
E_DRVSPi_ENDIAN	eDRVSPi_LSB_FIRST(0)	小端优先发送
	eDRVSPi_MSB_FIRST(1)	大端优先发送
E_DRVSPi_SSLTRIG	eDRVSPi_EDGE_TRIGGER (0)	Edge 触发
	eDRVSPi_LEVEL_TRIGGER (1)	Level 触发
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

类型	值	描述
		selected
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. 函数

DrvSPi_Open

原型

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

描述

这个函数用来打开 SPI 功能。配置 SPI 工作在主/从/Joystick 模式、SPI 总线时序和每笔传输的长度。

参数

- eSpiPort [in]**
说明 SPI 端口.
- eMode [in]**
工作模式：主 (eDRVSPi_MASTER) / 从 (eDRVSPi_SLAVE) / Joystick (eDRVSPi_JOYSTiCK)
- eType [in]**
传输类型，也就是总线时序 包括 eDRVSPi_TYPE0~eDRVSPi_TYPE7.
- i32BitLength [in]**

每笔传输的比特长度.

头文件

Driver/DrvSPI.h

返回值

E_SUCCESS	成功.
E_DRVSPI_ERR_INIT	SPI 端口已经被打开了
E_DRVSPIMS_ERR_BIT_LENGTH	比特长度超过范围.
E_DRVSPIMS_ERR_BUSY	SPI 端口正忙.

DrvSPI_Close

原型

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

描述

关闭 SPI 功能并且关闭 SPI 中断.

参数

eSpiPort [in]
说明 SPI 端口

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_Set2BitSerialDataIOMode

原型

```
void DrvSPI_Set2BitSerialDataIOMode (
    E_DRVSPI_PORT eSpiPort,
    BOOL bEnable
);
```

描述

设置 2 比特串行数据 I/O 模式.

参数

eSpiPort [in]

说明 SPI 端口.

bEnable [in]

使能(TRUE)/ 关闭 (FALSE)

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_SetEndian

原型

```
void DrvSPI_SetEndian (
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_ENDIAN eEndian
);
```

描述

配置每笔传输的比特顺序.

参数

eSpiPort [in]

说明 SPI 端口.

eEndian [in]

说明小端优先还是大端优先.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_SetBitLength

原型

```
ERRCODE
```

```
DrvSPI_SetBitLength(
    E_DRVSPI_PORT eSpiPort,
    int32_t i32BitLength
);
```

描述

配置每笔 SPI 传输的比特长度.

参数

eSpiPort [in]

说明 SPI 端口.

i32BitLength [in]

说明比特长度 (1~32 bits).

头文件

Driver/DrvSPI.h

返回值

E_SUCCESS

成功.

E_DRVSPI_ERR_BIT_LENGTH

比特长度超过范围.

DrvSPI_SetByteSleep

原型

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

描述

这个函数可以用来使能/关闭字节休眠功能(Byte Sleep function)

参数

eSpiPort [in]

说明 SPI 端口.

bEnable [in]

使能 (TRUE)/ 禁止 (FALSE)

头文件

Driver/DrvSPI.h

返回值

E_SUCCESS	成功.
E_DRVSPIMS_ERR_BIT_LENGTH	比特长度不是 32 比特.

DrvSPI_SetByteEndian

原型

```

ERRCODE
DrvSPI_SetByteEndian (
    E_DRVSPI_PORT eSpiPort,
    BOOL bEnable
);
    
```

描述

这个函数可以用来设定字节次序功能，只能用于传输长度是 16/24/32 比特时

参数

eSpiPort [in]
说明 SPI 端口.

bEnable [in]
使能 (TRUE) / 关闭(FALSE)

头文件

Driver/DrvSPI.h

返回值

E_SUCCESS	成功.
E_DRVSPIMS_ERR_BIT_LENGTH	比特长度错误，比特长度必须是 16/24/32

DrvSPI_SetTriggerMode

原型

```

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

描述

设定从选择线触发模式

参数

eSpiPort [in]

说明 SPI 端口.

eSSTriggerMode [in]

说明触发模式. (eDRVSPI_EDGE_TRIGGER 或者 eDRVSPI_LEVEL_TRIGGER)

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_SetSlaveSelectActiveLevel

原型

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

描述

设定从选择线的激活级别.

参数

eSpiPort [in]

说明 SPI 端口.

eSSActType [in]

从选择线激活级别.

eDRVSPI_ACTIVE_LOW_FALLING:

电平触发模式下, 从片选信号低激活; 边缘触发模式下, 从片选信号下降沿激活.

eDRVSPI_ACTIVE_HIGH_RISING:

电平触发模式下, 从片选信号高激活; 边缘触发模式下, 从片选信号上升沿激活.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_GetLevelTriggerStatus

原型

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

描述

这个函数可以用来取得电平触发传送的状态，只用于 NUC1xx 工作在从模式

参数

eSpiPort [in]
说明 SPI 端口.

头文件

Driver/DrvSPI.h

返回值

TRUE: 整个传输过程中，每次接收到的笔数和每笔收到的比特数跟 TX_NUM 和 TX_BIT_LEN 的设定匹配.
FALSE: 某次传输的笔数或者接收到的比特数与设定不符

DrvSPI_EnableAutoCS

原型

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

描述

这个函数可以用来使能自动片选功能并且配置片选信号的激活电平。自动片选意味着当 SPI 传输数据的时候，将自动激活片选信号，传输完成的时候将自动取消激活。对一些设备来说，一次激活可以进行多次传输，这时用户应该关闭自动片选功能，改为手动控制。只用于 NUC1xx 是主模式

参数

eSpiPort [in]

说明 SPI 端口.

eSlaveSel [in]

选择从设备片选引脚.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_DisableAutoCS

原型

```
void DrvSPI_DisableAutoCS (
    E_DRV_SPI_PORT eSpiPort
);
```

描述

这个函数可以用来关闭自动片选功能。如果在多次传输过程中片选需要一直保持高/低，用户应该关闭自动片选功能，改为手动控制。只用于 NUC1xx 是主模式

参数

eSpiPort [in]

说明 SPI 端口.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_SetCS

原型

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

描述

激活/配置从设备片选信号. 只用于 NUC1xx 是主模式

参数

eSpiPort [in]

说明 SPI 端口.

eSlaveSel [in]

自动从设备片选模式下, 这个参数将用作传输时的片选.

手动从设备片选模式下, 片选信号将被激活. 可以是 eDRVSPI_NONE, eDRVSPI_SS0, eDRVSPI_SS1 或者 eDRVSPI_SS0_SS1.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_ClrCS

原型

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

描述

从设备片选信号取消激活. 只用于 NUC1xx 是主模式

参数

eSpiPort [in]

说明 SPI 端口.

eSlaveSel [in]

从设备片选.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_Busy

原型

```

BOOL
DrvSPI_Busy(
    E_DRVSPI_PORT eSpiPort
);

```

描述

检查 SPI 端口是否正忙

参数

eSpiPort [in]
说明 SPI 端口.

头文件

Driver/DrvSPI.h

返回值

TURE: SPI 端口正忙.
FALSE: SPI 端口空闲.

DrvSPI_BurstTransfer

原型

```

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

```

描述

配置突发传输模式的相关参数.

参数

eSpiPort [in]
说明 SPI 端口.

i32BurstCnt [in]
说明一次突发传输中的传输笔数。可以是 1 或者 2.

i32Interval [in]

两次连续的传输之间的时钟间隔. 可以是 2~17.

头文件

Driver/DrvSPI.h

返回值

E_SUCCESS	成功.
E_DRVSPIMS_ERR_BURST_CNT	突发传输笔数超过范围.
E_DRVSPIMS_ERR_TRANSMIT_INTERVAL	间隔超过范围.

DrvSPI_SetClock

原型

```
uint32_t
DrvSPI_SetClock(
    E_DRVSPI_PORT eSpiPort,
    uint32_t u32Clock1,
    uint32_t u32Clock2
);
```

描述

配置 SPI 时钟频率.

参数

eSpiPort [in]

说明 SPI 端口.

u32Clock1 [in]

说明 SPI 时钟频率, 单位 Hz。在固定时钟频率模式下, 它是 SPI 的基本时钟频率; 在可变时钟频率模式下, 它是可变时钟 1.

u32Clock2 [in]

说明 SPI 时钟频率, 单位 Hz。它是可变时钟 2

头文件

Driver/DrvSPI.h

Driver/DrvSYS.h

返回值

时钟 1 的除数值。由于硬件的限制, 实际的时钟频率可能与目标有差异

DrvSPI_GetClock1

原型

```
uint32_t
DrvSPI_SetClock1(
    E_DRV_SPI_PORT eSpiPort
);
```

描述

取得 SPI 时钟 1 的频率，单位 Hz.

参数

eSpiPort [in]

说明 SPI 端口.

头文件

Driver/DrvSPI.h
Driver/DrvSYS.h

返回值

SPI 时钟 1 的频率，单位 Hz.

DrvSPI_GetClock2

原型

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

描述

取得 SPI 时钟 2 的频率，单位 Hz.

参数

eSpiPort [in]

说明 SPI 端口.

头文件

Driver/DrvSPI.h
Driver/DrvSYS.h

返回值

SPI 时钟 2 的频率，单位 Hz

DrvSPI_SetVariableClockPattern

原型

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

描述

如果时钟模板 VARCLK 是 '0', SPICLK 的输出频率将根据 DIVIDER 寄存器的值.

如果时钟模板 VARCLK 是 '1', SPICLK 的输出频率将根据 DIVIDER2 寄存器的值.

参数

eSpiPort [in]

说明 SPI 端口.

u32Pattern [in]

说明时钟模板.

头文件

Driver/DrvSPI.h

返回值

无.

DrvSPI_SetVariableClockFunction

原型

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

描述

使能/关闭可变时钟频率功能.

参数

eSpiPort [in]

说明 SPI 端口.

bEnable [in]

使能 (TRUE) / 关闭 (FALSE)

头文件

Driver/DrvSPI.h

返回值

无.

DrvSPI_EnableInt

原型

```
void DrvSPI_EnableInt(
    E_DRVSPi_PORT eSpiPort,
    PFN_DRVSPi_CALLBACK pfnCallback,
    uint32_t u32UserData
);
```

描述

使能指定 SPI 端口的 SPI 中断，并安装中断回调函数.

参数

u16Port [in]

说明 SPI 端口.

pfnCallback [in]

回调函数指针.

u32UserData [in]

传给回调函数的参数.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_DisableInt

原型

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

描述

关闭指定的 SPI 端口的 SPI 中断。

参数

eSpiPort [in]

说明 SPI 端口。

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_SingleRead

原型

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

描述

从 SPI 接收寄存器读数据，并触发下一次 SPI 传输。

参数

eSpiPort [in]

说明 SPI 端口。

pu32Data [out]

储存数据的缓存指针。

头文件

Driver/DrvSPI.h

返回值

TRUE: 存在 pu32Data 中的数据有效.

FALSE: 存在 pu32Data 中的数据无效.

DrvSPI_SingleWrite

原型

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

描述

发送数据到 SPI 总线

参数

eSpiPort [in]

说明 SPI 端口.

pu32Data [in]

要写到 SPI 总线的数据.

头文件

Driver/DrvSPI.h

返回值

TRUE: 存在 pu32Data 中的数据已被发送.

FALSE: SPI 正忙。存在 pu32Data 中的数据未被发送.

DrvSPI_BurstRead

原型

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

描述

从 SPI 接收寄存器读两个 word（4 字节），并触发下一次传输.

参数

eSpiPort [in]

说明 SPI 端口.

pu32Buf [out]

缓存地址，用来储存从 SPI 总线读到的数据.

头文件

Driver/DrvSPI.h

返回值

TRUE: 存在 pu32Data 中的数据有效.

FALSE: 存在 pu32Data 中的数据无效.

DrvSPI_BurstWrite

原型

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

描述

发送两个 word（4 字节）到 SPI 总线.

参数

eSpiPort [in]

说明 SPI 端口.

pu32Buf [in]

要写到 SPI 总线的数据缓存地址

头文件

Driver/DrvSPI.h

返回值

TRUE: 存在 pu32Data 中的数据已经被发送

FALSE: SPI 正忙. 存在 pu32Data 中的数据未被发送.

DrvSPI_DumpRxRegister

原型

```
uint32_t
DrvSPI_DumpRxRegister (
    E_DRV_SPI_PORT eSpiPort,
    uint32_t *pu32Buf,
```

```
uint32_t u32DataCount
);
```

描述

从接收寄存器读数据. 但是不触发下一次数据传输.

参数

eSpiPort [in]

说明 SPI 端口.

pu32Buf [out]

缓存指针, 用来存放从接收寄存器收到的数据.

u32DataCount [in]

要接收的数据笔数, 不能超过 2 笔.

头文件

Driver/DrvSPI.h

返回值

实际从 Rx 寄存器读到的笔数.

DrvSPI_SetTxRegister

原型

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

描述

写数据到发送寄存器, 但是不触发.

参数

eSpiPort [in]

说明 SPI 端口.

pu32Buf [in]

将要写到发送寄存器的缓存指针

u32DataCount [in]

写到发送寄存器的笔数.

头文件

Driver/DrvSPI.h

返回值

实际写到发送寄存器的笔数.

DrvSPI_SetGo

原型

```
void DrvSPI_SetGo (
    E_DRV_SPI_PORT eSpiPort
);
```

描述

设定 GO_BUSY 比特来触发 SPI 数据传输.

参数

eSpiPort [in]

说明 SPI 端口.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_GetJoyStickIntType

原型

```
E_DRV_SPI_JOYSTICK_INT_FLAG
DrvSPI_GetJoyStickIntType (
    E_DRV_SPI_PORT eSpiPort
);
```

描述

取得 JOYSTICK 模式的中断标志.

参数

eSpiPort [in]

说明 SPI 端口.

头文件

Driver/DrvSPI.h

返回值

eDRVSPI_JOYSTICK_DATA_READY: 在缓存中有 8 字节的有效数据.

eDRVSPI_JOYSTICK_CS_ACTIVE: 片选被激活.

eDRVSPI_JOYSTICK_CS_DEACT: 片选未被激活.

eDRVSPI_JOYSTICK_NONE: 无.

DrvSPI_SetJoyStickStatus

原型

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

描述

设定 JoyStick 装态成就绪或者未就绪.

参数

eSpiPort [in]

说明 SPI 端口.

bReady [in]

TRUE: SPI 就绪可以传输数据.

FALSE: SPI 未准备好, 不能发送数据.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_GetJoyStickMode

原型

```
E_DRVSPI_JOYSTICK_RW_MODE
DrvSPI_GetJoyStickMode (
    E_DRVSPI_PORT eSpiPort
);
```

描述

取得 JoyStick 操作模式.

参数

eSpiPort [in]

说明 SPI 端口.

头文件

Driver/DrvSPI.h

返回值

Joystick 操作模式:

eDRVSPI_JOYSTICK_TRANSMIT_MODE: 主设备写数据到从设备.

eDRVSPI_JOYSTICK_RECEIVE_MODE: 主设备从从设备读数据.

DrvSPI_StartPMDA

原型

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

描述

启动/停止 DMA，并且配置 DMA.

参数

eSpiPort [in]

说明 SPI 端口.

eDmaMode [in]

说明 DMA 模式，接收/发送

eEnable [in]

True: 使能 DMA.

False: 关闭 DMA.

头文件

Driver/DrvSPI.h

返回值

无

DrvSPI_GetVersion

原型

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

描述

取得驱动版本号.

参数

无.

头文件

```
Driver/DrvSPI.h
```

返回值

版本号:

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

13. DrvI2C 介绍

13.1. 介绍

I2C 是 Inter-IC (integrated circuit) 总线的首字母缩写。

I2C 对低速、设备间通讯提供很好的支持。它是一个简单、低带宽、短距离协议。I2C 设备最高速度可以达到 1Mbps。因为有内嵌的寻址机制，I2C 可以很容易的将多个设备连接在一起。I2C 设备可以作为主或者从设备。

13.2. 特性

I2C 包含下面的特性:

- 和飞利浦 I2C 标准兼容，支持主和从模式，最高速度可以达到 1Mbps。
- 内嵌一个 14 比特的超时计数器，如果 I2C 总线被挂起并且超时发生，I2C 将发出中断。
- 支持 7 比特寻址模式。
- 支持多地址识别功能。(四个从属地址，支持掩码)

14. DrvI2C APIs 说明

14.1. 函数

DrvI2C_Open

原型

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

描述

打开 I2C 功能，并配置 I2C 总线时钟。

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

clock_Hz [in]

I2C 时钟源频率. 单位 Hz.

baudrate [in]

I2C 传输比特率。单位 bps.

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_Close

原型

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

描述

关闭 I2C 功能。

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_SetClock

原型

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

描述

配置 I2C 总线时钟.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

clock_Hz [in]

I2C 时钟源频率. 单位 Hz.

baudrate [in]

I2C 传输比特率. 单位 bps.

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_GetClock

原型

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

描述

取得 I2C 总线的时钟频率.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

u32clock [in]

I2C 时钟源频率, 单位 Hz.

头文件

Driver/DrvI2C.h

返回值

I2C 总线时钟频率

DrvI2C_SetAddress

原型

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

描述

设定 I2C 从地址, 总共可以设定 4 个从地址. 只用于 NUC1xx 工作在从模式

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

slaveNo [in]

要设定的从地址编号. 可以是 0 ~ 3.

slave_addr [in]

要设定的 7 比特从地址.

GC_Flag [in]

使能/关闭普遍呼叫功能(general call). (1:使能, 0:关闭)

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_SetAddressMask

原型

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

描述

设定 I2C 从地址掩码。

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

slaveNo [in]

从地址编号。可以是 0 ~ 3.

slaveAddrMask [in]

要设定的 7 比特从地址掩码。掩码是 1 的比特，相应的地址比特忽略。

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_GetStatus

原型

```
uint32_t DrvI2C_GetStatus(E_I2C_PORT port);
```

描述

取得 I2C 的状态。共定义了 26 个状态码。

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

I2C 的状态

DrvI2C_WriteData

原型

```
void DrvI2C_WriteData(E_I2C_PORT port, uint8_t u8data);
```

描述

写数据到发送寄存器.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

u8data [in]

要发送的数据.

头文件

Driver/DrvI2C.h

返回值

无

DrvI2C_ReadData

原型

```
uint8_t DrvI2C_ReadData(E_I2C_PORT port);
```

描述

从 I2C 总线读一个字节的数据.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

一个字节的数据

DrvI2C_Ctrl

原型

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

描述

设定 I2C 控制比特, 包括控制寄存器中的 STA, STO, AA, SL.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

start [in]

STA 比特. (1:拉高, 0: 拉低)

stop [in]

STO 比特. (1: 拉高, 0: 拉低)

intFlag [in]

是否清除 SI (状态中断标志) 比特. (1:清除, 0:不起作用)

ack [in]

是否使能 AA 比特. (1:使能, 0:关闭)

头文件

Driver/DrvI2C.h

返回值

无

DrvI2C_GetIntFlag

原型

```
uint8_t DrvI2C_GetIntFlag(E_I2C_PORT port);
```

描述

取得 I2C 中断状态, 就是 SI 的值.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

中断状态 (1 或者 0)

DrvI2C_ClearIntFlag

原型

```
void DrvI2C_ClearIntFlag(E_I2C_PORT port);
```

描述

清除 I2C 中断标志，就是将 SI 清 0.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

无

DrvI2C_EnableInt

原型

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

描述

使能 I2C 中断和相应的 NVIC(m0 core)比特.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_DisableInt

原型

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

描述

关闭 I2C 中断和相应的 NVIC 比特.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_InstallCallBack

原型

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

描述

安装 I2C 中断回调函数.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

Type [in]

回调函数有四种类型. (I2CFUNC / ARBITLOSS / BUSERROR / TIMEOUT)

I2CFUNC: 正常的 I2C 中断

ARBITLOSS: 主模式下仲裁丢失。状态码 0x38.

BUSERROR: 总线错误。状态码 0x00.

TIMEOUT: 14 比特超时计数器溢出.

callbackfn [in]

回调函数指针

头文件

Driver/DrvI2C.h

返回值

0 成功

<0 失败

DrvI2C_UninstallCallBack

原型

```
int32_t DrvI2C_UninstallCallBack(E_I2C_PORT port, E_I2C_CALLBACK_TYPE Type);
```

描述

卸载 I2C 中断回调函数.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

Type [in]

回调函数有四种类型. (I2CFUNC / ARBITLOSS / BUSERROR / TIMEOUT)

I2CFUNC: 正常的 I2C 中断

ARBITLOSS: 主模式下仲裁丢失。状态码 0x38.

BUSERROR: 总线错误。状态码 0x00.

TIMEOUT: 14 比特超时计数器溢出

头文件

Driver/DrvI2C.h

返回值

0 成功

<0 失败

DrvI2C_EnableTimeoutCount

原型

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

描述

使能/关闭 14 比特超时计数器.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

i32enable [in]

使能/关闭 14 比特超时计数器. (1:使能, 0:关闭)

u8div4 [in]

使能 DIV4 功能。如果超时计数器使能的话，计数器输入时钟将被除以 4。超时时间将延长 4 倍

头文件

Driver/DrvI2C.h

返回值

0 成功

DrvI2C_ClearTimeoutFlag

原型

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

描述

清除 I2C 超时中断标志.

参数

port [in]

说明 I2C 端口. (I2C_PORT0 / I2C_PORT1)

头文件

Driver/DrvI2C.h

返回值

无

15. DrvRTC 介绍

15.1. RTC 控制器介绍

NUC1xx 包含一个内嵌的 RTC，当系统断电以后，实时时钟(RTC) 可以由单独的电源供电。RTC 使用一个 32.768 KHz 的外部 crystal。RTC 可以周期性产生中断，中断周期可以是 0.25/ 0.5/ 1/ 2/ 4/ 8 秒。有一个 RTC 溢出计数器，可以由软件调节

15.2. RTC 特性

- 有一个时钟计数器，用户可以用来查看时间。
- 唤醒系统时，如果电池电压很低，上电超时功能可以关掉系统，避免系统挂起。
- 支持时钟 tick 中断
- 支持唤醒功能

16. DrvRTC APIs 说明

16.1. 静态定义

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

名字	值	描述
DRVRTC_CLOCK_12	0	12 小时模式
DRVRTC_CLOCK_24	1	24 小时模式
DRVRTC_AM	1	上午
DRVRTC_PM	2	下午
DRVRTC_LEAP_YEAR	1	闰年
DRVRTC_TICK_1_SEC	0	每秒 1 tick
DRVRTC_TICK_1_2_SEC	1	每秒 2 tick
DRVRTC_TICK_1_4_SEC	2	每秒 4 tick
DRVRTC_TICK_1_8_SEC	3	每秒 8 tick
DRVRTC_TICK_1_16_SEC	4	每秒 16 tick
DRVRTC_TICK_1_32_SEC	5	每秒 32 tick
DRVRTC_TICK_1_64_SEC	6	每秒 64 tick
DRVRTC_TICK_1_128_SEC	7	每秒 128 tick
DRVRTC_SUNDAY	0	星期天
DRVRTC_MONDAY	1	星期一
DRVRTC_TUESDAY	2	星期二
DRVRTC_WEDNESDAY	3	星期三
DRVRTC_THURSDAY	4	星期四
DRVRTC_FRIDAY	5	星期五
DRVRTC_SATURDAY	6	星期六
DRVRTC_ALARM_INT	0x01	警报中断
DRVRTC_TICK_INT	0x02	Tick 中断
DRVRTC_ALL_INT	0x03	所有中断
DRVRTC_IOC_IDENTIFY_LEAP_YEAR	0	标识闰年命令
DRVRTC_IOC_SET_TICK_MODE	1	设定 tick 模式命令
DRVRTC_IOC_GET_TICK	2	取得 tick 数命令

DRVRTC_IOC_RESTORE_TICK	3	恢复 tick 命令
DRVRTC_IOC_ENABLE_INT	4	使能中断命令
DRVRTC_IOC_DISABLE_INT	5	关闭中断命令
DRVRTC_IOC_SET_CURRENT_TIME	6	设定当前时间命令
DRVRTC_IOC_SET_ALAMRM_TIME	7	设定警报时间命令
DRVRTC_IOC_SET_FREQUENCY	8	设定频率命令
DRVRTC_CURRENT_TIME	0	当前时间
DRVRTC_ALARM_TIME	1	警报时间

16.2. 函数

DrvRTC_SetFrequencyCompenation

原型

```
int32_t
DrvRTC_SetFrequencyCompenation (
    float fnumber;
);
```

描述

设定频率补偿值

参数

fnumber [in]
频率补偿值.

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_FCR_VALUE: 补偿值错误

DrvRTC_WriteEnable

原型

```
int32_t
```



```
DrvRTC_WriteEnable (void);
```

描述

寄存器 AER 的比特 15~0 作为 RTC 寄存器读/写的密码。可以用来避免关机时信号干扰。上电以后写 0xA965 到 AER 寄存器之后等待 512 个 RTC 时钟,就可以访问 RTC 寄存器了。

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_FAILED: 失败.

DrvRTC_Init

原型

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

描述

初始化 RTC

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_EIO: 初始化 RTC 失败.

DrvRTC_Open

原型

```
int32_t  
DrvRTC_Open (  
    S_DRVRTC_TIME_DATA_T *sPt  
);
```

描述

设定当前时间和日期,并设定其属性.

参数

***sPt [in]**

说明时间/日期属性和当前的时间/日期

u8cClockDisplay : DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24

u8cAmPm : DRVRTC_AM / DRVRTC_PM

u32cSecond : 秒

u32cMinute : 分 Minute value

u32cHour : 小时

u32cDayOfWeek : 星期

u32cDay : 日

u32cMonth : 月

u32Year : 年

pfnAlarmCallBack : 警报回调函数指针

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_EIO : 初始化 RTC 失败.

DrvRTC_Read

原型

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

描述

从 RTC 读取当前时间/日期或者警报的时间/日期

参数

eTime [in]

说明读取当前时间还是警报时间.

DRVRTC_CURRENT_TIME: 当前时间

DRVRTC_ALARM_TIME: 警报时间

***sPt [in]**

存放时间/日期的结构指针. 包括

u8cClockDisplay : DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24

u8cAmPm : DRVRTC_AM / DRVRTC_PM

u32cSecond : 秒

u32cMinute : 分

u32cHour : 小时

u32cDayOfWeek : 星期

u32cDay : 日

u32cMonth : 月

u32Year : 年

pfnAlarmCallBack : 警报回调函数指针

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_EIO: 读 RTC 失败.

DrvRTC_Write

原型

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

描述

写当前时间/日期或者警报时间/日期到 RTC 中

参数

eTime [in]

说明写当前时间还是警报时间.

DRVRTC_CURRENT_TIME: 当前时间

DRVRTC_ALARM_TIME: 警报时间

*sPt [in]

存放时间/日期的结构指针. 包括

u8cClockDisplay : DRVRTC_CLOCK_12 / DRVRTC_CLOCK_24

u8cAmPm : DRVRTC_AM / DRVRTC_PM

u32cSecond : 秒

u32cMinute : 分

u32cHour : 小时

u32cDayOfWeek : 星期

u32cDay : 日

u32cMonth : 月

u32Year : 年

pfnAlarmCallBack : 警报回调函数指针

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_EIO : 写 RTC 失败.

DrvRTC_Ioctl

原型

```
int32_t
DrvRTC_Ioctl (
    INT32          i32Num
    E_DRVRTC_CMD   eCmd,
    UINT32          u32Arg0,
    FLOAT           fArg1
);
```

描述

支持一些 RTC 控制.

参数

i32Num [in]

预留

eCmd [in]

命令

DRVRTC_IOC_IDENTIFY_LEAP_YEAR: 检查是否是闰年

DRVRTC_IOC_SET_TICK_MODE: 设置 Tick 模式

DRVRTC_IOC_GET_TICK: 取得 tick 计数

DRVRTC_IOC_RESTORE_TICK: 恢复 tick 计数

DRVRTC_IOC_ENABLE_INT: 使能中断

DRVRTC_IOC_DISABLE_INT: 关闭中断

DRVRTC_IOC_SET_CURRENT_TIME: 设定当前时间

DRVRTC_IOC_SET_ALAMRM_TIME: 设定警报时间

DRVRTC_IOC_SET_FREQUENCY: 设定频率补偿值

u32Arg0 [in]

1. 存放返回的闰年标志 (DRVRTC_IOC_IDENTIFY_LEAP_YEAR)
2. 存放 tick 模式数据 (DRVRTC_IOC_SET_TICK_MODE)
3. 存放返回的 tick 计数(DRVRTC_IOC_GET_TICK)
4. 存放使能的中断类型 (DRVRTC_IOC_ENABLE_INT)
5. 存放关闭的中断类型 (DRVRTC_IOC_DISABLE_INT)
6. 存放频率补偿值 (DRVRTC_IOC_SET_FREQUENCY)

fArg1 [in]

预留.

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

E_DRVRTC_ERR_ENOTTY 命令不支持, 或者参数错误.

E_DRVRTC_ERR_ENODEV RTC 端口说明错误, i32Num 只能是 0

DrvRTC_Close

原型

int32_t

DrvRTC_Close (VOID);

描述

关闭 RTC 中断.

头文件

Driver/DrvRTC.h

返回值

E_SUCCESS: 成功

DrvRTC_GetVersion

原型

int32_t
DrvRTC_GetVersion (void);

描述

取得驱动版本号.

头文件

Driver/DrvRTC.h

返回值

版本号 :

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

17. DrvCAN 介绍

17.1. CAN 介绍

控制器局域网(CAN) 是一个串行通讯协议, 支持多主设备并且可以有效的支持分布式实时控制, 并且有很高的保密性. 在 CAN 系统中, 一个节点(Node)不使用系统配置的任何信息。不用请求其它节点软件/硬件任何改变, 节点就可以加入 CAN 网络.

17.2. CAN 特性

CAN 处理器包含下面的特性:

- 与 CAN 2.0B 协议兼容
- 与 AMBA APB 总线接口兼容
- 多主设备节点
- 支持 11 比特标识符也支持 29 比特标识符
- 最高比特率可达 1Mbit/s
- NRZ 比特编码
- 错误侦测: 比特错误, 填充错误, 格式错误, 15 比特 CRC 校验错误, 和应答错误
- 只侦听模式(没有应答, 不激活错误标志)
- 报文验收滤波扩展(4 字节标识符, 4 字节掩码)
- 每个 CAN 总线错误都有错误中断
- 扩展接收缓存(8 字节缓冲区)
- 唤醒功能

18. DrvCAN APIs 说明

18.1. 函数

DrvCAN_Open

原型

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

描述

这个函数可以用来打开并初始化 CAN.

参数

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
或者用户自己的配置值

头文件

Driver/DrvCAN.h

返回值

E_SUCEESS

DrvCAN_DisableInt

原型

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


描述

这个函数可以用来关闭 CAN 中断并卸载中断回调函数.

参数

port [in]

CAN_PORT0 / CAN_PORT1

u32InterruptFlag [in]

INT_BEI/INT_ALI/INT_WUI/INT_TI/INT_RI.

头文件

Driver/DrvCAN.h

返回值

E_SUCCESS

DrvCAN_EnableInt

原型

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

描述

这个函数可以用来使能 CAN 中断并且安装中断回调函数.

参数

port [in]

CAN 通道: CAN_PORT0 / CAN_PORT1.

u32InterruptFlag [in]

中断标志 INT_BEI/INT_ALI/INT_WUI/INT_TI/INT_RI.

pfncallback [in]

回调函数指针.

头文件

Driver/DrvCAN.h

返回值

无

DrvCAN_GetErrorStatus

原型

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

描述

这个函数可以用来取得 CAN 错误状态

参数

port [in]

CAN 通道: CAN_PORT0 / CAN_PORT1.

头文件

Driver/DrvCAN.h

返回值

E_SUCCESS 成功

DrvCAN_ReadMsg

原型

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

描述

这个函数可以用来取得 CAN 的接收信息.

参数

无

头文件

Driver/DrvCAN.h

返回值

CAN structure

DrvCAN_SetAcceptanceFilter

原型

```
int32_t DrvCAN_SetAcceptanceFilter (
```

```
CAN_PORT      port,
int32_t        id_Filter
);
```

描述

这个函数可以用来设定接收标识符过滤。

参数

port [in]

CAN 端口 CAN_PORT0 / CAN_PORT1

id_Filter [in]

写到特定标识符过滤器中的数据

头文件

Driver/DrvCAN.h

返回值

E_SUCCESS 成功.

DrvCAN_SetMaskFilter

原型

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

描述

这个函数可以用来设定标识符过滤掩码。

参数

port [in]

端口: CAN_PORT0 / CAN_PORT1

id_Filter [in]

写到特定标识符过滤掩码中的数据

头文件

Driver/DrvCAN.h

返回值

E_SUCCESS 成功

DrvCAN_WaitReady

原型

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

描述

这个函数可以用来检查 CAN 总线是否繁忙

参数

port [in]

CAN 端口 CAN_PORT0 / CAN_PORT1

头文件

Driver/DrvCAN.h

返回值

无

DrvCAN_WriteMsg

原型

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

描述

这个函数可以用来设定 CAN 信息并送到 CAN 总线

参数

port [in]

CAN 端口 CAN_PORT0 / CAN_PORT1

Msg [in]

说明 CAN 的特性. 包括

id: 18 比特或者 29 比特标识符

u32cData[2]: 发送的数据域

u8cLen: 数据域长度, 单位是字节

u8cFormat: 标准或者扩展标识符

u8cType: FRAME 或者 REMOTE FRAME

u8OverLoad: 关闭或者使能 overload

头文件

Driver/DrvCAN.h

返回值
无

DrvCAN_GetVersion

原型
iint32_t
DrvCAN_GetVersion (void);

描述
返回驱动当前版本号.

头文件
Driver/DrvCAN.h

返回值
版本号 :

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

19. DrvPWM 介绍

19.1. PWM 介绍

NUC1xx 提供 2 组 PWM 发生器，每组可以配置成 4 个独立的 PWM 输出 PWM0~PWM3 或者 2 对互补的 PWM (PWM0,PWM1) 和 (PWM2,PWM3)。每组 PWM 有一个 8 比特的预分频器，一个时钟分频器可以提供 5 种时钟源(1, 1/2, 1/4, 1/8, 1/16)，两个 PWM 定时器包括两个时钟选择器，两个 16 比特递减计数器，两个 16 比特比较器，逆变器，一个死区发生器。它们都由 PWM 时钟源驱动。有四种时钟源：12 MHz crystal 时钟, 32 KHz crystal 时钟, HCLK, 和内部 22 MHz 时钟.每个 PWM 定时器从时钟分频器接收自己的时钟信号。每个时钟分频器的时钟源来自 8 比特预分频器。每个通道的 16 比特计数器接收来自时钟选择器的时钟信号作为一个时钟周期。16 比特比较器比较计数器和极限值寄存器中的值来控制 PWM 的占空比

为了避免 PWM 在不稳定的状态下驱动输出引脚，16 比特的计数器和 16 比特的比较器使用双缓存模式。用户可以随意写数据到计数器缓存寄存器和比较器缓存寄存器，不用考虑短时脉冲波型干扰。

当 16 比特递减计数器减到 0 时，中断产生通知 CPU 时间到。当计数器减到 0 时，如果计数器被设成触发(toggle)模式，它的值会被重新自动加载并且自动开始下一轮循环。用户也可以将计数器设成单次(one-shot)模式，这样减到 0 的时候，计数器将停止计数并且产生中断。.

20. DrvPWM APIs 说明

20.1. 静态定义

名称	值	描述
DRVPWM_TIMER0	0x00	PWM 定时器 0
DRVPWM_TIMER1	0x01	PWM 定时器 1
DRVPWM_TIMER2	0x02	PWM 定时器 2
DRVPWM_TIMER3	0x03	PWM 定时器 3
DRVPWM_CAP0	0x10	PWM 捕获器 0
DRVPWM_CAP1	0x11	PWM 捕获器 1
DRVPWM_CAP2	0x12	PWM 捕获器 2
DRVPWM_CAP3	0x13	PWM 捕获器 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	输入时钟除以 1
DRVPWM_CLOCK_DIV_2	0	输入时钟除以 2
DRVPWM_CLOCK_DIV_4	1	输入时钟除以 4
DRVPWM_CLOCK_DIV_8	2	输入时钟除以 8
DRVPWM_CLOCK_DIV_16	3	输入时钟除以 16
DRVPWM_TOGGLE_MODE	1	PWM Timer Toggle mode
DRVPWM_ONE_SHOT_MODE	0	PWM Timer One-shot mode

20.2. 函数

DrvPWM_IsTimerEnabled

原型

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

描述

这个函数可以用来取得 PWM 定时器的状态

参数

u8Timer [in]

指定定时器.

DRVPWM_TIMER0: PWM 定时器 0.

DRVPWM_TIMER1: PWM 定时器 1.

DRVPWM_TIMER2: PWM 定时器 2.

DRVPWM_TIMER3: PWM 定时器 3.

头文件

Driver/DrvPWM.h

返回值

1: 指定的定时器是使能的.

0: 指定的定时器没有使能.

DrvPWM_SetTimerCounter

原型

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

描述

这个函数可以用来设定 PWM 定时器的计数值.

参数

u8Timer [in]

指定定时器.

DRVPWM_TIMER0: PWM 定时器 0.

DRVPWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

u16Counter [in]

定时器的计数值. (0~65535)

头文件

Driver/DrvPWM.h

返回值

无

Note

如果计数值是 0, 定时器将停止.

DrvPWM_GetTimerCounter

原型

```
uint32_t DrvPWM_GetTimerCounter(uint8_t u8Timer);
```

描述

这个函数可以用来取得 PWM 定时器的计数值

参数

u8Timer [in]

指定定时器.

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

头文件

Driver/DrvPWM.h

返回值

定时器的计数值.

DrvPWM_EnableInt

原型

```
void DrvPWM_EnableInt(uint8_t u8Timer, uint8_t u8Int, PFN_DRV_PWM_CALLBACK  
pfncallback);
```

描述

这个函数可以用来使能 PWM 定时器/捕获器的中断并且安装中断回调函数

参数

u8Timer [in]

指定定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV_PWM_CAP0: PWM 捕获器 0.

DRV_PWM_CAP1: PWM 捕获器 1.

DRV_PWM_CAP2: PWM 捕获器 2.

DRV_PWM_CAP3: PWM 捕获器 3.

u8Int [in]

说明捕获器中断类型(只在 PWM 运行在捕获功能时这个参数才有效)

DRV_PWM_CAP_RISING_INT: 捕获上升沿时中断.

DRV_PWM_CAP_FALLING_INT: 捕获下降沿时中断.

DRV_PWM_CAP_ALL_INT: 上升/下降沿都发生中断.

pfncallback [in]

中断回调函数指针.

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_DisableInt

原型

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

描述

这个函数可以用来关闭 PWM 定时器/捕获器中断

参数

u8Timer [in]

指定定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV_PWM_CAP0: PWM 捕获器 0.

DRV_PWM_CAP1: PWM 捕获器 1.

DRV_PWM_CAP2: PWM 捕获器 2.

DRV_PWM_CAP3: PWM 捕获器 3.

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_ClearInt

原型

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

描述

这个函数可以用来清除 PWM 定时器/捕获器中断标志

参数

u8Timer [in]

指定定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV_PWM_CAP0: PWM 捕获器 0.

DRV_PWM_CAP1: PWM 捕获器 1.

DRV_PWM_CAP2: PWM 捕获器 2.

DRV_PWM_CAP3: PWM 捕获器 3.

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_GetIntFlag

原型

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

描述

这个函数可以用来取得 PWM 定时器/捕获器中断标志

参数

u8Timer [in]

指定定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV_PWM_CAP0: PWM 捕获器 0.

DRV_PWM_CAP1: PWM 捕获器 1.

DRV_PWM_CAP2: PWM 捕获器 2.

DRV_PWM_CAP3: PWM 捕获器 3.

头文件

Driver/DrvPWM.h

返回值

1: 指定的中断已经发生.

0: 指定的中断没有发生.

DrvPWM_GetRisingCounter

原型

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

描述

这个函数可以用来取得当有上升转变时，锁存的计数值。

参数

u8Capture [in]

说明捕获器。

DRVPWM_CAP0: PWM 捕获器 0.

DRVPWM_CAP1: PWM 捕获器 1.

DRVPWM_CAP2: PWM 捕获器 2.

DRVPWM_CAP3: PWM 捕获器 3.

头文件

Driver/DrvPWM.h

返回值

锁存的计数值。

DrvPWM_GetFallingCounter

原型

```
uint16_t DrvPWM_GetFallingCounter(uint8_t u8Capture);
```

描述

这个函数可以用来取得当有下降转变时，锁存的计数值。

参数

u8Capture [in]

说明捕获器。

DRVPWM_CAP0: PWM 捕获器 0.

DRVPWM_CAP1: PWM 捕获器 1.

DRVPWM_CAP2: PWM 捕获器 2.

DRVPWM_CAP3: PWM 捕获器 3.

头文件

Driver/DrvPWM.h

返回值

锁存的计数值。

DrvPWM_GetCaptureIntStatus

原型

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

描述

这个函数可以用来取得捕获器中断状态，也就是检查是否有发生上升 / 下降变换

参数

u8Capture [in]

说明捕获器.

DRVPWM_CAP0: PWM 捕获器 0.

DRVPWM_CAP1: PWM 捕获器 1.

DRVPWM_CAP2: PWM 捕获器 2.

DRVPWM_CAP3: PWM 捕获器 3.

u8IntType [in]

说明要检查的中断类型.

DRVPWM_CAP_RISING_FLAG: 上升沿中断标志

DRVPWM_CAP_FALLING_FLAG: 下降沿中断标志

头文件

Driver/DrvPWM.h

返回值

TRUE: 中断发生

FALSE: 中断没有发生.

DrvPWM_ClearCaptureIntStatus

原型

```
void DrvPWM_ClearCaptureIntStatus(uint8_t u8Capture, uint8_t u8IntType);
```

描述

清除上升/下降中断标志

参数

u8Capture [in]

说明捕获器.

DRVPWM_CAP0: PWM 捕获器 0.

DRV_PWM_CAP1: PWM 捕获器 1.

DRV_PWM_CAP2: PWM 捕获器 2.

DRV_PWM_CAP3: PWM 捕获器 3.

u8IntType [in]

说明中断类型.

DRV_PWM_CAP_RISING_FLAG: 上升沿中断标志.

DRV_PWM_CAP_FALLING_FLAG: 下降沿中断标志.

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_Open

原型

void DrvPWM_Open(void);

描述

打开 PWM 时钟并且复位 PWM

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_Close

原型

void DrvPWM_Close(void);

描述

关闭 PWM 功能包括时钟和中断

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_EnableDeadZone

原型

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

描述

这个函数可以用来配置死区长度并且使能/关闭死区功能。

参数

u8Timer [in]

说明定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

u8Length [in]

说明死区长度: 0~255.

i32EnableDeadZone [in]

使能 DeadZone (1) / 关闭 DeadZone (0)

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_Enable

原型

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

描述

这个函数可以用来使能 PWM 定时器/捕获器功能

参数

u8Timer [in]

说明定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV PWM_TIMER2: PWM 定时器 2.

DRV PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV PWM_CAP0: PWM 捕获器 0.

DRV PWM_CAP1: PWM 捕获器 1.

DRV PWM_CAP2: PWM 捕获器 2.

DRV PWM_CAP3: PWM 捕获器 3.

i32Enable [in]

Enable (1) / Disable (0)

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_SetTimerClk

原型

uint32_t DrvPWM_SetTimerClk(uint8_t u8Timer, S_DRV PWM_TIME_DATA_T *sPt);

描述

这个函数可以用来配置频率/脉冲/模式/逆转功能

参数

u8Timer [in]

说明定时器

DRV PWM_TIMER0: PWM 定时器 0.

DRV PWM_TIMER1: PWM 定时器 1.

DRV PWM_TIMER2: PWM 定时器 2.

DRV PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV PWM_CAP0: PWM 捕获器 0.

DRV PWM_CAP1: PWM 捕获器 1.

DRV PWM_CAP2: PWM 捕获器 2.

DRV PWM_CAP3: PWM 捕获器 3.

***sPt [in]**

包含下面的参数

u8Frequency: 定时器/捕获器频率

u8HighPulseRatio: 高脉冲比率

u8Mode: DRV_PWM_ONE_SHOT_MODE / DRV_PWM_TOGGLE_MODE

bInverter: 逆转使能 (1) / 逆转关闭 (0)

u8ClockSelector: 时钟选择器

DRV_PWM_CLOCK_DIV_1:

DRV_PWM_CLOCK_DIV_2:

DRV_PWM_CLOCK_DIV_4:

DRV_PWM_CLOCK_DIV_8:

DRV_PWM_CLOCK_DIV_16:

(只有当 u8Frequency = 0 时这个参数才起作用)

u8PreScale: 预分频 (2 ~ 256)

(只有当 u8Frequency = 0 时这个参数才起作用)

u32Duty: Pulse duty

(只有当 u8Frequency = 0 或者 u8Timer =

DRV_PWM_CAP0/DRV_PWM_CAP1/DRV_PWM_CAP2/DRV_PWM_CAP3 时这个参数才起作用)

头文件

Driver/DrvPWM.h

返回值

实际的频率.

Note

1. 当用户设定一个非 0 频率值的时候, 这个函数将自动设定频率属性
2. 当用户设定的频率值为 0 的时候, 用户也可以自己设定频率属性(时钟选择器/预分频/占空比).
3. 对于捕获器功能, 这个函数可以设定合适的频率属性 (时钟选择器/预分频), 对于占空比用户需要自己设定

DrvPWM_SetTimerIO

原型

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

描述

这个函数可以用来使能/关闭 PWM 定时器/捕获器功能

参数

u8Timer [in]

说明定时器

DRV_PWM_TIMER0: PWM 定时器 0.

DRV_PWM_TIMER1: PWM 定时器 1.

DRV_PWM_TIMER2: PWM 定时器 2.

DRV_PWM_TIMER3: PWM 定时器 3.

或者捕获器.

DRV_PWM_CAP0: PWM 捕获器 0.

DRV_PWM_CAP1: PWM 捕获器 1.

DRV_PWM_CAP2: PWM 捕获器 2.

DRV_PWM_CAP3: PWM 捕获器 3.

i32Enable [in]

使能 (1) / 关闭(0)

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_SelectClockSource

原型

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

描述

这个函数可以用来选择 PWM0/PWM1 和 PWM2/PWM3 的时钟源.

参数

u8Timer [in]

说明定时器

DRV_PWM_TIMER0/DRV_PWM_TIMER1: PWM 定时器 0 or PWM 定时器 1.

DRV_PWM_TIMER2/DRV_PWM_TIMER3: PWM 定时器 2 or PWM 定时器 3.

u8ClockSourceSelector [in]

DRV_PWM_EXT_12M/DRV_PWM_EXT_32K/DRV_PWM_HCLK/DRV_PWM_INTERNAL_22M

DRV_PWM_EXT_12M: 外部 12 MHz crystal 时钟

DRV_PWM_EXT_32K: 外部 32 KHz crystal 时钟

DRV_PWM_HCLK: HCLK

DRV_PWM_INTERNAL_22M: 内部 22 MHz crystal 时钟

头文件

Driver/DrvPWM.h

返回值

无

DrvPWM_GetVersion

原型

iint32_t

DrvPWM_GetVersion (void);

描述

返回驱动当前版本号.

头文件

Driver/DrvCAN.h

返回值

版本号 :

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

21. DrvPS2 介绍

21.1. PS2 介绍

PS/2 设备控制器为 PS2 通讯提供基本的时序。设备和主机之间的所有通讯都通过 CLK 和 DATA 引脚管理。接收到一个请求之后，设备控制器产生 CLK 信号，但是主机有最高控制权。从主机发送到设备的数据在上升沿读取，从设备发送到主机的数据在上升沿以后改变。一个 16 字节的发送缓冲用来减少 CPU 的干预，但是没有接收缓冲。连续发送的时候，软件可以选择 1-16 个字节的发送缓冲深度。

因为 PS2 设备控制器非常简单，为了速度考虑，我们推荐尽量使用宏定义。因为没有接收缓冲，所以 DrvPS2_Read 只读一个字节；但是 DrvPS2_Write 可以写任意长度的字节到主机

缺省 PS2 中断处理函数已经实现，就是 PS2_IRQHandler。用户可以通过函数 DrvPS2_EnableInt 安装中断回调函数，通过函数 DrvPS2_DisableInt 卸载掉

21.2. PS2 特性

PS2 设备控制器包含下面的特性：

- APB 接口兼容.
- 主机通讯抑制并请求发送检测.
- 接收帧错误检测
- 可编程 1 到 16 字节发送缓冲，以降低 CPU 干涉，但是没有接收缓冲
- 接收支持双缓冲
- 支持软件重置总线

22. DrvSP2 APIs 说明

22.1. 宏

DRVPS2_OVERRIDE

原型

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

描述

这个宏可以用来使能/关闭软件控制 DATA/CLK 线的能力.

参数

state **[in]**

说明是否使能软件重置. 1 意味着使能软件控制 PS2 CLK/DATA 引脚状态; 0 意味着关闭软件重置功能.

头文件

Driver/DrvPS2.h

返回值

无.

DRVPS2_PS2CLK

原型

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

描述

如果软件重置功能被使能, 这个宏可以用来迫使 PS2CLK 高/低, 而不考虑设备控制器内部的状态

参数

state **[in]**

指示 PS2CLK 线高/低

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_PS2DATA

原型

void DRVPS2_PS2DATA(bool state);

描述

如果软件重置功能被使能，这个宏可以用来迫使 PS2DATA 高/低，而不考虑设备控制器内部的状态.

参数

u16Port [in]

指示 PS2DATA 线高/低

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_CLRFIFO

原型

void DRVPS2_CLRFIFO();

描述

这个宏可以用来清除发送缓冲.

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_ACKNOTALWAYS

原型

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

描述

这个宏可以用来使能/关闭总是应答功能。state=1 时，如果校验错误或者停止位没有收到，在第 12 个时钟的时候，应答比特将不会被发送给主机；反之，在第 12 个时钟的时候，总是发送应答比特到主机

参数

state [in]

使能/关闭总是应答功能

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_RXINTENABLE

原型

```
void DRVPS2_RXINTENABLE();
```

描述

这个宏可以用来使能接收中断。当主机发送数据给设备的时候，应答比特被发送给主机之后，接收中断将发生

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_RXINTDISABLE

原型


```
void DRVPS2_RXINTDISABLE();
```

描述

这个宏可以用来关闭接收中断

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_TXINTENABLE

原型

```
void DRVPS2_TXINTENABLE();
```

描述

这个宏可以用来使能发送中断。当 STOP 比特被发送时，发送中断将发生.

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_TXINTDISABLE

原型

```
void DRVPS2_TXINTDISABLE ();
```

描述

这个宏可以用来关闭发送中断

参数

无

头文件

Driver/DrvPS2.h

返回值

无.

DRVPS2_PS2ENABLE

原型

void RVPS2_PS2ENABLE ();

描述

这个宏可以用来使能 PS2 设备控制器

参数

无

头文件

Driver/DrvPS2.h

返回值

无.

DRVPS2_PS2DISABLE

原型

void RVPS2_PS2DISABLE ();

描述

这个宏可以用来关闭 PS2 设备控制器.

参数

无

头文件

Driver/DrvPS2.h

返回值

无.

DRVPS2_TXFIFO

原型

```
void DRVPS2_TXFIFO();
```

描述

这个宏可以用来设定发送缓冲深度。范围[0,15]

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_SWOVERRIDE

原型

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

描述

如果软件重置功能被使能，这个宏可以用来设定 PS2DATA 和 PS2CLK 线的状态。它等于下面的宏：

```
DRVPS2_PS2DATA(data);
```

```
DRVPS2_PS2CLK(clk);
```

```
DRVPS2_OVERRIDE(1);
```

参数

data [in]

说明 PS2DATA 线高/低

clk [in]

说明 PS2CLK 线高/低

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_INTCLR

原型

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

描述

这个宏可以用来清除中断标志.

参数

intclr [in]

清除接收/发送中断. Intclr=0x1: 清除接收中断; Intclr=0x2 清除发送中断; Intclr=0x3 清除接收和发送中断

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_RXDATA

原型

```
uint8_t DRVPS2_RXDATA();
```

描述

这个宏可以用来从接收寄存器读一个字节.

参数

无

头文件

Driver/ DrvPS2.h

返回值

从主机收到的一个字节.

DRVPS2_TXDATAWAIT

原型

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

描述

这个宏可用来发送数据，它将等待发送缓冲区空，然后设定发送缓冲区深度，然后将 data 填充到发送缓冲寄存器 0(共有四个发送寄存器 0~3)，也就是发送缓冲区的 0-3。如果总线空闲，数据将马上被发送。参数 len 的范围 [0, 15]

当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1

参数

data [in]

要发送的数据

len [in]

要发送的数据长度。单位是字节。范围[0, 15]

头文件

Driver/ DrvPS2.h

返回值

无。

DRVPS2_TXDATA

原型

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

描述

这个宏可用来发送数据，它将设定发送缓冲区深度，然后将 data 填充到发送缓冲寄存器 0(共有四个发送寄存器 0~3)，也就是发送缓冲区的 0-3。与 DRVPS2_TXDATAWAIT 的区别只在于，不等待发送缓冲区空。如果总线空闲，数据将马上被发送。参数 len 的范围 [0, 15]

当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1

参数

data [in]

要发送的数据

len [in]

要发送的数据长度，单位字节，范围 [0, 15]

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_TXDATA0

原型

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

描述

这个宏可以用来填充发送缓冲寄存器 0(共有四个发送寄存器 0~3)，也就是发送缓冲区的 0-3，但是不等待发送缓冲空，也不设定发送缓冲区深度（用户可以用宏 DRVPS2_TXFIFO(depth)来设定发送缓冲区深度）。如果总线空闲，数据将马上被发送。当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1。

参数

data [in]

要发送的数据

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_TXDATA1

原型

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

描述

这个宏可以用来填充发送缓冲寄存器 1(共有四个发送寄存器 0~3)，也就是发送缓冲区的 4-7。但是不等待发送缓冲区空，也不设定发送缓冲区深度。（用户可以用宏 DRVPS2_TXFIFO(depth)来设定发送缓冲区深度）

当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1。

参数

data [in]

要发送的数据

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_TXDATA2

原型

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

描述

这个宏可以用来填充发送缓冲寄存器 2(共有四个发送寄存器 0~3)，也就是发送缓冲区的 8-11. 但是不等待发送缓冲区空，也不设定发送缓冲区深度.（用户可以用宏 DRVPS2_TXFIFO(depth)来设定发送缓冲区深度）.

当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1.

参数

data [in]

说明要发送的数据

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_TXDATA3

原型

```
void DRVPS2_TXDATA3(uint32_t data);
```

描述

这个宏可以用来填充发送缓冲寄存器 3(共有四个发送寄存器 0~3)，也就是发送缓冲区的 12-15. 但是不等待发送缓冲区空，也不设定发送缓冲区深度.（用户可以用宏 DRVPS2_TXFIFO(depth)来设定发送缓冲区深度）.

当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1

参数

data [in]

要发送的数据.

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_ISTXEMPTY

原型

```
void DRVPS2_ISTXEMPTY();
```

描述

这个宏可以用来检查发送缓冲区是否为空

当发送字节数等于发送缓冲区深度时，发送缓冲区空标志将被设成 1.

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_ISFRAMEERR

原型

```
void DRVPS2_ISFRAMEERR();
```

描述

这个宏可以用来检查是否发生帧错误。主机发送数据到设备的时候，如果

STOP 比特没有收到，帧错误发生。如果帧错误发生，第 12 个时钟之后，DATA 线将保持在低电平状态。这时软件重置 PS2CLK 来发送时钟信号，直到 PS2DATA 变成高电平。这之后，设备发送一个“Resend”命令到主机。

参数

无

头文件

Driver/ DrvPS2.h

返回值

无.

DRVPS2_ISRXBUSY

原型

```
void DRVPS2_ISRXBUSY();
```

描述

这个宏可以用来检查 PS2 是否正在接收数据。

参数

无

头文件

Driver/DrvPS2.h

返回值

无.

22.2. 函数

DrvPS2_Open

原型

```
int32_t DrvPS2_Open();
```

描述

这个函数可以用来初始化 PS2. 它包括使能 PS2 时钟, 使能 PS2 控制器, 清除发送 FIFO, 设定发送缓冲深度为 0

参数

无

头文件

Driver/DrvPS2.h

返回值

E_SUCCESS.

DrvPS2_Close

原型

```
void DrvPS2_Close();
```

描述

这个函数可以用来关闭 PS2 控制器和 PS2 时钟.

参数

无

头文件

Driver/ DrvPS2.h

返回值

无

DrvPS2_EnableInt

原型

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

描述

这个函数可以用来使能接收/发送中断，并且安装中断回调函数.

参数

u32InterruptFlag [in]

说明要使能的接收/发送中断标志. 可以是 DRVPS2_TXINT 或者 DRVPS2_RXINT 或者 DRVPS2_TXINT| DRVPS2_RXINT

pfncallback [in]

说明中断回调函数指针. 当 PS2 中断发生时，这个函数将被调用

头文件

Driver/ DrvPS2.h

返回值

E_SUCCESS

DrvPS2_DisableInt

原型

```
void DrvPS2_DisableInt(uint32_t u32InterruptFlag);
```

描述

这个函数可以用来关闭接收/发送中断并且卸载中断回调函数

参数

u32InterruptFlag [in]

说明发送/接收中断标志，可以是 DRVPS2_TXINT 或者 DRVPS2_RXINT 或者 DRVPS2_TXINT| DRVPS2_RXINT.

头文件

Driver/ DrvPS2.h

返回值

无

DrvPS2_IsIntEnabled

原型

```
uint32_t DrvPS2_IsIntEnabled(uint32_t u32InterruptFlag);
```

描述

这个函数可以用来检查是否中断被使能

参数

u32InterruptFlag [in]

说明要检查的发送/接收中断标志，可以是 DRVPS2_TXINT 或者 DRVPS2_RXINT 或者 DRVPS2_TXINT| DRVPS2_RXINT.

头文件

Driver/ DrvPS2.h

返回值

无

DrvPS2_ClearIn

原型

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

描述

这个函数可以用来清除中断标志.

参数

U32InterruptFlag [in]

说明要清除的发送/接收中断标志。可以是 DRVPS2_TXINT 或者 DRVPS2_RXINT
或者 DRVPS2_TXINT| DRVPS2_RXINT

头文件

Driver/DrvPS2.h

返回值

E_SUCCESS 成功.

DrvPS2_GetIntStatus

原型

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

描述

这个函数可以用来检查中断标志。如果相应中断发生将返回 TRUE

参数

U32InterruptFlag [in]

说明要检查的发送/接收中断标志。可以是 DRVPS2_TXINT 或者 DRVPS2_RXINT

头文件

Driver/ DrvPS2.h

返回值

TRUE: 相应中断发生

FALSE: 检查的中断没有发生

DrvPS2_SetTxFIFODepth

原型

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

描述

这个函数可以用来设定发送缓冲区深度。这个函数将调用宏 DRVPS2_TXFIFO 来设定发送缓冲区深度

参数

u16TxFIFODepth [in]

说明发送缓冲区深度。范围[0, 15]

头文件

Driver/ DrvPS2.h

返回值

无

DrvPS2_Read

原型

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

描述

这个函数可以用来读一个字节到缓存 pu8RxBuf 中. 这个函数将调用宏 DRVPS2_RXDATA 来接收数据

参数

pu8RxBuf [out]

存放接收数据的缓存地址。缓存只要一个字节就可以

头文件

Driver/ DrvPS2.h

返回值

E_SUCCESS 成功.

DrvPS2_Write

原型

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

描述

这个函数可以用来写缓存 pu32TxBuf 中的数据到主机。如果要发送的数据长度小于 16, 为了性能考虑, 请使用系列宏定义 DRVPS2_TXDATAxxx

参数

pu32TxBuf [in]

要发送的数据.

u32WriteBytes [in]

要发送的数据长度.

头文件

Driver/ DrvPS2.h

返回值

E_SUCCESS 成功.

DrvPS2_GetVersion

原型

int32_t DrvPS2_GetVersion(void);

描述

返回驱动当前版本号.

头文件

Driver/ DrvPS2.h

返回值

版本号 :

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

23. DrvFMC 介绍

23.1. 介绍

NUC1xx 系列配置了 128/64/32k 字节的片上嵌入式闪存，用于存储应用程序(APROM)。NUC1xx 系列还额外提供 4K 字节数据闪存，用于存放应用程序相关的数据。对于 128K 字节的设备，数据闪存和 128K 应用程序闪存共享 128K 字节空间；用户可以从 APROM 中划一块区域用于存放数据，数据区地址直通过 Config1 来配置。

23.2. 特性

FMC 包含下面的特性:

- 128/64/32kB 应用程序闪存 (APROM)，擦除单位 512 字节.
- 4kB 在系统编程闪存(LDROM).
- 4kB 数据闪存，擦除单位 512 字节.
- 128K 字节应用程序闪存数据区开始地址可以配置.

24. DrvFMC APIs 说明

24.1. 函数

DrvFMC_EnableISP

原型

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

描述

使能 ISP 功能。包括读/写/擦除 APROM、LDROM、DATA flash、Config 区域都需要打开这个功能。

参数

i32Enable [in]

1:使能, 0:关闭

头文件

Driver/DrvFMC.h

返回值

无

DrvFMC_BootSelect

原型

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

描述

选择下次从 APROM 还是 LDROM 启动.

参数

boot [in]

说明 APROM 还是 LDROM.

头文件

Driver/DrvFMC.h

返回值

无

DrvFMC_GetBootSelect

原型

E_FMC_BOOTSELECT DrvFMC_GetBootSelect(void);

描述

取得当前启动设定.

参数

无.

头文件

Driver/DrvFMC.h

返回值

APROM 从 APROM 启动

LDROM 从 LDROM 启动

DrvFMC_EnableLDUpdate

原型

void DrvFMC_EnableLDUpdate(int32_t i32Enable);

描述

使能 LDROM 更新功能.

参数

i32Enable [in]

1:使能, 0:关闭

头文件

Driver/DrvFMC.h

返回值

无

DrvFMC_EnablePowerSaving

原型

```
void DrvFMC_EnablePowerSaving(int32_t i32Enable);
```

描述

使能闪存访问节电功能.

参数

i32Enable [in]

1:使能, 0:关闭

头文件

Driver/DrvFMC.h

返回值

无

DrvFMC_ReadCID

原型

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

描述

读取公司 ID.

参数

u32data [in]

存放公司 ID 的缓存指针.

头文件

Driver/DrvFMC.h

返回值

0 成功

<0 失败

DrvFMC_ReadDID

原型

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

描述

读取设备 ID.

参数

u32data [in]

存放设备 ID 的缓存指针.

头文件

Driver/DrvFMC.h

返回值

0 成功

<0 失败

DrvFMC_Write

原型

```
int32_t DrvFMC_Write(uint32_t u32addr, uint32_t u32data);
```

描述

写 4 个字节到 APROM, LDR0M, Data Flash 或者 Config 区域.

参数

u32addr [in]

闪存的地址, 4 字节对齐.

u32data [in]

要写到闪存中的数据.

头文件

Driver/DrvFMC.h

返回值

0 成功

<0 失败

DrvFMC_Read

原型

```
int32_t DrvFMC_Read(uint32_t u32addr, uint32_t * u32data);
```

描述

从 APROM, LDROM, Data Flash 或者 Config 区域中读 4 个字节的数据.

参数

u32addr [in]

闪存的地址, 4 字节对齐.

u32data [in]

缓存地址, 用于存放从闪存中读取的数据.

头文件

Driver/DrvFMC.h

返回值

0 成功

<0 失败

DrvFMC_Erase

原型

```
int32_t DrvFMC_Erase(uint32_t u32addr);
```

描述

以页为单位擦除闪存或者 Config 区域. 每页 512 个字节.

参数

u32addr [in]

闪存或者 Config0 的地址, 页对齐.

头文件

Driver/DrvFMC.h

返回值

0 成功

<0 失败

DrvFMC_WriteConfig

原型

```
int32_t DrvFMC_WriteConfig(uint32_t u32data0, uint32_t u32data1);
```

描述

擦除 Config 区域并且写数据到 Config0 和 Config1.

参数

u32data0 [in]

写到 Config0 的数据.

u32data1 [in]

写到 Config1 的数据.

头文件

Driver/DrvFMC.h

返回值

0 成功

<0 失败

DrvFMC_ReadDataFlashBaseAddr

原型

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

描述

取得数据闪存基地址.

参数

无

头文件

Driver/DrvFMC.h

返回值

数据闪存基地址

25. DrvUSB 介绍

25.1. 介绍

假设用户对 USB1.1/USB2.0 熟悉.

25.2. 特性

- 与 USB2.0 全速兼容, 12Mbps.
- 提供 1 个中断源, 4 个中断事件.
- 支持控制, 批量, 中断, 和等时传输.
- 当没有总线信号超过 3ms 时, 挂起.
- 提供 6 个端点, 可配置.
- 包含 512 个字节的内部 SRAM 用作 USB 缓存.
- 提供远程唤醒能力.

25.3. Call Flow

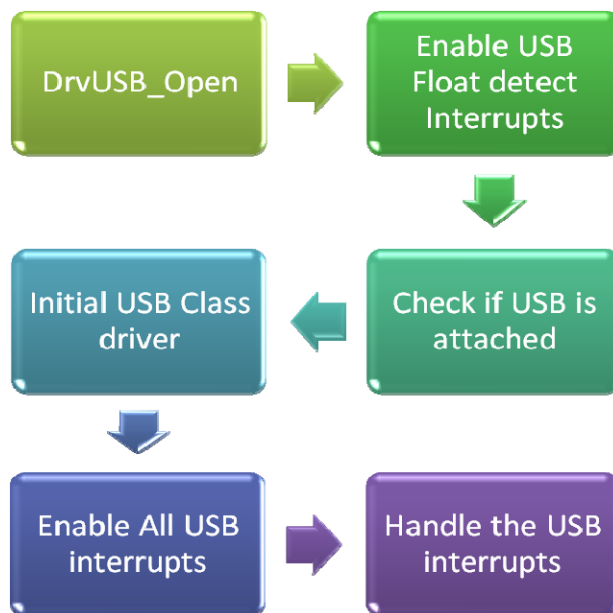


Figure 25-1: USB Driver Call Flow

26. DrvUSB APIs 说明

26.1. 宏

_DRVUSB_ENABLE_MISC_INT

原型

```
static __inline  
void _DRVUSB_ENABLE_MISC_INT (  
    uint32_t    u32Flags  
);
```

描述

使能/关闭各种 USB 中断.

参数

u32Flags [in]

USB 中断事件. 可以是下列的标志.

IEF_WAKEUP: 唤醒中断标志.

IEF_FLD: Float-detection 中断标志 (用来探测 USB 插入/拔出).

IEF_USB: USB 事件中断标志.

IEF_BUS: 总线事件中断标志.

u32Flag = 0 将关闭所有的 USB 中断.

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_ENABLE_WAKEUP

原型

```
static __inline
```



```
void _DRVUSB_ENABLE_WAKEUP (void);
```

描述

使能 USB 唤醒功能.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_WAKEUP

原型

```
static __inline  
void _DRVUSB_DISABLE_WAKEUP (void);
```

描述

关闭 USB 唤醒功能.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_ENABLE_WAKEUP_INT

原型

```
static __inline  
void _DRVUSB_ENABLE_WAKEUP_INT (void);
```

描述

使能唤醒中断.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_WAKEUP_INT

原型

```
static __inline
void _DRVUSB_DISABLE_WAKEUP_INT (void);
```

描述

关闭唤醒中断.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_ENABLE_FLD_INT

原型

```
static __inline
void _DRVUSB_ENABLE_FLD_INT (void);
```

描述

使能 float-detection 中断.USB 插入/拔出将发生中断

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_FLD_INT

原型

```
static __inline
void _DRVUSB_DISABLE_FLD_INT (void);
```

描述

关闭 float-detection 中断.USB 插入/拔出将不会发生中断

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_ENABLE_USB_INT

原型

```
static __inline
void _DRVUSB_ENABLE_USB_INT (void);
```

描述

使能 USB 中断.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_USB_INT

原型

```
static __inline
```

```
void _DRVUSB_DISABLE_USB_INT (void);
```

描述

关闭 USB 中断.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_ENABLE_BUS_INT

原型

```
static __inline  
void _DRVUSB_ENABLE_BUS_INT (void);
```

描述

使能总线中断.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_BUS_INT

原型

```
static __inline  
void _DRVUSB_DISABLE_BUS_INT (void);
```

描述

关闭总线中断.

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL

原型

```
static __inline
void _DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL (
    uint32_t u32EPNum
);
```

描述

清除端点 In/Out 就绪标志并且回应 STALL,

参数

u32EPNum[in]
端点号(有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址.

_DRVUSB_CLEAR_EP_READY

原型

```
static __inline
void _DRVUSB_CLEAR_EP_READY (
    uint32_t u32EPNum
);
```

描述

清除端点 In/Out 就绪标志.

参数

u32EPNum[in]

端点号(有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址.

_DRVUSB_SET_SETUP_BUF

原型

```
static __inline
void _DRVUSB_SET_SETUP_BUF (
    uint32_t    u32BufAddr
);
```

描述

说明 Setup 传输的内部 SRAM 地址.

参数

u32BufAddr [in]

setup token 的传输地址. 必须是 USB_BA+0x100 ~ USB_BA+0x1FF.

头文件

Driver/DrvUsb.h

返回值

无

Notes

u32BufAddr 必须在 USB_BA+0x100 ~ USB_BA+0x1FF 之间, 并且必须是 8 的倍数.

_DRVUSB_SET_EP_BUF

原型

```
static __inline
void _DRVUSB_SET_EP_BUF (
    uint32_t    u32EPNum,
    uint32_t    u32BufAddr
);
```

描述

说明端点传输的内部 SRAM 地址.

参数

u32EPNum [in]

端点号 (有效值: 0 ~ 5).

u32BufAddr [in]

缓存地址.

头文件

Driver/DrvUsb.h

返回值

无

Notes

u32BufAddr 必须在 USB_BA+0x100 ~ USB_BA+0x1FF 之间, 并且必须是 8 的倍数.

这里, 端点号意味着 USB IP 中端点配置索引, 并不是 USB 协议中的端点地址.

_DRVUSB_TRIG_EP

原型

```
static __inline
void _DRVUSB_TRIG_EP (
    uint32_t    u32EPNum,
    uint32_t    u32TrigSize
);
```

描述

触发端点下一次传输。端点配置寄存器可以配置端点成 In/Out

参数

u32EPNum [in]

端点号 (有效值: 0 ~ 5).

u32TrigSize [in]

对 Data Out 传输来说, 这个值意味着从主机接收的最大字节数。对 Data In 传输来说, 这个值意味着发送到主机的字节数。

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址。

_DRVUSB_GET_EP_DATA_SIZE

原型

```
static __inline
uint32_t
_DRVUSB_GET_EP_DATA_SIZE (
    uint32_t    u32EPNum
);
```

描述

取得某个端点发送或者从主机收到的数据长度

参数

u32EPNum [in]

端点号(有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

返回 MXPLDx 寄存器的值, x = 0 ~ 5

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址。

_DRVUSB_SET_EP_TOG_BIT

原型

```
static __inline
```



```
void _DRVUSB_SET_EP_TOG_BIT (
    uint32_t    u32EPNum,
    int32_t     bData0
)
```

描述

收到 IN token 之后，说明发送 Data0 还是 Data1 token.

参数

u32EPNum [in]

端点号 (有效值: 0 ~ 5).

bData0 [in]

说明数据阶段使用 Data0 还是 Data1 token.

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里，端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址.

_DRVUSB_SET_EVF

原型

```
static __inline
void _DRVUSB_SET_EVF (
    uint32_t    u32Data
);
```

描述

写中断事件标志寄存器，以清除中断标志

参数

u32Data [in]

写到 EVF 寄存器中的值

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_GET_EVF

原型

```
static __inline
uint32_t
_DRVUSB_GET_EVF(void);
```

描述

取得中断事件标志寄存器的值

参数

无

头文件

Driver/DrvUsb.h

返回值

返回 EVF 寄存器的值

_DRVUSB_CLEAR_EP_STALL

原型

```
static __inline
void _DRVUSB_CLEAR_EP_STALL (
    uint32_t    u32EPNum
);
```

描述

清除强制端点回应 STALL 标志

参数

u32EPNum [in]
端点号 (有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址

_DRVUSB_TRIG_EP_STALL

原型

```
static __inline
void _DRVUSB_TRIG_EP_STALL (
    uint32_t    u32EPNum
);
```

描述

触发端点(0 ~ 5), 并强制设备响应 STALL

参数

u32EPNum [in]
端点号 (有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址

_DRVUSB_CLEAR_EP_DSQ

原型

```
static __inline
void _DRVUSB_CLEAR_EP_DSQ (
    uint32_t    u32EPNum
);
```

描述

清除 DSQ 比特, 收到 IN token 之后将回 Data0 token

参数

u32EPNum [in]

端点号(有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

无

Notes

这里, 端点号意味着 USB IP 中端点配置索引,并不是 USB 协议中的端点地址

_DRVUSB_SET_CFG

原型

```
static __inline
void _DRVUSB_SET_CFG (
    uint32_t    u32CFGNum,
    uint32_t    u32Data
);
```

描述

配置 CFG 寄存器.

参数

u32CFGNum [in]

CFG 寄存器编号 (有效值: 0 ~ 5).

u32Data [in]

写到 CFG 寄存器的值

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_GET_CFG

原型

```
static __inline
uint32_t
```

```
_DRVUSB_GET_CFG (
    uint32_t    u32CFGNum
);
```

描述

取得 CFG 寄存器的值.

参数

u32CFGNum [in]
CFG 寄存器编号 (有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

返回 CFG 寄存器的值

_DRVUSB_SET_FADDR

原型

```
static __inline
void _DRVUSB_SET_FADDR (
    uint32_t    u32Addr
)
```

描述

设定 USB 设备地址

参数

u32Addr [in]
说明设备地址

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_GET_FADDR

原型

```
static __inline
uint32_t
_DRVUSB_GET_FADDR (void)
```

描述

取得 USB 设备地址

参数

无

头文件

Driver/DrvUsb.h

返回值

USB 设备地址

_DRVUSB_SET_STS

原型

```
static __inline
void _DRVUSB_SET_STS (
    uint32_t    u32Data
)
```

描述

写系统状态寄存器

参数

u32Data [in]

要写到系统状态寄存器（STS）中的值

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_GET_STS

原型

```
static __inline
```

uint32_t

_DRVUSB_GET_STS (void)

描述

取得系统状态寄存器的值

参数

无

头文件

Driver/DrvUsb.h

返回值

系统状态寄存器的值

_DRVUSB_SET_CFGP

原型

static __inline

void _DRVUSB_SET_CFGP(

uint8_t u8CFGPNum,

uint32_t u32Data

);

描述

写 CFGP 寄存器.

参数

u8CFGPNum[in]

CFGP 寄存器编号 (有效值: 0 ~ 5).

u32Data [in]

要写到 CFGP 寄存器中的值

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_GET_CFGP

原型

```
static __inline
uint32_t
_DRVUSB_GET_CFGP (
    uint32_t    u32CFGNum
);
```

描述

取得 CFGP 寄存器的值.

参数

u32CFGNum[in]
CFGP r 寄存器编号 (有效值: 0 ~ 5).

头文件

Driver/DrvUsb.h

返回值

CFGP 寄存器的值

_DRVUSB_ENABLE_USB

原型

```
static __inline
void _DRVUSB_ENABLE_USB (void)
```

描述

使能 USB, PHY , 关闭远程唤醒功能

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_USB

原型

```
static __inline
void _DRVUSB_DISABLE_USB (void)
```

描述

关闭 USB, PHY ， 使能远程唤醒功能

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_PHY

原型

```
static __inline
void _DRVUSB_DISABLE_PHY (void)
```

描述

关闭 PHY ， 并且关闭远程唤醒功能

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_ENABLE_SE0

原型

```
static __inline
void _DRVUSB_ENABLE_SE0 (void)
```

描述

迫使 USB PHY 驱动 SE0。之后 PC 将开始枚举 Usb 设备

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_DISABLE_SE0

原型

```
static __inline
void _DRVUSB_DISABLE_SE0 (void)
```

描述

关闭 SE0。之后 PC 将不会识别 USB 设备

参数

无

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_SET_CFGP0

原型

```
static __inline
void _DRVUSB_SET_CFGP0 (
    uint32_t    u32Data
)
```

描述

设定端点 0 的 CFGP 寄存器.

参数

u32Data [in]

写到 CFGP0 寄存器的值.

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_SET_CFGP1

原型

```
static __inline
void _DRVUSB_SET_CFGP1 (
    uint32_t    u32Data
)
```

描述

设定端点 1 的 CFGP 寄存器.

参数

u32Data [in]

写到 CFGP1 寄存器的值.

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_SET_CFGP2

原型

```
static __inline
void _DRVUSB_SET_CFGP2 (
    uint32_t    u32Data
)
```

描述

设定端点 2 的 CFGP 寄存器.

参数

u32Data [in]

写到 CFGP2 寄存器的值

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_SET_CFGP3

原型

```
static __inline
void _DRVUSB_SET_CFGP3 (
    uint32_t    u32Data
)
```

描述

设定端点 3 的 CFGP 寄存器.

参数

u32Data [in]

写到 CFGP3 寄存器的值.

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_SET_CFGP4

原型

```
static __inline
void _DRVUSB_SET_CFGP4 (
    uint32_t    u32Data
)
```

描述

设定端点 4 的 CFGP 寄存器.

参数

u32Data [in]

写到 CFGP4 寄存器的值.

头文件

Driver/DrvUsb.h

返回值

无

_DRVUSB_SET_CFGP5

原型

```
static __inline
void _DRVUSB_SET_CFGP5 (
    uint32_t    u32Data
)
```

描述

设定端点 5 的 CFGP 寄存器.

参数

u32Data [in]

写到 CFGP5 寄存器的值.

头文件

Driver/DrvUsb.h

返回值

无

26.2. 函数

DrvUSB_Open

原型

```
int32_t
DrvUSB_Open (
    void *    pVoid
```

)

描述

这个函数可以用来复位 USB 控制器，初始化 USB 端点、中断和 USB 驱动结构。如果在调用 DrvUSB_Open 之前，USB 已经插入，函数 DrvUSB_Open 会处理这种情况。用户在调用 DrvUSB_Open 之前必须填好结构 sEpDescription 和 g_sBusOps。

sEpDescription:

sEpDescription 结构定义如下:

```
typedef struct
{
    //比特7 指示方向, 1: input; 0: output
    uint32_t u32EPAddr;
    uint32_t u32MaxPacketSize;
    uint8_t * u8SramBuffer;
}S_DRVUSB_EP_CTRL;
```

这个结构用来设定端点号，最大包尺寸和端点传输缓存地址。NUC1xx 系列 USB 控制器有 6 个端点可用。

g_sBusOps:

g_sBusOps 结构定义如下:

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

可以用来安装 USB 总线事件处理函数，例如:

```
/* 总线事件回调 */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
    {NULL, NULL}, /* 连接事件回调函数 */
    {NULL, NULL}, /* 脱离事件回调函数 */
    {DrvUSB_BusResetCallback, &g_HID_sDevice}, /* 总线复位事件回调函数*/
    {NULL, NULL}, /* 总线暂停事件回调函数*/
    {NULL, NULL}, /* 总线重新开始事件回调函数*/
}
```

```
{DrvUSB_CtrlSetupAck, &g_HID_sDevice},    /* setup 事件回调函数*/
};
```

参数

pVoid

NULL	无
Callback function	中断回调函数指针.

头文件

Driver/DrvUsb.h

返回值

E_SUCCESS: 成功

DrvUSB_Close

原型

```
void    DrvUsb_Close (void);
```

描述

关闭 USB 控制器并且关闭 USB 中断.

头文件

Driver/DrvUsb.h

DrvUSB_PreDispatchEvent

原型

```
void DrvUSB_PreDispatchEvent(void);
```

描述

基于 EVF 寄存器的值, 预转发事件.

参数

无

头文件

Driver/DrvUsb.h

DrvUSB_Isr_PreDispatchEvent

原型

```
void DrvUSB_Isr_PreDispatchEvent(void)
```

描述

基于 EVF 寄存器的值，预转发事件并且同时转发它们。这个函数可以在中断处理函数中调用。

参数

无

头文件

Driver/DrvUsb.h

返回值

无

DrvUSB_DispatchEvent

原型

```
void DrvUSB_Isr_PreDispatchEvent(void)
```

描述

转发杂项和端点事件。杂项事件包括连接/脱离/总线复位/总线暂停/总线重新开始和 setup ACK，杂项事件处理函数由结构 g_sBusOps[] 定义。在使用 USB 驱动之前，用户必须提供 g_sBusOps[]。

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_IsData0

原型

```
int32_t DrvUSB_IsData0(uint32_t u32EpId)
```

描述

检查是否当前 DATA 使用 DATA0. 如果返回 FALSE, 当前 DATA 使用 DATA1.

参数

u32EpId 硬件端点索引. 可以是 0~5.

头文件

Driver/DrvUSB.h

返回值

TRUE 当前数据包使用 DATA0
FALSE 当前数据包使用 DATA1

DrvUSB_GetUsbState

原型

E_DRVUSB_STATE DrvUSB_GetUsbState(void)

描述

取得当前 USB 状态. 状态列表如下:

USB 状态	描述
eDRVUSB_DETACHED	USB 设备已经脱离主机.
eDRVUSB_ATTACHED	USB 设备已经连接到主机.
eDRVUSB_POWERED	The USB is powered.
eDRVUSB_DEFAULT	缺省 USB 状态.
eDRVUSB_ADDRESS	USB 设备已经被分配地址.
eDRVUSB_CONFIGURED	USB 设备已经被设置 CONFIGURATION.
eDRVUSB_SUSPENDED	USB 暂停.

参数

无

头文件

Driver/DrvUSB.h

返回值

当前 USB 状态.

DrvUSB_SetUsbState

原型

```
void DrvUSB_SetUsbState(E_DRVUSB_STATE eUsbState)
```

描述

改变当前 USB 的状态. 关于可用状态, 请参考 DrvUSB_GetUsbState.

参数

eUsbState USB 状态.

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_GetEpIdentity

原型

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

描述

根据端点号和方向取得端点索引. 端点索引可以用来定位 USB 硬件的端点源. 端点索引可以是 0 ~ 5. 端点号由软件分配, 根据 USB 标准可以是 0 ~ 15. 主机通过端点号访问 USB 设备.

参数

u32EpNum 端点号

u32EpAttr 端点属性. 可以是 EP_INPUT 或者 EP_OUTPUT.

头文件

Driver/DrvUSB.h

返回值

0~5 端点地址 u32EpNum 对应的端点索引.

otherwise 不能得到相应的端点索引.

DrvUSB_GetEpId

原型

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

描述

根据端点地址得到端点索引。参数“u32EpNum”和 DrvUSB_GetEpIdentity 的参数不同，因为参数“u32EpNum”的比特 7 包含方向信息。例如：0x81。如果比特 7 是高，意味着这个端点是 EP_INPUT 的，否则是 EP_OUTPUT 的。

参数

u32EpNum 比特 7 带方向信息的端点地址。

头文件

Driver/DrvUSB.h

返回值

0~5 端点地址 u32EpNum 对应的端点索引。

otherwise 不能得到相应的端点索引。

DrvUSB_DataOutTrigger

原型

int32_t DrvUSB_DataOutTrigger(uint32_t u32EpNum, uint32_t u32Size)

描述

写寄存器 MXPLD 来触发数据输出就绪标志。这表示相应的端点内存就绪，可以接收输出的数据包。

参数

u32EpNum 端点号。

u32Size 想从 USB 接收的最大包大小

头文件

Driver/DrvUSB.h

返回值

0 成功

E_DRVUSB_SIZE_TOO_LONG 参数 u32Size 的值大于设定的最大包大小。

DrvUSB_GetOutData

原型

uint8_t * DrvUSB_GetOutData(uint32_t u32EpNum, uint32_t *u32Size)

描述

这个函数将返回端点 u32EpNum 的 USB SRAM 缓存地址。用户可以用这个指针来得到输出数据包的数据。

参数

u32EpNum 端点号.
u32Size 从 USB 收到的数据包大小

头文件

Driver/DrvUSB.h

返回值

返回 USB SRAM 地址.

DrvUSB_DataIn

原型

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

描述

触发数据发送就绪标志。从主机收到 IN token 以后，USB 控制器将发送数据给主机。如果 u8Buffer == NULL && u32Size == 0 则发送数据长度是 0 的 DATA1，否则交替发送 DATA0 和 DATA1。

参数

u32EpNum 端点号.
u8Buffer 收到 IN token 以后要发送的数据.
u32Size 要发送的数据大小.

头文件

Driver/DrvUSB.h

返回值

E_SUCCESS	成功
E_DRVUSB_SIZE_TOO_LONG	u32Size 大于设定的最大包大小

DrvUSB_BusResetCallback

原型

```
void DrvUSB_BusResetCallback(void * pVoid)
```

描述

总线复位处理函数。收到总线复位信号之后，这个函数将被调用。它将复位 USB 地址，设定 SETUP 缓存地址并初始化端点。

参数

pVoid 由结构 g_sBusOps[]传递的参数.

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_InstallClassDevice

原型

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

描述

注册 USB 设备类结构指针到 USB 驱动.

参数

sUsbClass USB 设备类结构指针.

头文件

Driver/DrvUSB.h

返回值

Return USB driver pointer

DrvUSB_InstallCtrlHandler

原型

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

描述

注册控制管道处理函数，包括对 USB 标准定义的 Standard/Vendor/Class 命令的处理。每个命令包括对 SETUP ACK, IN ACK, OUT ACK 的处理函数.

参数

device USB 设备驱动指针.
psCtrlCallbackEntry 处理函数结构指针.

u32RegCnt 处理函数结构大小.

头文件

Driver/DrvUSB.h

返回值

E_SUCCESS 成功
E_DRVUSB_NULL_POINTER 处理函数结构指针为空

DrvUSB_CtrlSetupAck

原型

```
void DrvUSB_CtrlSetupAck(void * pArgu)
```

描述

当 SETUP ack 中断发生时, 这个函数将被调用. 它将根据收到的命令调用 DrvUSB_InstallCtrlHandler 注册的 SETUP 处理函数

参数

pArgu 由结构 g_sBusOps[]传递的参数.

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlDataInAck

原型

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

描述

当 IN ack 中断发生时, 这个函数将被调用. 它将根据收到的命令调用 DrvUSB_InstallCtrlHandler 注册的 IN ACK 处理函数.

参数

pArgu 由结构 g_sBusOps[]传递的参数.

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlDataOutAck

原型

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

描述

当 OUT ack 中断发生时, 这个函数将被调用. 它将根据收到的命令调用 DrvUSB_InstallCtrlHandler 注册的 OUT ACK 处理函数

参数

pArgu 由结构 g_sBusOps[]传递的参数.

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlDataInDefault

原型

```
void DrvUSB_CtrlDataInDefault(void * pVoid)
```

描述

IN ACK 缺省处理函数。收到 IN ACK 之后, 触发 OUT 端点, 接收主机发送的 0 长度的数据包.

参数

pVoid 由函数 DrvUSB_InstallCtrlHandler 传递的参数.

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlDataOutDefault

原型

```
void DrvUSB_CtrlDataOutDefault(void * pVoid)
```

描述

OUT ACK 缺省处理函数。收到主机的 IN token 之后，返回 0 长度的数据包给主机。

参数

pVoid 由函数 DrvUSB_InstallCtrlHandler 传递的参数。

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_Reset

原型

```
void DrvUSB_Reset(uint32_t u32EpNum)
```

描述

根据参数 u32EpNum 恢复相应的 CFGx 和 CFGPx 寄存器的缺省值。

参数

u32EpNum 要复位的端点号

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_ClrCtrlReady

原型

```
void DrvUSB_ClrCtrlReady(void)
```

描述

清除控制管道就绪标志。这个标志由写寄存器 MXPLD 来设定。

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_ClrCtrlReadyAndTrigStall

原型

```
void DrvUSB_ClrCtrlReadyAndTrigStall(void);
```

描述

清除控制管道就绪标志（写寄存器 MXPLD 可设置就绪标志），并发送 STALL.

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_GetSetupBuffer

原型

```
uint32_t DrvUSB_GetSetupBuffer(void)
```

描述

取得 USB SRAM 的 setup 缓存地址.

参数

无

头文件

Driver/DrvUSB.h

返回值

Setup 缓存地址

DrvUSB_GetFreeSram

原型

```
uint32_t DrvUSB_GetFreeSram(void)
```

描述

取得在填写结构 sEpDescription[]之后，空闲的 USB SRAM 缓存地址. 用户可以取得这个地址用于双缓存.

参数

无

头文件

Driver/DrvUSB.h

返回值

空闲的 USB SRAM 地址

DrvUSB_EnableSelfPower

原型

void DrvUSB_EnableSelfPower(void)

描述

使能自供电属性.

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_DisableSelfPower

原型

void DrvUSB_DisableSelfPower(void)

描述

关闭自供电属性.

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_IsSelfPowerEnabled

原型

```
int32_t DrvUSB_IsSelfPowerEnabled(int32_t * pbVoid)
```

描述

查看自供电是使能的还是关闭的.

参数

无

头文件

Driver/DrvUSB.h

返回值

TRUE	USB 设备是自供电的.
FALSE	USB 设备是总线供电的

DrvUSB_EnableRemoteWakeup

原型

```
void DrvUSB_EnableRemoteWakeup(void)
```

描述

使能远程唤醒属性.

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_DisableRemoteWakeup

原型

```
void DrvUSB_DisableRemoteWakeup(void)
```

描述

关闭远程唤醒属性.

参数

无

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_IsRemoteWakeupEnabled

原型

```
int32_t DrvUSB_IsRemoteWakeupEnabled (int32_t * pVoid)
```

描述

查看远程唤醒是使能的还是关闭的.

参数

无

头文件

Driver/DrvUSB.h

返回值

TRUE	USB 设备支持远程唤醒
FALSE	USB 设备不支持远程唤醒

DrvUSB_SetMaxPower

原型

```
int32_t DrvUSB_SetMaxPower(uint32_t u32MaxPower)
```

描述

配置最大电流, 单位 2mA。MaxPower 的最大值是 0xFA (500mA), 缺省值是 0x32 (100mA)

参数

u32MaxPower	最大电流值
-------------	-------

头文件

Driver/DrvUSB.h

返回值

E_SUCCESS	成功
<0	最大值错误

DrvUSB_GetMaxPower

原型

```
int32_t DrvUSB_GetMaxPower(void)
```

描述

取得当前最大电流值，单位 2mA，也就是说 0x32 = 100mA.

参数

无

头文件

Driver/DrvUSB.h

返回值

最大电流值. (单位 2mA)

DrvUSB_EnableUsb

原型

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

描述

使能 USB 和 PHY.

参数

psDevice	USB 设备驱动指针
----------	------------

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_DisableUsb

原型

```
void DrvUSB_DisableUsb(S_DRVUSB_DEVICE * psDevice)
```

描述

关闭 USB 和 PHY.

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_PreDispatchWakeupEvent

原型

```
void DrvUSB_PreDispatchWakeupEvent(S_DRVUSB_DEVICE *psDevice)
```

描述

预转发唤醒事件。这个函数是预留的。

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_PreDispatchFdtEvent

原型

```
void DrvUSB_PreDispatchFdtEvent(S_DRVUSB_DEVICE * psDevice)
```

描述

预转发插入/拔出事件

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_PreDispatchBusEvent

原型

void DrvUSB_PreDispatchBusEvent(S_DRVUSB_DEVICE *psDevice)

描述

预转发总线事件

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_PreDispatchEPEvent

原型

void DrvUSB_PreDispatchEPEvent(S_DRVUSB_DEVICE * psDevice)

描述

预转发端点事件, 包括 IN ACK/IN NAK/OUT ACK/ISO 端点事件. 这个函数用来识别端点事件并记录它们, 将来函数 DrvUSB_DispatchEPEvent()将做进一步处理. 所有端点事件的缺省处理函数定义在 g_sUsbOps[]中.

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_DispatchWakeupEvent

原型

```
void DrvUSB_DispatchWakeupEvent(S_DRVUSB_DEVICE *psDevice)
```

描述

转发唤醒事件。这个函数是预留的。

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_DispatchMiscEvent

原型

```
void DrvUSB_DispatchMiscEvent(S_DRVUSB_DEVICE * psDevice)
```

描述

转发杂项事件。这个事件包含连接/脱离/总线复位/总线暂停/总线重新开始 b 和 setup ACK。杂项事件的处理函数定义在结构 g_sBusOps[]中。

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_DispatchEPEvent

原型

```
void DrvUSB_DispatchEPEvent(S_DRVUSB_DEVICE * psDevice)
```

描述

转发端点事件，它处理由函数 DrvUSB_PreDispatchEPEvent() 转发的事件。包括 IN ACK/IN NAK/OUT ACK/ISO end。端点事件的处理函数定义在结构 g_sUsbOps[]中。

参数

psDevice USB 设备驱动指针

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlSetupSetAddress

原型

void DrvUSB_CtrlSetupSetAddress(void * pVoid)

描述

set address 命令的 Setup ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlSetupClearSetFeature

原型

void DrvUSB_CtrlSetupClearSetFeature(void * pVoid)

描述

Clear feature 命令的 Setup ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlSetupGetConfiguration

原型

void DrvUSB_CtrlSetupGetConfiguration(void * pVoid)

描述

Get configuration 命令的 Setup ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlSetupGetStatus

原型

void DrvUSB_CtrlSetupGetStatus(void * pVoid)

描述

Get status 命令的 Setup ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlSetupGetInterface

原型

void DrvUSB_CtrlSetupGetInterface(void * pVoid)

描述

Get interface 命令的 Setup ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlSetupSetConfiguration

原型

void DrvUSB_CtrlSetupSetConfiguration(void * pVoid)

描述

Set configuration 命令的 Setup ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_CtrlDataInSetAddress

原型

void DrvUSB_CtrlDataInSetAddress(void * pVoid)

描述

Set address 命令的 IN ACK 处理函数

参数

pVoid 由 DrvUSB_InstallCtrlHandler 传递的参数

头文件

Driver/DrvUSB.h

返回值

无

DrvUSB_GetVersion

原型

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

描述

取得驱动当前版本号.

参数

无

头文件

```
Driver/DrvUsb.h
```

返回值

版本号 :

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

27. DvPDMA 介绍

27.1. PDMA 介绍

NUC1xx 包含一个外设直接内存访问(PDMA)控制器，可以读/写内存或者读/写 APB，不需要 CPU 介入。PDMA 有 9 个 DMA 通道(外设到内存或者内存到外设或者内存到内存)。每个 PDMA 通道(PDMA CH0~CH8)，有一个 4 字节的缓存用于缓存到 APB 总线和内存的数据。

软件可以通过关闭 PDMA[PDMACEN]来停止 PDMA 操作。CPU 可以通过软件轮询或者中断的方式来识别一次 PDMA 的完成。NUC1xx PDMA 控制器能增加源或者目标地址，也能固定源/目标地址。

27.2. PDMA 特性

PDMA 包含下面的特性:

- AMBA AHB 主/从接口兼容，用于数据传输和寄存器读写。
- PDMA 支持 32 比特源/目标地址增加/固定。

28. DrvPDMA APIs 说明

28.1. 函数

DrvPDMA_Init

原型

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

描述

这个函数可用来初始化 PDMA

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功.

DrvPDMA_Close

原型

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

描述

这个函数可以用来关闭所有的 PDMA 通道时钟和 AHB PDMA 时钟

参数

无

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功.

DrvPDMA_CHEnableTransfer

原型

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

描述

这个函数可以用来使能 PDMA 某个通道数据读/写功能

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功.

DrvPDMA_CHSoftwareReset

原型

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

描述

这个函数可以用来复位某个通道

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功.

DrvPDMA_Open

原型

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

描述

这个函数可以用来配置 PDMA 通道

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

sParam [in]

配置 PDMA.通道的结构，包括

sSrcAddr: 源地址

sDestAddr: 目标地址

u8TransWidth: 传输宽度

u8Mode: 操作模式

i32ByteCnt: 字节数

头文件

Driver/DrvPDMA.h

返回值

E_DRVPDMA_ERR_PORT_INVALID.	PDMA 通道号错误
E_SUCCESS.	成功

DrvPDMA_ClearInt

原型

```
void
DrvPDMA_ClearInt(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
```



```
E_DRVPDMA_INT_FLAG eIntFlag
);
```

描述

这个函数可以用来清除通道中断状态

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eIntFlag [in]

要清除的中断标志 : eDRVPDMA_TABORT/eDRVPDMA_BLKD

头文件

Driver/DrvPDMA.h

返回值

无

DrvPDMA_PollInt

原型

```
int32_t
DrvPDMA_PollInt(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    E_DRVPDMA_INT_FLAG eIntFlag
);
```

描述

这个函数可以用来轮询通道中断状态

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eIntFlag [in]

要查看的中断标志 : eDRVPDMA_TABORT/eDRVPDMA_BLKD

头文件

Driver/DrvPDMA.h

返回值

True: 查看的中断发生.

False: 查看的中断没有发生.

DrvPDMA_SetAPBTransferWidth

原型

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

描述

这个函数可以用来设定 APB 传输宽度

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eTransferWidth [in]

eDRVPDMA_WIDTH_32BITS

eDRVPDMA_WIDTH_8BITS

eDRVPDMA_WIDTH_16BITS

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS 成功

DrvPDMA_SetCHForAPBDevice

原型

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

描述

这个函数可以用来给 APB 设备选择 PDMA 通道

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eDevice [in]

APB 设备. 可以是

eDRVPDMA_SPI0~3,eDRVPDMA_UART0~1,
eDRVPDMA_USB,eDRVPDMA_ADC

eRWAPB [in]

eDRVPDMA_WRITE_APB / eDRVPDMA_READ_APB

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS	成功
E_DRVPDMA_FALSE_INPUT	参数错误

DrvPDMA_DisableInt

原型

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

描述

这个函数可以用来关闭通道的中断

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eIntSource [in]

中断源 : eDRVPDMA_TABORT/eDRVPDMA_BLKD

头文件

Driver/DrvPDMA.h

返回值

无

DrvPDMA_EnableInt

原型

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

描述

这个函数可以用来使能通道的中断

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eIntSource [in]

中断源: eDRVPDMA_TABORT/eDRVPDMA_BLKD

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功.

DrvPDMA_GetAPBTransferWidth

原型

```
int32_t
DrvPDMA_GetAPBTransferWidth(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

描述

这个函数可以用来取得通道的传输宽度

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功.

DrvPDMA_GetCHForAPBDevice

原型

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

描述

这个函数可以用来取得 APB 设备使用的 PDMA 通道

参数

eDevice [in]

APB 设备。 可以是

eDRVPDMA_SPI0~3,eDRVPDMA_UART0~1,
eDRVPDMA_USB,eDRVPDMA_ADC

eRWAPB [in]

eDRVPDMA_READ_APB/eDRVPDMA_WRITE_APB

头文件

Driver/DrvPDMA.h

返回值

通道号

E_DRVPDMA_FALSE_INPUT 参数错误

DrvPDMA_GetCurrentDestAddr

原型

```
uint32_t
DrvPDMA_GetCurrentDestAddr(
```

```
E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

描述

这个函数可以用来取得 PDMA 通道的当前目标地址

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

当前目标地址

DrvPDMA_GetCurrentSourceAddr

原型

```
uint32_t
DrvPDMA_GetCurrentSourceAddr(
    E_DRVPDMA_CHANNEL_INDEX eChannel
)
```

描述

这个函数可以用来取得 PDMA 通道的当前源地址.

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

当前源地址

DrvPDMA_GetCurrentTransferCount

原型

```
uint32_t
```

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

描述

这个函数可以用来取得 PDMA 通道的当前传输总数

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

当前传输总数

DrvPDMA_GetInternalBufPointer

原型

```
uint32_t  
DrvPDMA_GetInternalBufPointer(  
    E_DRVPDMA_CHANNEL_INDEX eChannel  
);
```

描述

这个函数可以用来取得内部缓存的地址

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

内部缓存的地址

DrvPDMA_GetSharedBufData

原型

```
uint32_t
DrvPDMA_GetSharedBufData(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
)
```

描述

这个函数可以用来取得 PDMA 通道的共享缓存的内容

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

共享缓存的内容(就是共享缓存寄存器 SBUF 的值)

DrvPDMA_GetTransferLength

原型

```
int32_t
DrvPDMA_GetTransferLength(
    E_DRVPDMA_CHANNEL_INDEX eChannel,
    uint32_t* pu32TransferLength
);
```

描述

这个函数可以用来取得 PDMA 通道的传输长度设定

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

pu32TransferLength [in]

指向存放传输长度的缓存指针

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功

DrvPDMA_InstallCallBack

原型

int32_t

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

描述

这个函数可以用来给 PDMA 通道安装中断回调函数

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eIntSource [in]

中断源: eDRVPDMA_TABORT/eDRVPDMA_BLKD

pfncallback [in]

回调函数指针

头文件

Driver/DrvPDMA.h

返回值

E_SUCCESS: 成功

DrvPDMA_IsCHBusy

原型

int32_t

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

描述

这个函数可以用来检查 PDMA 通道是否正在收/发数据

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

头文件

Driver/DrvPDMA.h

返回值

TRUE: 通道正在收/发数据

FALSE: 通道空闲.

DrvPDMA_IsIntEnabled

原型

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

描述

这个函数可以用来检查是否 PDMA 通道的某个中断是使能的

参数

eChannel [in]

说明 PDMA 通道 eDRVPDMA_CHANNEL_0~8

eIntSource [in]

中断源: eDRVPDMA_TABORT/eDRVPDMA_BLKD

头文件

Driver/DrvPDMA.h

返回值

TRUE: 中断源是使能的.

FALSE: 中断源是关闭的.

DrvPDMA_GetVersion

原型

int32_T
DrvPDMA_GetVersion (void);

描述

返回驱动当前版本号.

头文件

Driver/DrvPDMA.h

返回值

版本号:

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

29. Revision History

版本	日期	描述
V1.00.001	1. 8, 2009	• 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.