NuMicro™ NUC100 Series Driver Reference Guide

V1.05.001

Publication Release Date: June. 2011

Support Chips:
NuMicro™ NUC100 series

Support Platforms:
Nuvoton

Table of Contents

1. Overview 18

1.1. Organization 18
1.2. Relative Documents 18
1.3. Abbreviations and Glossaries 18
1.4. Data Type Definition 20

2. SYS Driver 21

2.1. Introduction 21
2.2. Clock Diagram 22
2.3. Type Definition 23
E_SYS_IP_RST 23
E_SYS_IP_CLK 23
E_SYS_PLL_CLKSRC 24
E_SYS_IP_DIV 24
E_SYS_IP_CLKSRC 24
E_SYS_CHIP_CLKSRC 25
E_SYS_PD_TYPE 25
2.4. Functions 25
DrvSYS_ReadProductID 25
DrvSYS_GetResetSource 26
DrvSYS_ClearResetSource 26
DrvSYS_ResetIP 27
DrvSYS_ResetCPU 28
DrvSYS_ResetChip 28
DrvSYS_SelectBODVolt 29
DrvSYS_SetBODFunction 29
DrvSYS_EnableBODLowPowerMode 30
DrvSYS_DisableBODLowPowerMode 31
DrvSYS_EnableLowVoltReset 31
DrvSYS_DisableLowVoltReset 32
DrvSYS_GetBODState 32
DrvSYS_EnableTemperatureSensor 33
DrvSYS_DisableTemperatureSensor 33
DrvSYS_UnlockProtectedReg 34
DrvSYS_LockProtectedReg 34
DrvSYS_IsProtectedRegLocked 35
DrvSYS_EnablePOR 36
DrvSYS_DisablePOR 36
DrvSYS_SetIPClock 37

DrvSYS_SelectHCLKSource 38
DrvSYS_SelectSysTickSource 38
DrvSYS_SelectIPClockSource 39
DrvSYS_SetClockDivider 41
DrvSYS_SetOscCtrl 41
DrvSYS_SetPowerDownWakeUpInt 42
DrvSYS_EnterPowerDown 43
DrvSYS_SelectPLLSource 44
DrvSYS_SetPLLMode 44
DrvSYS_GetExtClockFreq 45
DrvSYS_GetPLLContent 45
DrvSYS_SetPLLContent 46
DrvSYS_GetPLLClockFreq 46
DrvSYS_GetHCLKFreq 47
DrvSYS_Open 47
DrvSYS_SetFreqDividerOutput 48
DrvSYS_EnableHighPerformanceMode 49
DrvSYS_DisableHighPerformanceMode 50
DrvSYS_Delay 50
DrvSYS_GetChipClockSourceStatus 51
DrvSYS_GetClockSwitchStatus 52
DrvSYS_ClearClockSwitchStatus 52
DrvSYS_GetVersion 53

3. UART Driver 54

3.1. UART Introduction 54
3.2. UART Feature 54
3.3. Constant Definition 55
3.4. Type Definition 55
E_UART_PORT 55
E_INT_SOURCE 55
E_DATABITS_SETTINGS 55
E_PARITY_SETTINGS 55
E_STOPBITS_SETTINGS 56
E_FIFO_SETTINGS 56
E_UART_FUNC 56
E_MODE_RS485 56
3.5. Macros 57
_DRVUART_SENDBYTE 57
_DRVUART_RECEIVEBYTE 57
_DRVUART_SET_DIVIDER 57
_DRVUART_RECEIVEAVAILABLE 58
_DRVUART_WAIT_TX_EMPTY 58
3.6. Functions 59
DrvUART_Open 59
DrvUART_Close 61
DrvUART_EnableInt 61
DrvUART_DisableInt 62

DrvUART_ClearIntFlag 63
DrvUART_GetIntStatus 64
DrvUART_GetCTSInfo 65
DrvUART_SetRTS 66
DrvUART_Read 67
DrvUART_Write 68
DrvUART_EnablePDMA 69
DrvUART_DisablePDMA 69
DrvUART_SetFnIRDA 70
DrvUART_SetFnRS485 71
DrvUART_SetFnLIN 72
DrvUART_GetVersion 73

4. TIMER/WDT Driver 74

4.1. TIMER/WDT Introduction 74

4.2. TIMER/WDT Feature 74

4.3. Type Definition 74

E_TIMER_CHANNEL 74
E_TIMER_OPMODE 75
E_TIMER_TX_PHASE 75
E_TIMER_TEX_EDGE 75
E_TIMER_RSTCAP_MODE 75
E_WDT_CMD 75
E_WDT_INTERVAL 76

4.4. Functions 76

DrvTIMER_Init 76
DrvTIMER_Open 77
DrvTIMER_Close 78
DrvTIMER_SetTimerEvent 78
DrvTIMER_ClearTimerEvent 79
DrvTIMER_EnableInt 80
DrvTIMER_DisableInt 80
DrvTIMER_GetIntFlag 81
DrvTIMER_ClearIntFlag 81
DrvTIMER_Start 82
DrvTIMER_GetIntTicks 82
DrvTIMER_ResetIntTicks 83
DrvTIMER_Delay 83
DrvTIMER_OpenCounter 84
DrvTIMER_StartCounter 85
DrvTIMER_GetCounters 85
DrvTIMER_OpenCapture 86
DrvTIMER_CloseCapture 87
DrvTIMER_SelectExternalMode 87
DrvTIMER_SelectCaptureEdge 88
DrvTIMER_EnableCaptureInt 89
DrvTIMER_DisableCaptureInt 89
DrvTIMER_EnableCapture 90
DrvTIMER_DisableCapture 90
DrvTIMER_GetCaptureData 91

DrvTIMER_GetCaptureIntFlag....92
DrvTIMER_ClearCaptureIntFlag....92
DrvTIMER_EnableCaptureDebounce....93
DrvTIMER_DisableCaptureDebounce....94
DrvTIMER_EnableCounterDebounce....94
DrvTIMER_DisableCounterDebounce....95
DrvTIMER_SelectCounterDetectPhase....96
DrvTIMER_GetVersion....96
DrvWDT_Open....97
DrvWDT_Close....97
DrvWDT_InstallISR....98
DrvWDT_Ioctl....98

5. GPIO Driver....100

5.1. GPIO introduction....100

5.2. GPIO Feature....100

5.3. Type Definition....100

E_DRVGPIO_PORT....100
E_DRVGPIO_IO....100
E_DRVGPIO_INT_TYPE....101
E_DRVGPIO_INT_MODE....101
E_DRVGPIO_DBCLKSRC....101
E_DRVGPIO_FUNC....101

5.4. Macros....102

_DRVGPIO_DOUT....102
GPA_[n] / GPB_[n] / GPC_[n] / GPD_[n] / GPE_[n]....103

5.5. Functions....104

DrvGPIO_Open....104
DrvGPIO_Close....105
DrvGPIO_SetBit....105
DrvGPIO_GetBit....106
DrvGPIO_ClrBit....107
DrvGPIO_SetPortBits....107
DrvGPIO_GetPortBits....108
DrvGPIO_GetDoutBit....108
DrvGPIO_GetPortDoutBits....109
DrvGPIO_SetBitMask....110
DrvGPIO_GetBitMask....110
DrvGPIO_ClrBitMask....111
DrvGPIO_SetPortMask....112
DrvGPIO_GetPortMask....112
DrvGPIO_ClrPortMask....113
DrvGPIO_EnableDigitalInputBit....113
DrvGPIO_DisableDigitalInputBit....114
DrvGPIO_EnableDebounce....115
DrvGPIO_DisableDebounce....115
DrvGPIO_SetDebounceTime....116
DrvGPIO_GetDebounceSampleCycle....116
DrvGPIO_EnableInt....117

DrvGPIO_DisableInt 118
DrvGPIO_SetIntCallback 119
DrvGPIO_EnableEINT0 119
DrvGPIO_DisableEINT0 120
DrvGPIO_EnableEINT1 120
DrvGPIO_DisableEINT1 121
DrvGPIO_GetIntStatus 122
DrvGPIO_InitFunction 122
DrvGPIO_GetVersion 123

6. ADC Driver 125

6.1. ADC Introduction 125

6.2. ADC Feature 125

6.3. Type Definition 126
E_ADC_INPUT_MODE 126
E_ADC_OPERATION_MODE 126
E_ADC_CLK_SRC 126
E_ADC_EXT_TRI_COND 126
E_ADC_CH7_SRC 126
E_ADC_CMP_CONDITION 126
E_ADC_DIFF_MODE_OUTPUT_FORMAT 126

6.4. Macros 127
_DRVADC_CONV 127
_DRVADC_GET_ADC_INT_FLAG 127
_DRVADC_GET_CMP0_INT_FLAG 128
_DRVADC_GET_CMP1_INT_FLAG 128
_DRVADC_CLEAR_ADC_INT_FLAG 128
_DRVADC_CLEAR_CMP0_INT_FLAG 129
_DRVADC_CLEAR_CMP1_INT_FLAG 129

6.5. Functions 130
DrvADC_Open 130
DrvADC_Close 131
DrvADC_SetADCChannel 132
DrvADC_ConfigADCChannel7 132
DrvADC_SetADCInputMode 133
DrvADC_SetADCOperationMode 134
DrvADC_SetADCClkSrc 134
DrvADC_SetADCDivisor 135
DrvADC_EnableADCInt 135
DrvADC_DisableADCInt 136
DrvADC_EnableADCCmp0Int 137
DrvADC_DisableADCCmp0Int 138
DrvADC_EnableADCCmp1Int 138
DrvADC_DisableADCCmp1Int 139
DrvADC_GetConversionRate 140
DrvADC_EnableExtTrigger 140
DrvADC_DisableExtTrigger 141
DrvADC_StartConvert 141
DrvADC_StopConvert 142

DrvADC_IsConversionDone....142
DrvADC_GetConversionData....143
DrvADC_EnablePDMA....143
DrvADC_DisablePDMA....144
DrvADC_IsDataValid....144
DrvADC_IsDataOverrun....145
DrvADC_EnableADCCmp0....145
DrvADC_DisableADCCmp0....146
DrvADC_EnableADCCmp1....147
DrvADC_DisableADCCmp1....148
DrvADC_EnableSelfCalibration....148
DrvADC_IsCalibrationDone....149
DrvADC_DisableSelfCalibration....149
DrvADC_DiffModeOutputFormat....150
DrvADC_GetVersion....150

7. SPI Driver....152

7.1. SPI Introduction....152
7.2. SPI Feature....152
7.3. Type Definition....153
E_DRVSPI_PORT....153
E_DRVSPI_MODE....153
E_DRVSPI_TRANS_TYPE....153
E_DRVSPI_ENDIAN....153
E_DRVSPI_BYTE_REORDER....153
E_DRVSPI_SSLTRIG....154
E_DRVSPI_SS_ACT_TYPE....154
E_DRVSPI_SLAVE_SEL....154
E_DRVSPI_DMA_MODE....154
7.4. Functions....155
DrvSPI_Open....155
DrvSPI_Close....156
DrvSPI_Set2BitTransferMode....157
DrvSPI_SetEndian....158
DrvSPI_SetBitLength....158
DrvSPI_SetByteReorder....159
DrvSPI_SetSuspendCycle....160
DrvSPI_SetTriggerMode....161
DrvSPI_SetSlaveSelectActiveLevel....162
DrvSPI_GetLevelTriggerStatus....163
DrvSPI_EnableAutoSS....164
DrvSPI_DisableAutoSS....165
DrvSPI_SetSS....165
DrvSPI_ClrSS....166
DrvSPI_IsBusy....167
DrvSPI_BurstTransfer....168
DrvSPI_SetClockFreq....169
DrvSPI_GetClock1Freq....170
DrvSPI_GetClock2Freq....171
DrvSPI_SetVariableClockFunction....171

DrvSPI_EnableInt....173
DrvSPI_DisableInt....174
DrvSPI_GetIntFlag....174
DrvSPI_ClrIntFlag....175
DrvSPI_SingleRead....176
DrvSPI_SingleWrite....176
DrvSPI_BurstRead....177
DrvSPI_BurstWrite....178
DrvSPI_DumpRxRegister....179
DrvSPI_SetTxRegister....179
DrvSPI_SetGo....180
DrvSPI_ClrGo....181
DrvSPI_SetPDMA....182
DrvSPI_SetFIFOMode....183
DrvSPI_IsRxEmpty....183
DrvSPI_IsRxFull....184
DrvSPI_IsTxEmpty....185
DrvSPI_IsTxFull....186
DrvSPI_ClrRxFIFO....187
DrvSPI_ClrTxFIFO....188
DrvSPI_EnableDivOne....188
DrvSPI_DisableDivOne....189
DrvSPI_Enable3Wire....190
DrvSPI_Disable3Wire....190
DrvSPI_3WireAbort....191
DrvSPI_Enable3WireStartInt....192
DrvSPI_Disable3WireStartInt....193
DrvSPI_Get3WireStartIntFlag....193
DrvSPI_Clr3WireStartIntFlag....194
DrvSPI_GetVersion....195

8. I2C Driver....196

8.1. I2C Introduction....196

8.2. I2C Feature....196

8.3. Type Definition....196

E_I2C_PORT....196
E_I2C_CALLBACK_TYPE....196

8.4. Functions....197

DrvI2C_Open....197
DrvI2C_Close....197
DrvI2C_SetClockFreq....198
DrvI2C_GetClockFreq....198
DrvI2C_SetAddress....199
DrvI2C_SetAddressMask....200
DrvI2C_GetStatus....200
DrvI2C_WriteData....201
DrvI2C_ReadData....201
DrvI2C_Ctrl....202
DrvI2C_GetIntFlag....203
DrvI2C_ClearIntFlag....203

DrvI2C_EnableInt....204
DrvI2C_DisableInt....204
DrvI2C_InstallCallBack....205
DrvI2C_UninstallCallBack....205
DrvI2C_SetTimeoutCounter....206
DrvI2C_ClearTimeoutFlag....207
DrvI2C_GetVersion....207

9. RTC Driver....209

9.1. RTC Introduction....209
9.2. RTC Features....209
Constant Definition....210
9.3. Type Definition....210
E_DRVRTC_INT_SOURCE....210
E_DRVRTC_TICK....210
E_DRVRTC_TIME_SELECT....210
E_DRVRTC_DWR_PARAMETER....211
9.4. Functions....211
DrvRTC_SetFrequencyCompensation....211
DrvRTC_IsLeapYear....212
DrvRTC_GetIntTick....212
DrvRTC_ResetIntTick....213
DrvRTC_WriteEnable....213
DrvRTC_Init....214
DrvRTC_SetTickMode....214
DrvRTC_EnableInt....215
DrvRTC_DisableInt....216
DrvRTC_Open....217
DrvRTC_Read....218
DrvRTC_Write....219
DrvRTC_Close....220
DrvRTC_GetVersion....220

10. CAN Driver....221

10.1. CAN Introduction....221
10.2. CAN Feature....221
10.3. Constant Definition....221
10.4. Functions....222
DrvCAN_Init....222
DrvCAN_Close....223
DrvCAN_Open....223
DrvCAN_SetTiming....224
DrvCAN_ResetMsgObj....225
DrvCAN_ResetAllMsgObj....225
DrvCAN_SetTxMsgObj....226

DrvCAN_SetMsgObjMask....227
DrvCAN_SetRxMsgObj....228
DrvCAN_ClrIntPnd....229
DrvCAN_SetTxRqst....229
DrvCAN_ReadMsgObj....230
DrvCAN_WaitEndOfTx....230
DrvCAN_BasicSendMsg....231
DrvCAN_BasicReceiveMsg....232
DrvCAN_EnterInitMode....232
DrvCAN_LeaveInitMode....233
DrvCAN_EnterTestMode....233
DrvCAN_LeaveTestMode....234
DrvCAN_IsNewDataReceived....234
DrvCAN_IsTxRqstPending....235
DrvCAN_IsIntPending....235
DrvCAN_IsObjectValid....236
DrvCAN_ResetIF....237
DrvCAN_WaitMsg....237
DrvCAN_EnableInt....238
DrvCAN_DisableInt....238
DrvCAN_InstallCallback....239
DrvCAN_UninstallCallback....239
DrvCAN_EnableWakeUp....240
DrvCAN_DisableWakeUp....240
DrvCAN_GetCANBitRate....241
DrvCAN_GetTxErrCount....241
DrvCAN_GetRxErrCount....242
DrvCAN_GetVersion....242

11. PWM Driver....244

11.1. PWM Introduction....244
11.2. PWM Features....244
11.3. Constant Definition....245
11.4. Functions....246

DrvPWM_IsTimerEnabled....246
DrvPWM_SetTimerCounter....246
DrvPWM_GetTimerCounter....247
DrvPWM_EnableInt....248
DrvPWM_DisableInt....249
DrvPWM_ClearInt....250
DrvPWM_GetIntFlag....251
DrvPWM_GetRisingCounter....252
DrvPWM_GetFallingCounter....252
DrvPWM_GetCaptureIntStatus....253
DrvPWM_ClearCaptureIntStatus....254
DrvPWM_Open....255
DrvPWM_Close....255
DrvPWM_EnableDeadZone....256
DrvPWM_Enable....257
DrvPWM_SetTimerClk....258

DrvPWM_SetTimerIO....261
DrvPWM_SelectClockSource....262
DrvPWM_SelectClearLatchFlagOption....263
DrvPWM_GetVersion....263

12. PS2 Driver....265

12.1. PS2 Introduction....265

12.2. PS2 Feature....265

12.3. Constant Defination....265

12.4. Macros....266

_DRVPS2_OVERRIDE....266
_DRVPS2_PS2CLK....266
_DRVPS2_PS2DATA....267
_DRVPS2_CLRFIFO....267
_DRVPS2_ACKNOTALWAYS....268
_DRVPS2_ACKALWAYS....268
_DRVPS2_RXINTENABLE....269
_DRVPS2_RXINTDISABLE....269
_DRVPS2_TXINTENABLE....270
_DRVPS2_TXINTDISABLE....270
_DRVPS2_PS2ENABLE....271
_DRVPS2_PS2DISABLE....271
_DRVPS2_TXFIFO....272
_DRVPS2_SWOVERRIDE....272
_DRVPS2_INTCLR....273
_DRVPS2_RXDATA....274
_DRVPS2_TXDATAWAIT....274
_DRVPS2_TXDATA....275
_DRVPS2_TXDATA0....276
_DRVPS2_TXDATA1....277
_DRVPS2_TXDATA2....277
_DRVPS2_TXDATA3....278
_DRVPS2_ISTXEMPTY....278
_DRVPS2_ISFRAMEERR....279
_DRVPS2_ISRXBUSY....279

12.5. Functions....280

DrvPS2_Open....280
DrvPS2_Close....281
DrvPS2_EnableInt....281
DrvPS2_DisableInt....282
DrvPS2_IsIntEnabled....282
DrvPS2_ClearIn....283
DrvPS2_GetIntStatus....284
DrvPS2_SetTxFIFODepth....285
DrvPS2_Read....285
DrvPS2_Write....286
DrvPS2_GetVersion....286

13. FMC Driver....288

13.1. FMC Introduction 288
13.2. FMC Feature 288
Memory Address Map 288
Flash Memory Structure 289
13.3. Type Definition 289
E_FMC_BOOTSELECT 289
13.4. Functions 289
DrvFMC_EnableISP 289
DrvFMC_DisableISP 290
DrvFMC_BootSelect 291
DrvFMC_GetBootSelect 291
DrvFMC_EnableLDUpdate 292
DrvFMC_DisableLDUpdate 292
DrvFMC_EnableConfigUpdate 293
DrvFMC_DisableConfigUpdate 293
DrvFMC_EnableAPUpdate 294
DrvFMC_DisableAPUpdate 295
DrvFMC_EnablePowerSaving 295
DrvFMC_DisablePowerSaving 296
DrvFMC_Write 296
DrvFMC_Read 297
DrvFMC_Erase 298
DrvFMC_WriteConfig 298
DrvFMC_ReadDataFlashBaseAddr 299
DrvFMC_EnableLowFreqOptMode 300
DrvFMC_DisableLowFreqOptMode 300
DrvFMC_GetVersion 301
14. USB Driver 302
14.1. Introduction 302
14.2. Feature 302
14.3. USB Framework 303
14.4. Call Flow 304
14.5. Constant Definition 304
USB Register Address 304
INTEN Register Bit Definition 305
INTSTS Register Bit Definition 306
ATTR Register Bit Definition 306
Confiuration Register Bit Definition 306
Extera-Confiuration Register Bit Definition 307
14.6. Macro 307
_DRVUSB_ENABLE_MISC_INT 307
_DRVUSB_ENABLE_WAKEUP 308
_DRVUSB_DISABLE_WAKEUP 308
_DRVUSB_ENABLE_WAKEUP_INT 309

_DRVUSB_DISABLE_WAKEUP_INT............309
_DRVUSB_ENABLE_FLDET_INT............310
_DRVUSB_DISABLE_FLDET_INT............310
_DRVUSB_ENABLE_USB_INT............311
_DRVUSB_DISABLE_USB_INT............311
_DRVUSB_ENABLE_BUS_INT............312
_DRVUSB_DISABLE_BUS_INT............312
_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL............312
_DRVUSB_CLEAR_EP_READY............313
_DRVUSB_SET_SETUP_BUF............314
_DRVUSB_SET_EP_BUF............314
_DRVUSB_TRIG_EP............315
_DRVUSB_GET_EP_DATA_SIZE............316
_DRVUSB_SET_EP_TOG_BIT............316
_DRVUSB_SET_EVENT_FLAG............317
_DRVUSB_GET_EVENT_FLAG............318
_DRVUSB_CLEAR_EP_STALL............318
_DRVUSB_TRIG_EP_STALL............319
_DRVUSB_CLEAR_EP_DSQ_SYNC............319
_DRVUSB_SET_CFG............320
_DRVUSB_GET_CFG............321
_DRVUSB_SET_FADDR............321
_DRVUSB_GET_FADDR............322
_DRVUSB_GET_EPSTS............322
_DRVUSB_SET_CFGP............323
_DRVUSB_GET_CFGP............324
_DRVUSB_ENABLE_USB............324
_DRVUSB_DISABLE_USB............325
_DRVUSB_DISABLE_PHY............325
_DRVUSB_ENABLE_SE0............326
_DRVUSB_DISABLE_SE0............326
_DRVUSB_SET_CFG_EP0............327
_DRVUSB_SET_CFG_EP1............327
_DRVUSB_SET_CFG_EP2............328
_DRVUSB_SET_CFGP3............329
_DRVUSB_SET_CFGP4............329
_DRVUSB_SET_CFGP5............330

14.7. Functions............331

DrvUSB_GetVersion............331
DrvUSB_Open............331
DrvUSB_Close............333
DrvUSB_PreDispatchEvent............333
DrvUSB_DispatchEvent............334
DrvUSB_IsData0............334
DrvUSB_GetUsbState............335
DrvUSB_SetUsbState............336
DrvUSB_GetEpIdentity............336
DrvUSB_GetEpId............337
DrvUSB_DataOutTrigger............337
DrvUSB_GetOutData............338
DrvUSB_DataIn............338
DrvUSB_BusResetCallback............339
DrvUSB_InstallClassDevice............340

DrvUSB_InstallCtrlHandler 340
DrvUSB_CtrlSetupAck 341
DrvUSB_CtrlDataInAck...342
DrvUSB_CtrlDataOutAck 342
DrvUSB_CtrlDataInDefault 343
DrvUSB_CtrlDataOutDefault...344
DrvUSB_Reset 344
DrvUSB_ClrCtrlReady..345
DrvUSB_ClrCtrlReadyAndTrigStall...345
DrvUSB_GetSetupBuffer 346
DrvUSB_GetFreeSRAM 346
DrvUSB_EnableSelfPower...347
DrvUSB_DisableSelfPower..347
DrvUSB_IsSelfPowerEnabled 348
DrvUSB_EnableRemoteWakeup...348
DrvUSB_DisableRemoteWakeup..349
DrvUSB_IsRemoteWakeupEnabled..349
DrvUSB_SetMaxPower...350
DrvUSB_GetMaxPower 350
DrvUSB_EnableUSB 351
DrvUSB_DisableUSB 351
DrvUSB_PreDispatchWakeupEvent 352
DrvUSB_PreDispatchFDTEvent 352
DrvUSB_PreDispatchBusEvent 353
DrvUSB_PreDispatchEPEvent..353
DrvUSB_DispatchWakeupEvent...354
DrvUSB_DispatchMiscEvent...354
DrvUSB_DispatchEPEvent 355
DrvUSB_CtrlSetupSetAddress..355
DrvUSB_CtrlSetupClearSetFeature 356
DrvUSB_CtrlSetupGetConfiguration 356
DrvUSB_CtrlSetupGetStatus..357
DrvUSB_CtrlSetupGetInterface 357
DrvUSB_CtrlSetupSetInterface...358
DrvUSB_CtrlSetupSetConfiguration..358
DrvUSB_CtrlDataInSetAddress 359
DrvUSB_memcpy..359

15. PDMA Driver 361

15.1. PDMA Introduction 361
15.2. PDMA Feature 361
15.3. Constant Definition.. 361
15.4. Type Definition.. 361

E_DRVPDMA_CHANNEL_INDEX...361
E_DRVPDMA_DIRECTION_SELECT 362
E_DRVPDMA_TRANSFER_WIDTH 362
E_DRVPDMA_INT_ENABLE..362
E_DRVPDMA_APB_DEVICE...362
E_DRVPDMA_APB_RW..363
E_DRVPDMA_MODE 363

15.5. Functions 363
DrvPDMA_Init 363
DrvPDMA_Close 363
DrvPDMA_CHEnableTransfer 364
DrvPDMA_CHSoftwareReset 365
DrvPDMA_Open 365
DrvPDMA_ClearIntFlag 367
DrvPDMA_PollInt 368
DrvPDMA_SetAPBTransferWidth 369
DrvPDMA_SetCHForAPBDevice 370
DrvPDMA_SetSourceAddress 371
DrvPDMA_SetDestAddress 371
DrvPDMA_DisableInt 372
DrvPDMA_EnableInt 373
DrvPDMA_GetAPBTransferWidth 373
DrvPDMA_GetCHForAPBDevice 374
DrvPDMA_GetCurrentDestAddr 375
DrvPDMA_GetCurrentSourceAddr 376
DrvPDMA_GetRemainTransferCount 377
DrvPDMA_GetInternalBufPointer 377
DrvPDMA_GetSharedBufData 379
DrvPDMA_GetTransferLength 380
DrvPDMA_GetSourceAddress 380
DrvPDMA_GetDestAddress 381
DrvPDMA_InstallCallBack 382
DrvPDMA_IsCHBusy 383
DrvPDMA_IsIntEnabled 383
DrvPDMA_GetVersion 384

16. I2S Driver 386

16.1. I2S Introduction 386
16.2. I2S Feature 386
16.3. Constant Definition 387
16.4. Type Definition 388
E_I2S_CHANNEL 388
E_I2S_CALLBACK_TYPE 388
16.5. Macro Functions 388
_DRVI2S_WRITE_TX_FIFO 388
_DRVI2S_READ_RX_FIFO 389
_DRVI2S_READ_TX_FIFO_LEVEL 389
_DRVI2S_READ_RX_FIFO_LEVEL 390
16.6. Functions 391
DrvI2S_Open 391
DrvI2S_Close 392
DrvI2S_EnableInt 392
DrvI2S_DisableInt 393
DrvI2S_GetBCLKFreq 394
DrvI2S_SetBCLKFreq 395

DrvI2S_GetMCLKFreq ... 395
DrvI2S_SetMCLKFreq ... 396
DrvI2S_SetChannelZeroCrossDetect ... 396
DrvI2S_EnableTxDMA ... 397
DrvI2S_DisableTxDMA ... 397
DrvI2S_EnableRxDMA ... 398
DrvI2S_DisableRxDMA ... 398
DrvI2S_EnableTx ... 399
DrvI2S_DisableTx ... 399
DrvI2S_EnableRx ... 400
DrvI2S_DisableRx ... 400
DrvI2S_EnableTxMute ... 401
DrvI2S_DisableTxMute ... 401
DrvI2S_EnableMCLK ... 401
DrvI2S_DisableMCLK ... 402
DrvI2S_ClearTxFIFO ... 402
DrvI2S_ClearRxFIFO ... 403
DrvI2S_SelectClockSource ... 403
DrvI2S_GetSourceClockFreq ... 404
DrvI2S_GetVersion ... 404

17. EBI Driver ... 406

17.1. EBI Introduction ... 406
17.2. EBI Feature ... 406
17.3. Type Definition ... 407
E_DRVEBI_DATA_WIDTH ... 407
E_DRVEBI_ADDR_WIDTH ... 407
E_DRVEBI_MCLKDIV ... 407
17.4. API Functions ... 408
DrvEBI_Open ... 408
DrvEBI_Close ... 409
DrvEBI_SetBusTiming ... 409
DrvEBI_GetBusTiming ... 410
DrvEBI_GetVersion ... 411

18. Appendix ... 412

18.1. NuMicro™ NUC100 Series Products Selection Guide ... 412
18.2. PDID Table ... 414

19. Revision History ... 416

1. Overview

1.1. Organization

This document describes the NuMicro™ NUC100 series driver reference manual. System-level software developers can use the NuMicro™ NUC100 series driver to do the fast application software development, instead of using the register level programming, which can reduce the total development time significantly. In this document, a description, usage and an illustrated example code are provided for each driver application interface. The full driver samples and driver source codes can be found in the BSP (Board Support Package) of the NuMicro™ NUC100 series.

This document is organized into several chapters. Chapter 1 is an overview. From Chapter 2 to Chapter 17 are the detailed driver descriptions including the followings: System Driver, UART Driver, Timer Driver, GPIO Driver, ADC Driver, SPI Driver, I2C Driver, RTC Driver, CAN Driver, PWM Driver, PS2 Driver, FMC Driver, USB Driver, PDMA Driver, I2S Driver and EBI Driver.

Finally, for the NuMicro™ NUC100 series selection guide and product identity list are described in Appendix.

1.2. Relative Documents

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

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

1.3. Abbreviations and Glossaries

ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
BOD	Brown Out Detection
BUF	Buffer
CAN	Controller Area Network
CFG	Configuration
DSQ	Data Sequence

EBI	External Bus Interface
EP	End Point
FIFO	First-In-First-Out
FLD	Float-Detection
FMC	Flash Memory Controller
GPIO	General Purpose Input/Output
I2C	Inter Integrated Circuit
I2S	Integrated Interchip Sound
LIN	Local Interconnect Network
LVR	Low Voltage Reset
PDID	Product Device Identity
PDMA	Peripheral Direct Memory Access
PHY	Physical layer
PLL	Phase-Locked Loop
POR	Power On Reset
PWM	Pulse-Width Modulation
PS/2	IBM Personal System/2
SPI	Serial Peripheral Interface
TOG	Toggle
TRIG	Trigger
TRM	Technical Reference Manual
UART	Universal Asynchronous Receiver/Transmitter

1.4. Data Type Definition

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

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

2. SYS Driver

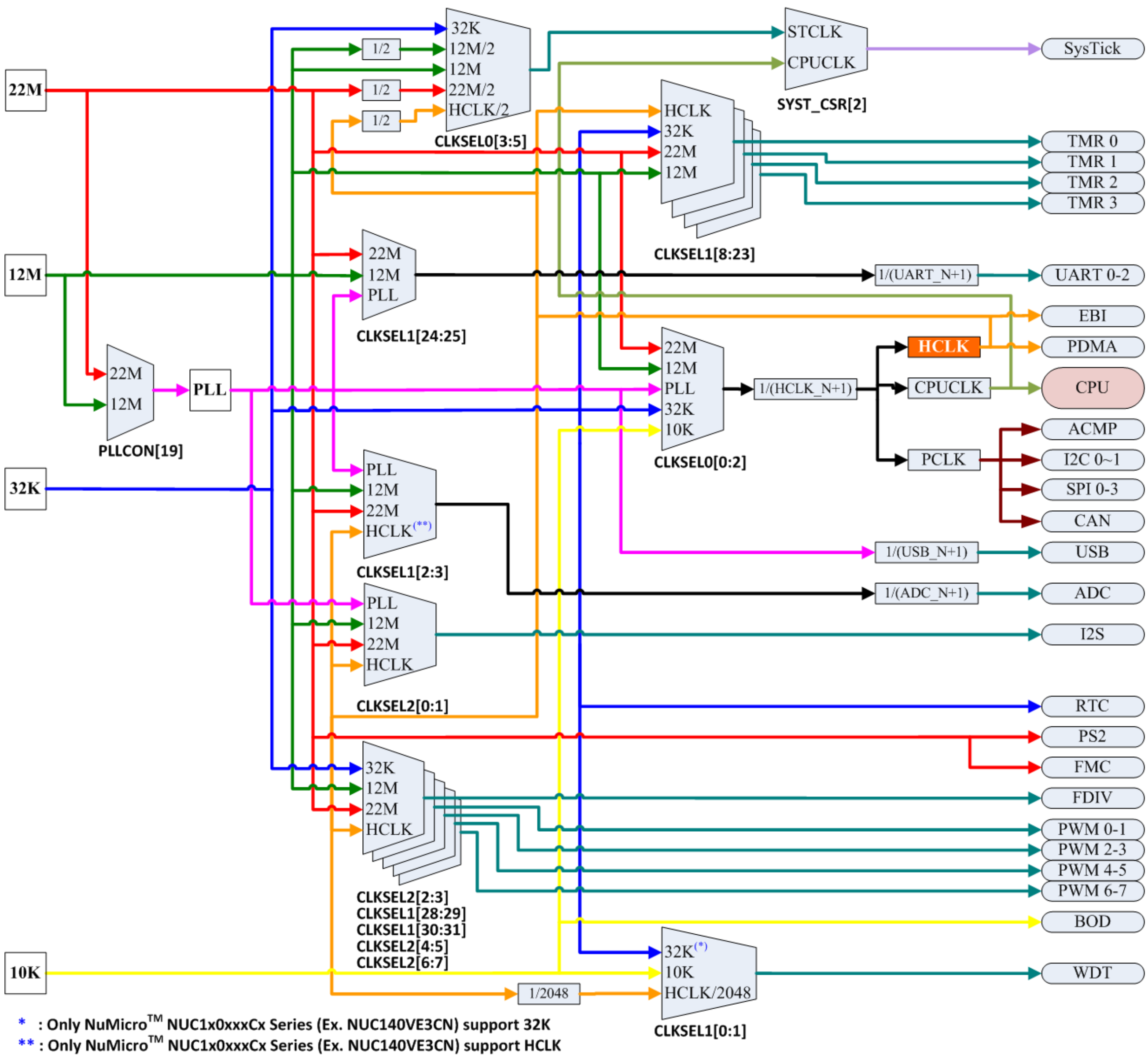
2.1. Introduction

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

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

2.2. Clock Diagram

The clock diagram shows all relative clocks for the whole chip, including system clocks (CPU clock, HCLK, and PCLK) and all peripheral clocks. Here, 12M means the external crystal clock source and it is connected with 12MHz crystal. 22M means internal 22MHz RC clock source and its frequency is 22.1184Mhz with 3% deviation. 32K means the external 32768Hz crystal for RTC purpose. 10K means internal 10KHz RC clock source with 50% deviation.



2.3. Type Definition

E_SYS_IP_RST

Enumeration identifier	Value	Description
E_SYS_GPIO_RST	1	GPIO reset
E_SYS_TMR0_RST	2	Timer0 reset
E_SYS_TMR1_RST	3	Timer1 reset
E_SYS_TMR2_RST	4	Timer2 reset
E_SYS_TMR3_RST	5	Timer3 reset
E_SYS_I2C0_RST	8	I2C0 reset
E_SYS_I2C1_RST	9	I2C1 reset
E_SYS_SPI0_RST	12	SPI0 reset
E_SYS_SPI1_RST	13	SPI1 reset
E_SYS_SPI2_RST	14	SPI2 reset
E_SYS_SPI3_RST	15	SPI3 reset
E_SYS_UART0_RST	16	UART0 reset
E_SYS_UART1_RST	17	UART1 reset
E_SYS_UART2_RST	18	UART2 reset
E_SYS_PWM03_RST	20	PWM0~3 reset
E_SYS_PWM47_RST	21	PWM4~7 reset
E_SYS_ACMP_RST	22	Analog Comparator reset
E_SYS_PS2_RST	23	PS2 reset
E_SYS_CAN0_RST	24	CAN0 reset
E_SYS_USBD_RST	27	USB device reset
E_SYS_ADC_RST	28	ADC reset
E_SYS_I2S_RST	29	I2S reset
E_SYS_PDMA_RST	32	PDMA reset
E_SYS_EBI_RST	33	EBI reset

E_SYS_IP_CLK

Enumeration identifier	Value	Description
E_SYS_WDT_CLK	0	Watch Dog Timer clock enable control
E_SYS_RTC_CLK	1	RTC clock enable control
E_SYS_TMR0_CLK	2	Timer0 clock enable control
E_SYS_TMR1_CLK	3	Timer1 clock enable control
E_SYS_TMR2_CLK	4	Timer2 clock enable control
E_SYS_TMR3_CLK	5	Timer3 clock enable control
E_SYS_FDIV_CLK	6	Clock Divider clock enable control
E_SYS_I2C0_CLK	8	I2C0 clock enable control

E_SYS_I2C1_CLK	9	I2C1 clock enable control
E_SYS_SPI0_CLK	12	SPI0 clock enable control
E_SYS_SPI1_CLK	13	SPI1 clock enable control
E_SYS_SPI2_CLK	14	SPI2 clock enable control
E_SYS_SPI3_CLK	15	SPI3 clock enable control
E_SYS_UART0_CLK	16	UART0 clock enable control
E_SYS_UART1_CLK	17	UART1 clock enable control
E_SYS_UART2_CLK	18	UART2 clock enable control
E_SYS_PWM01_CLK	20	PWM01 clock enable control
E_SYS_PWM23_CLK	21	PWM23 clock enable control
E_SYS_PWM45_CLK	22	PWM45 clock enable control
E_SYS_PWM67_CLK	23	PWM67 clock enable control
E_SYS_CAN0_CLK	24	CAN0 clock enable control
E_SYS_USBD_CLK	27	USB device clock enable control
E_SYS_ADC_CLK	28	ADC clock enable control
E_SYS_I2S_CLK	29	I2S clock enable control
E_SYS_ACMP_CLK	30	Analog Comparator clock enable control
E_SYS_PS2_CLK	31	PS2 clock enable control
E_SYS_PDMA_CLK	33	PDMA clock enable control
E_SYS_ISP_CLK	34	Flash ISP controller clock enable control
E_SYS_EBI_CLK	35	EBI clock enable control

E_SYS_PLL_CLKSRC

Enumeration identifier	Value	Description
E_SYS_EXTERNAL_12M	0	PLL source clock is from external 12MHz
E_SYS_INTERNAL_22M	1	PLL source clock is from internal 22MHz

E_SYS_IP_DIV

Enumeration identifier	Value	Description
E_SYS_ADC_DIV	0	ADC source clock divider setting
E_SYS_UART_DIV	1	UART source clock divider setting
E_SYS_USB_DIV	2	USB source clock divider setting
E_SYS_HCLK_DIV	3	HCLK source clock divider setting

E_SYS_IP_CLKSRC

Enumeration identifier	Value	Description
E_SYS_WDT_CLKSRC	0	Watch Dog Timer clock source setting
E_SYS_ADC_CLKSRC	1	ADC clock source setting
E_SYS_TMR0_CLKSRC	2	Timer0 clock source setting
E_SYS_TMR1_CLKSRC	3	Timer1 clock source setting

E_SYS_TMR2_CLKSRC	4	Timer2 clock source setting
E_SYS_TMR3_CLKSRC	5	Timer3 clock source setting
E_SYS_UART_CLKSRC	6	UART clock source setting
E_SYS_PWM01_CLKSRC	7	PWM01 clock source setting
E_SYS_PWM23_CLKSRC	8	PWM23 clock source setting
E_SYS_I2S_CLKSRC	9	I2S clock source setting
E_SYS_FRQDIV_CLKSRC	10	Frequency divider output clock source setting
E_SYS_PWM45_CLKSRC	11	PWM45 clock source setting
E_SYS_PWM67_CLKSRC	12	PWM67 clock source setting

E_SYS_CHIP_CLKSRC

Enumeration identifier	Value	Description
E_SYS_XTL12M	0	Select External 12M Crystal
E_SYS_XTL32K	1	Select External 32K Crystal
E_SYS_OSC22M	2	Select Internal 22M Oscillator
E_SYS_OSC10K	3	Select Internal 10K Oscillator
E_SYS_PLL	4	Select PLL clock

E_SYS_PD_TYPE

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

2.4. Functions

DrvSYS_ReadProductID

Prototype

uint32_t DrvSYS_ReadProductID (void);

Description

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

Product Device ID

Example

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

DrvSYS_GetResetSource

Prototype

uint32_t DrvSYS_GetResetSource (void);

Description

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

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The value in RSTSRC register.

Example

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

DrvSYS_ClearResetSource

Prototype

uint32_t DrvSYS_ClearResetSource (uint32_t u32Src);

Description

Clear reset source by writing a '1'.

Parameter

u32Src [in]

The corresponding bit of reset source.

Include

Driver/DrvSYS.h

Return Value

0 Succeed

Example

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

DrvSYS_ResetIP

Prototype

void DrvSYS_ResetIP (E_SYS_IP_RST eIpRst);

Description

To reset IP include GPIO, Timer0, Timer1, Timer2, Timer3, I2C0, I2C1, SPI0, SPI1, SPI2, SPI3, UART0, UART1, UART2, PWM03, PWM47, ACMP, PS2, CAN0, USBD, ADC, I2S, PDMA, and EBI.

Note

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

Parameter

eIpRst [in]

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

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_ResetIP (E_SYS_I2C0_RST); /* Reset I2C0 */

DrvSYS_ResetIP (E_SYS_SPI0_RST); /* Reset SPI0 */

DrvSYS_ResetIP (E_SYS_UART0_RST); /* Reset UART0 */

DrvSYS_ResetCPU

Prototype

void DrvSYS_ResetCPU (void);

Description

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

Note

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_ResetChip

Prototype

void DrvSYS_ResetChip (void);

Description

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

Note

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_ResetChip ( );   /* Reset whole chip */

DrvSYS_SelectBODVolt

Prototype

void   DrvSYS_SelectBODVolt (uint8_t u8Volt);

Description

To select Brown-Out threshold voltage.

Note

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

Parameter

u8Volt [in]

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

Include

Driver/DrvSYS.h

Return Value

None.

Example

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

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

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

DrvSYS_SetBODFunction

Prototype

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

Description

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

Note

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

Parameter

i32Enable [in]

1: enable, 0: disable

i32Flag [in]

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

bodcallbackFn [in]

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

Include

Driver/DrvSYS.h

Return Value

None

Example

/* Enable Brown-Out Detector , select Brown-Out Interrupt function and install callback function 'BOD_CallbackFn' */

DrvSYS_SetBODFunction (1, 0, BOD_CallbackFn);

/* Enable Brown-Out Detector and select Brown-Out reset function */

DrvSYS_SetBODFunction (1, 1, NULL);

/* Disable Brown-Out Detector */

DrvSYS_SetBODFunction (0, 0, NULL);

DrvSYS_EnableBODLowPowerMode

Prototype

void DrvSYS_EnableBODLowPowerMode (void);

Description

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

Note

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_DisableBODLowPowerMode

Prototype

void   DrvSYS_DisableBODLowPowerMode (void);

Description

To disable Brown-out Detector low power mode.

Note

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_EnableLowVoltReset

Prototype

void   DrvSYS_EnableLowVoltReset (void);

Description

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

Note

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_DisableLowVoltReset

Prototype

void   DrvSYS_DisableLowVoltReset (void);

Description

To disable low voltage reset function.

Note

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_GetBODState

Prototype

uint32_t DrvSYS_GetBODState (void);

Description

To get Brown-out Detector state.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

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

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

Example

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

DrvSYS_EnableTemperatureSensor

Prototype

```
void    DrvSYS_EnableTemperatureSensor (void);
```

Description

To enable temperature sensor function.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_EnableTemperatureSensor ( );    /* Enable temperature sensor function */
```

DrvSYS_DisableTemperatureSensor

Prototype

```
void    DrvSYS_DisableTemperatureSensor (void);
```

Description

To disable temperature sensor function.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

DrvSYS_DisableTemperatureSensor ( ); /* Disable temperature sensor function */

DrvSYS_UnlockProtectedReg

Prototype

int32_t DrvSYS_UnlockProtectedReg (void);

Description

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

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

Example

int32_t i32ret;

/* Unlock protected registers */

i32ret = DrvSYS_UnlockProtectedReg ( );

DrvSYS_LockProtectedReg

Prototype

int32_t DrvSYS_LockProtectedReg (void);

Description

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

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

Example

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

DrvSYS_IsProtectedRegLocked

Prototype

int32_t DrvSYS_IsProtectedRegLocked (void);

Description

To check the protected registers are locked or not.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

1: The Protected Registers are unlocked.

0: The Protected Registers are locked.

Example

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

else

/* do something for lock */

DrvSYS_EnablePOR

Prototype

void DrvSYS_EnablePOR (void);

Description

To re-enable power-on-reset control.

Note

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

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_DisablePOR

Prototype

void DrvSYS_DisablePOR (void);

Description

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

Note

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

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_SetIPClock

Prototype

void DrvSYS_SetIPClock (E_SYS_IP_CLK eIpClk, int32_t i32Enable);

Description

To enable or disable IP clock include Watch Dog TImer, RTC, Timer0, Timer1, Timer2, Timer3, I2C0, I2C1, SPI0, SPI1, SPI2, SPI3, UART0, UART1, UART2, PWM01, PWM23, PWM45, PWM67, CAN0, USBD, ADC, I2S, ACMP, PS2, PDMA, EBI, Flash ISP controller and Frequency Divider Output.

Note

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

Parameter

eIpClk [in]

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

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

None

Example

```
DrvSYS_SetIPClock (E_SYS_I2C0_CLK, 1);     /* Enable I2C0 engine clock */
DrvSYS_SetIPClock (E_SYS_I2C0_CLK, 0);     /* Disable I2C0 engine clock */
DrvSYS_SetIPClock (E_SYS_SPI0_CLK, 1);     /* Enable SPI0 engine clock */
DrvSYS_SetIPClock (E_SYS_SPI0_CLK, 0);     /* Disable SPI0 engine clock */
DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 1);     /* Enable TIMER0 engine clock */
```

DrvSYS_SetIPClock (E_SYS_TMR0_CLK, 0);   /* Disable TIMER0 engine clock */

DrvSYS_SelectHCLKSource

Prototype

int32_t DrvSYS_SelectHCLKSource (uint8_t u8ClkSrcSel);

Description

To select HCLK clock source from external 12M crystal clock, external 32K crystal clock, PLL clock, internal 10K oscillator clock, or internal 22M oscillator clock. Please refer to the Clock Diagram for HCLK usage in details.

Note

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

Parameter

u8ClkSrcSel [in]

0: External 12M clock

1: External 32K clock

2: PLL clock

3: Internal 10K clock

7: Internal 22M clock

Include

Driver/DrvSYS.h

Return Value

0 Succeed

< 0 Incorrect parameter

Example

DrvSYS_SelectHCLKSource (0);   /* Change HCLK clock source to be external 12M */

DrvSYS_SelectHCLKSource (2);   /* Change HCLK clock source to be PLL */

DrvSYS_SelectSysTickSource

Prototype

int32_t DrvSYS_SelectSysTickSource (uint8_t u8ClkSrcSel);

Description

To select Cortex-M0 SysTick clock source from external 12M crystal clock, external 32K crystal clock, external 12M crystal clock/2, HCLK/2, or internal 22M oscillator clock/2. The SysTick timer is a standard timer included by Cortex-M0.

Note

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

Parameter

u8ClkSrcSel [in]

0: External 12M clock

1: External 32K clock

2: External 12M clock / 2

3: HCLK / 2

7: Internal 22M clock / 2

Include

Driver/DrvSYS.h

Return Value

0 Succeed

< 0 Incorrect parameter

Example

```
DrvSYS_SelectSysTickSource (0);     /* Change SysTick clock source to be external 12M */
DrvSYS_SelectSysTickSource (3);     /* Change SysTick clock source to be HCLK / 2 */
```

DrvSYS_SelectIPClockSource

Prototype

int32_t DrvSYS_SelectIPClockSource (E_SYS_IP_CLKSRC eIpClkSrc, uint8_t u8ClkSrcSel);

Description

To select IP clock source include Watch Dog Timer, ADC, Timer 0~3, UART, PWM01, PWM23, PWM45, PWM67, I2S and Frequency Divider Output. Please refer to the Clock Diagram for IP clock source. The settings of IP's corresponding clock source are listed in Registers 'CLKSEL1' and 'CLKSEL2' of TRM in details.

Note

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

Parameter

eIpClkSrc [in]

E_SYS_WDT_CLKSRC / E_SYS_ADC_CLKSRC / E_SYS_TMR0_CLKSRC
E_SYS_TMR1_CLKSRC / E_SYS_TMR2_CLKSRC / E_SYS_TMR3_CLKSRC
E_SYS_UART_CLKSRC / E_SYS_PWM01_CLKSRC / E_SYS_PWM23_CLKSRC
E_SYS_PWM45_CLKSRC / E_SYS_PWM67_CLKSRC / E_SYS_FRQDIV_CLKSRC
E_SYS_I2S_CLKSRC.

u8ClkSrcSel [in]

IP's corresponding clock source.

u8ClkSrcSel	0	1	2	3	7
Watch Dog Timer	Reserved	Ext. 32K (*)	HCLK/2048	Internal 10K	X
ADC	External 12M	PLL	HCLK (*)	Internal 22M	X
Timer	External 12M	External 32K	HCLK	Reserved	Internal 22M
UART	External 12M	PLL	Reserved	Internal 22M	X
PWM	External 12M	External 32K	HCLK	Internal 22M	X
Frequency Divider Output	External 12M	External 32K	HCLK	Internal 22M	X
I2S	External 12M	PLL	HCLK	Internal 22M	X

Note (*)

Only NuMicro™ NUC1x0xxxCx Series (Ex. NUC140VE3CN) support External 32 KHz Crystal as Watch Dog Timer clock source and HCLK as ADC clock source. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Include

Driver/DrvSYS.h

Return Value

0 Succeed

< 0 Incorrect parameter

Example

```
/* Select ADC clock source from 12M */
DrvSYS_SelectIPClockSource (E_SYS_ADC_CLKSRC, 0x00);
/* Select TIMER0 clock source from HCLK */
DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0x02);
/* Select I2S clock source from HCLK */
DrvSYS_SelectIPClockSource (E_SYS_I2S_CLKSRC, 0x02);
```

DrvSYS_SetClockDivider

Prototype

int32_t DrvSYS_SetClockDivider (E_SYS_IP_DIV eIpDiv, int32_t i32value);

Description

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

The IP clock frequency is calculated by:

IP clock source frequency / (i32value + 1).

Parameter

eIpDiv [in]

E_SYS_ADC_DIV / E_SYS_UART_DIV / E_SYS_USB_DIV / E_SYS_HCLK_DIV

i32value [in]

Divide number.

HCLK, USB, UART: 0~15

ADC: 0~255

Include

Driver/DrvSYS.h

Return Value

0 Succeed

< 0 Incorrect parameter

Example

```
/* Set ADC clock divide number 0x01; ADC clock = ADC source clock / (1+1) */
DrvSYS_SetClockDivider (E_SYS_ADC_DIV, 0x01);
/* Set UART clock divide number 0x02; UART clock = UART source clock / (2+1) */
DrvSYS_SetClockDivider (E_SYS_UART_DIV, 0x02);
/* Set HCLK clock divide number 0x03; HCLK clock = HCLK source clock / (3+1) */
DrvSYS_SetIPClockSource (E_SYS_HCLK_DIV, 0x03);
```

DrvSYS_SetOscCtrl

Prototype

int32_t DrvSYS_SetOscCtrl (E_SYS_CHIP_CLKSRC eClkSrc, int32_t i32Enable);

Description

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

Note

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

Parameter

eOscCtrl [in]

E_SYS_XTL12M / E_SYS_XTL32K / E_SYS_OSC22M / E_SYS_OSC10K.

i32Enable [in]

1: enable, 0: disable

Include

Driver/DrvSYS.h

Return Value

0 Succeed

< 0 Incorrect parameter

Example

```
DrvSYS_SetOscCtrl (E_SYS_XTL12M, 1);        /* Enable external 12M */
DrvSYS_SetOscCtrl (E_SYS_XTL12M, 0);        /* Disable external 12M */
```

DrvSYS_SetPowerDownWakeUpInt

Prototype

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

Description

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

Note

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

Parameter

i32Enable [in]

1: enable, 0: disable

pdwucallbackFn [in]

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

i32enWUDelay [in]

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

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_SetPowerDownWakeUpInt (1, PWRWU_CallbackFn, 1);

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

DrvSYS_SetPowerDownWakeUpInt (0, NULL, 0);

DrvSYS_EnterPowerDown

Prototype

void DrvSYS_EnterPowerDown (E_SYS_PD_TYPE ePDType);

Description

To enter system power down mode immediately or after CPU enters sleep mode. When chip enters power down mode, the LDO, 12M crystal, and 22M oscillator will be disabled. Please refer to Application Note, *AN_1007_EN_Power_Management*, for application.

Note

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

Parameter

ePDType [in]

E_SYS_IMMEDIATE: Chip enters power down mode immediately.

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

Include

Driver/DrvSYS.h

Return Value

None

Example

/* Chip enter power mode immediately */

DrvSYS_EnterPowerDown (E_SYS_ IMMEDIATE);

/* Wait for CPU enters sleep mode, then Chip enter power mode */

DrvSYS_EnterPowerDown (E_SYS_WAIT_FOR_CPU);

DrvSYS_SelectPLLSource

Prototype

void DrvSYS_SelectPLLSource (E_SYS_PLL_CLKSRC ePllSrc);

Description

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

Parameter

ePllSrc [in]

E_SYS_EXTERNAL_12M / E_SYS_INTERNAL_22M

Include

Driver/DrvSYS.h

Return Value

None

Example

/* Select PLL clock source from 12M */

DrvSYS_ SelectPLLSource (E_SYS_EXTERNAL_12M);

/* Select PLL clock source from 22M */

DrvSYS_ SelectPLLSource (E_SYS_INTERNAL_22M);

DrvSYS_SetPLLMode

Prototype

void DrvSYS_SetPLLMode (int32_t i32Flag);

Description

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

Parameter

i32Flag [in]

1: PLL is in power down mode.

0: PLL is in normal mode.

Include

Driver/DrvSYS.h

Return Value

None

Example

/* Enable PLL power down mode, PLL operates in power down mode */

DrvSYS_SetPLLMode (1);

/* Disable PLL power down mode, PLL operates in normal mode */

DrvSYS_SetPLLMode (0);

DrvSYS_GetExtClockFreq

Prototype

uint32_t DrvSYS_GetExtClockFreq (void);

Description

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The external crystal clock frequency

Example

uint32_t u32clock;

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

DrvSYS_GetPLLContent

Prototype

uint32_t   DrvSYS_GetPLLContent(E_SYS_PLL_CLKSRC ePllSrc, uint32_t u32PllClk);

Description

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

Parameter

ePllSrc   [in]

E_SYS_EXTERNAL_12M / E_SYS_INTERNAL_22M

u32PllClk [in]

The target PLL clock frequency. The unit is in Hz. The range of u32PllClk is 25MHz~200MHz.

Include

Driver/DrvSYS.h

Return Value

The PLL control register setting.

Example

```
uint32_t u32PllCr;
/* Get PLL control register setting for target PLL clock 50MHz */
u32PllCr = DrvSYS_GetPLLContent (E_SYS_EXTERNAL_12M, 50000000);
```

DrvSYS_SetPLLContent

Prototype

void   DrvSYS_SetPLLContent (uint32_t u32PllContent);

Description

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

Parameter

u32PllContent [in]

The PLL register setting for the target PLL clock frequency.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
uint32_t u32PllCr;
/* Get PLL control register setting for target PLL clock 50MHz */
u32PllCr = DrvSYS_GetPLLContent (E_DRVSYS_EXTERNAL_12M, 50000000);
/* Set PLL control register setting to get nearest PLL clock */
DrvSYS_SetPLLContent (u32PllCr);
```

DrvSYS_GetPLLClockFreq

Prototype

uint32_t DrvSYS_GetPLLClockFreq (void);

Description

To get PLL clock output frequency.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The PLL clock output frequency in Hz

Example

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

DrvSYS_GetHCLKFreq

Prototype

uint32_t DrvSYS_GetHCLKFreq (void);

Description

To get HCLK clock frequency.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

The HCLK clock frequency in Hz

Example

```
uint32_t u32clock;
u32clock = DrvSYS_GetHCLKFreq ( );      /* Get current HCLK clock */
```

DrvSYS_Open

Prototype

int32_t DrvSYS_Open (uint32_t u32Hclk);

Description

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

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

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

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

Note

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

Parameter

u32Hclk [in]

The target HCLK clock frequency. The unit is in Hz. The range of u32Hclk is 25MHz~50MHz.

Include

Driver/DrvSYS.h

Return Value

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

Example

```
/* Set PLL clock 50MHz, and switch HCLK source clock to PLL */
DrvSYS_Open (50000000);
```

DrvSYS_SetFreqDividerOutput

Prototype

```
int32_t    DrvSYS_SetFreqDividerOutput (int32_t i32Flag, uint8_t u8Divider);
```

Description

NUC100 Series support to monitor clock source frequency by CLKO output pin. This function is used to enable or disable frequency clock output and set its divider number. The formula of output frequency is $F_{out} = \frac{F_{in}}{2^{N+1}}$, where F_{in} is the input clock frequency, F_{out} is the frequency of divider output clock, and N is a 4-bit value.

To monitor the clock source frequency, we can use this function to enable clock output function. However, we still need to set CLKO as output pin by GPIO multi-function selection to output the clock to output pin of NUC100 series.

Parameter

i32Flag [in]

1: enable; 0: disable.

u8Divider [in]

The divider number of output frequency. The value is 0~15.

Include

Driver/DrvSYS.h

Return Value

0	Succeed
<0	Incorrect parameter

Example

```
/* Enable frequency clock output and set its divide number 2,
The output frequency = input clock / 2^(2+1) */
DrvSYS_SetFreqDividerOutput (1, 2);
/* Disable frequency clock output */
DrvSYS_SetFreqDividerOutput (0, 0);
```

DrvSYS_EnableHighPerformanceMode

Prototype

void DrvSYS_EnableHighPerformanceMode (void);

Description

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

Note 1

Only Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Note 2

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

/* Enable high performance mode */

DrvSYS_EnableHighPerformanceMode ( );

DrvSYS_DisableHighPerformanceMode

Prototype

void DrvSYS_DisableHighPerformanceMode (void);

Description

To disable chip high performance mode.

Note 1

Only Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Note 2

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

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

/* Disable high performance mode */

DrvSYS_DisableHighPerformanceMode ( );

DrvSYS_Delay

Prototype

void DrvSYS_Delay (uint32_t us);

Description

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

Parameter

us [in]

Delay time. The maximal delay time is 335000 us.

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_GetChipClockSourceStatus

Prototype

int32_t DrvSYS_GetChipClockSourceStatus (E_SYS_CHIP_CLKSRC eClkSrc);

Description

To monitor if the chip clock source stable or not, include internal 10K, 22M oscillator, external 32K, 12M crystal, or PLL clock.

Note

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) and Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

eClkSrc [in]

E_SYS_XTL12M / E_SYS_XTL32K / E_SYS_OSC22M / E_SYS_OSC10K / E_SYS_PLL

Include

Driver/DrvSYS.h

Return Value

0 Clock source is not stable or not enabled

1 Clock source is stable

< 0 Incorrect parameter

Example

```
/* Enable external 12M */
DrvSYS_SetOscCtrl (E_SYS_XTL12M, 1);
/* Waiting for 12M Crystal stable */
```

```
while (DrvSYS_GetChipClockSourceStatus (E_SYS_XTL12M) != 1);
/* Disable PLL power down mode */
DrvSYS_SetPLLMode (0);
/* Waiting for PLL clock stable */
while (DrvSYS_GetChipClockSourceStatus (E_SYS_PLL) != 1);
```

DrvSYS_GetClockSwitchStatus

Prototype

uint32_t DrvSYS_GetClockSwitchStatus (void);

Description

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

Note

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) and Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

0: Clock switch success

1: Clock switch fail

Example

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

DrvSYS_ClearClockSwitchStatus

Prototype

void DrvSYS_ClearClockSwitchStatus (void);

Description

To clear the Clock Switch Fail Flag.

Note

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) and Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
uint32_t u32flag;
DrvSYS_SelectHCLKSource (0);      /* Change HCLK clock source to be external 12M */
u32flag = DrvSYS_GetClockSwitchStatus ( );      /* Get clock switch fail flag */
if (u32flag)
    DrvSYS_ClearClockSwitchStatus ( );     /* Clear clock switch fail flag */
```

DrvSYS_GetVersion

Prototype

uint32_t DrvSYS_GetVersion (void);

Description

Get this version of DrvSYS driver.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

Version number:

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

3. UART Driver

3.1. UART Introduction

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

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

3.2. UART Feature

The UART includes following features:

- 64 bytes(UART0)/16 bytes(UART1,UART2) entry FIFOs for received and transmitted data payloads
- Auto flow control/flow control function (CTS, RTS) are supported.
- Fully programmable serial-interface characteristics:
  -- 5-, 6-, 7-, or 8-bit character
  -- Even, odd, or no-parity bit generation and detection
  -- 1-, 1&1/2, or 2-stop bit generation
  -- Baud rate generation
  -- False start bit detection.
- Full-prioritized interrupt system controls
- Loop back mode for internal diagnostic testing
- Support IrDA SIR Function
- Support LIN (Local interconnect network) master mode.
- Programmable baud-rate generator that allows the clock to be divided by programmable divider

3.3. Constant Definition

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

3.4. Type Definition

E_UART_PORT

Enumeration identifier	Value	Description
UART_PORT0	0x000	UART port 0
UART_PORT1	0x100000	UART port 1
UART_PORT2	0x104000	UART port 2

E_INT_SOURCE

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

E_DATABITS_SETTINGS

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

E_PARITY_SETTINGS

Enumeration identifier	Value	Description
DRVUART_PARITY_NONE	0x0	None parity

DRVUART_PARITY_ODD	0x1	Odd parity enable
DRVUART_PARITY_EVEN	0x3	Even parity enable
DRVUART_PARITY_MARK	0x5	Parity mask
DRVUART_PARITY_SPACE	0x7	Parity space

E_STOPBITS_SETTINGS

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

E_FIFO_SETTINGS

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

E_UART_FUNC

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

E_MODE_RS485

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

3.5. Macros

_DRVUART_SENDBYTE

Prototype

void _DRVUART_SENDBYTE (u32Port, byData);

Description

Send 1 byte data from UART.

Include

Driver/DrvUART.h

Return Value

None.

Example

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

_DRVUART_RECEIVEBYTE

Prototype

uint8_t _DRVUART_RECEIVEBYTE (u32Port);

Description

Receive 1 byte data from specified UART FIFO.

Include

Driver/DrvUART.h

Return Value

One byte data.

Example

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

_DRVUART_SET_DIVIDER

Prototype

void _DRVUART_SET_DIVIDER (u32Port, u16Divider);

Description

To set the UART divider to control UART baud-rate

Include

Driver/DrvUART.h

Return Value

None.

Example

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

_DRVUART_RECEIVEAVAILABLE

Prototype

int8_t _DRVUART_RECEIVEAVAILABLE (u32Port);

Description

To get current Rx FIFO pointer

Include

Driver/DrvUART.h

Return Value

Rx FIFO pointer value.

Example

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

_DRVUART_WAIT_TX_EMPTY

Prototype

void _DRVUART_WAIT_TX_EMPTY (u32Port);

Description

Polling Tx empty flag to check Tx FIFO is empty.

Include

Driver/DrvUART.h

Return Value

None.

Example

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

3.6. Functions

DrvUART_Open

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

sParam [in]

Specify the property of UART. It includes

u32BaudRate: Baud rate (Hz)

u8cParity: NONE/EVEN/ODD parity

It could be

DRVUART_PARITY_NONE (None parity).

DRVUART_PARITY_EVEN (Even parity)

DRVUART_PARITY_ODD(Odd parity).

u8cDataBits: data bit setting

It could be

DRVUART_DATA_BITS_5 (5 data bits).

DRVUART_DATA_BITS_6 (6 data bits)

DRVUART_DATA_BITS_7 (7 data bits).

DRVUART_DATA_BITS_8 (8 data bits).

u8cStopBits: stop bits setting

It could be

DRVUART_STOPBITS_1 (1 stop bit).

DRVUART_STOPBITS_1_5 (1.5 stop bit)

DRVUART_STOPBITS_2 (2 stop bits).

u8cRxTriggerLevel: Rx FIFO interrupt trigger Level

LEVEL_X_BYTE means the trigger level of UART channel is X bytes

It could be

DRVUART_FIFO_1BYTE, DRVUART_FIFO_4BYTES

DRVUART_FIFO_8BYTES, DRVUART_FIFO_14BYTES

DRVUART_FIFO_30BYTES, DRVUART_FIFO_46BYTES

DRVUART_FIFO_62BYTES

In UART0 , it could be LEVEL_1_BYTE to LEVEL_62_BYTES.

Others, it could be LEVEL_1_BYTE to LEVEL_14_BYTES.

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

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success.

E_DRVUART_ERR_PORT_INVALID: Wrong UART port configure

E_DRVUART_ERR_PARITY_INVALID: Wrong party setting

E_DRVUART_ERR_DATA_BITS_INVALID: Wrong Data bit setting

E_DRVUART_ERR_STOP_BITS_INVALID: Wrong Stop bit setting

E_DRVUART_ERR_TRIGGERLEVEL_INVALID: Wrong trigger level setting

Example

```
/* Set UART0 under 115200bps, 8 data bits ,1 stop bit and none parity and 1 byte Rx trigger level settings. */
STR_UART_T sParam;
sParam.u32BaudRate          = 115200;
sParam.u8cDataBits          = DRVUART_DATABITS_8;
sParam.u8cStopBits          = DRVUART_STOPBITS_1;
```

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

DrvUART_Close

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

Include

Driver/DrvUART.h

Return Value

None

Example

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

DrvUART_EnableInt

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

pfncallback [in]

Call back function pointer

Include

Driver/DrvUART.h

Return Value

None

Note

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

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

Example

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

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

DrvUART_DisableInt

Prototype

void DrvUART_DisableInt (

E_UART_PORT u32Port

uint32_t u32InterruptFlag

);

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt and Time-out Interrupt

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

None

Note

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

Example

/* To disable the THRE interrupt enable flag. */

DrvUART_DisableInt (UART_PORT0, DRVUART_THREINT);

DrvUART_ClearIntFlag

Prototype

uint32_t

DrvUART_ClearIntFlag (

E_UART_PORT u32Port

uint32_t u32InterruptFlag

);

Description

The function is used to clear UART specified interrupt flag.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt.

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

E_SUCESS Success

Example

```
/* To clear UART0 LIN break interrupt flag */
DrvUART_ClearIntFlag (UART_PORT0, DRVUART_LININT);
```

DrvUART_GetIntStatus

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

u32InterruptFlag [in]

DRVUART_LININT : LIN RX Break Field Detected Interrupt Enable

DRVUART_BUFERRINT : Buffer Error Interrupt Enable

DRVUART_WAKEINT : Wakeup Interrupt.

DRVUART_MOSINT : MODEM Status Interrupt.

DRVUART_RLSNT : Receive Line Status Interrupt.

DRVUART_THREINT : Transmit Holding Register Empty Interrupt.

DRVUART_RDAINT : Receive Data Available Interrupt.

DRVUART_TOUTINT : Time-out Interrupt.

Include

Driver/DrvUART.h

Return Value

0: The specified interrupt did not happen.

1: The specified interrupt happened.

E_DRVUART_ARGUMENT: Error Parameter.

Note

It is recommended to poll one interrupt at a time.

Example

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

DrvUART_GetCTSInfo

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1 (UART_PORT2 is no supported.)

pu8CTSValue [out]

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

pu8CTSChangeState [out]

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

Include

Driver/DrvUART.h

Return Value

None

Example

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

uint8_t u8CTS_value, u8CTS_state;

DrvUART_GetCTSInfo(UART_PORT1,& u8CTS_value,& u8CTS_state);

DrvUART_SetRTS

Prototype

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

Description

The function is used to set RTS setting.

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1 (UART_PORT2 is no supported.)

u8Value [in]

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

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

u16TriggerLevel [in]

RTS Trigger Level :DRVUART_FIFO_1BYTES to DRVUART_FIFO_62BYTES

Include

Driver/DrvUART.h

Return Value

None

Example

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

DrvUART_Read

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

pu8RxBuf [out]

Specify the buffer to receive the data of receive FIFO.

u32ReadBytes [in]

Specify the read bytes number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success.

E_DRVUART_TIMEOUT: FIFO polling timeout.

Example

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

DrvUART_Write

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

pu8TxBuf [in]

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

u32WriteBytes [in]

Specify the byte number of data.

Include

Driver/DrvUART.h

Return Value

E_SUCCESS: Success

E_DRVUART_TIMEOUT: FIFO polling timeout

Example

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

DrvUART_EnablePDMA

Prototype

```
void
DrvUART_EnablePDMA (
    E_UART_PORT u32Port
    );
```

Description

The function is used to control enable PDMA transmit/receive channel

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1 (UART_PORT2 is no supported.)

Include

Driver/DrvUART.h

Return Value

None.

Example

```
/* Enable TX and RX PDMA in UART 1 */
DrvUART_EnablePDMA(UART_PORT1);
```

DrvUART_DisablePDMA

Prototype

```
void
DrvUART_DisablePDMA (
    E_UART_PORT u32Port
);
```

Description

The function is used to control disable PDMA transmit/receive channel

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1 (UART_PORT2 is no supported.)

Include

Driver/DrvUART.h

Return Value

None.

Example

/* Disable Tx and Rx PDMA in UART 1 */

DrvUART_DisablePDMA(UART_PORT1);

DrvUART_SetFnIRDA

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

str_IRCR [in]

The structure of IrDA

It includes of

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

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

u8cInvTX : Invert Tx signal function TRUE or FASLE

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

Include

Driver/DrvUART.h

Return Value

None

Note

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

Example

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

DrvUART_SetFnRS485

Prototype

```
void
DrvUART_OpenRS485 (
    E_UART_PORT u32Port,
    STR_RS485_T *str_RS485
);
```

Description

The function is used to set RS485 relative setting

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

str_RS485 [in]

The structure of RS485

It includes of

u8cModeSelect: Select operation mode

MODE_RS485_NMM: RS-485 Normal Multi-drop Mode

MODE_RS485_AAD: RS-485 Auto Address Detection Mode

MODE_RS485_AUD: RS-485 Auto Direction Mode

u8cAddrEnable: Enable or Disable RS-485 Address Detection

u8cAddrValue: Set Address match value

u8cDelayTime: Set transmit delay time value

u8cRxDisable: Enable or Disable receiver function.

Include

Driver/DrvUART.h

Return Value

None

Note

None

Example

```
/* Condition: Change UART1 to RS485 function. Set relative setting as below.*/
STR_RS485_T sParam_RS485;
sParam_RS485.u8cAddrEnable      = ENABLE;
sParam_RS485.u8cAddrValue       = 0xC0;          /* Address */
sParam_RS485.u8cModeSelect      = MODE_RS485_AAD|MODE_RS485_AUD;
sParam_RS485.u8cDelayTime       = 0;
sParam_RS485.u8cRxDisable       = TRUE;
DrvUART_SetFnRS485(UART_PORT1,&sParam_RS485);
```

DrvUART_SetFnLIN

Prototype

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

Description

The function is used to set LIN relative setting

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/UART_PORT2

u16Mode [in]

Specify LIN direction : MODE_TX and/or MODE_RX

u16BreakLength [in]

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

Include

Driver/DrvUART.h

Return Value

None

Example

/* Change UART1 to LIN function and set to transmit the header information. */

DrvUART_SetFnLIN(uart_ch,MODE_TX | MODE_RX,13);

DrvUART_GetVersion

Prototype

int32_t

DrvUART_GetVersion (void);

Description

Return the current version number of driver.

Include

Driver/DrvUART.h

Return Value

Version number:

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

4. TIMER/WDT Driver

4.1. TIMER/WDT Introduction

The timer module includes four channels, TIMER0~TIMER3, which allow you to easily implement a counting scheme for use. The timer can perform functions like frequency measurement, event counting, interval measurement, clock generation, delay timing, and so on. The timer can generate an interrupt signal upon timeout, or provide the current value of count during operation. And for external count and capture functions, only NUC1x0xxxBx and NUC1x0xxxCx series supported, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

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

4.2. TIMER/WDT Feature

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

4.3. Type Definition

E_TIMER_CHANNEL

Enumeration Identifier	Value	Description
E_TMR0	0x0	Specify the timer channel - 0

E_TMR1	0x1	Specify the timer channel - 1
E_TMR2	0x2	Specify the timer channel - 2
E_TMR3	0x3	Specify the timer channel - 3

E_TIMER_OPMODE

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

E_TIMER_TX_PHASE

Enumeration Identifier	Value	Description
E_PHASE_FALLING	0x0	Set falling edge of external count pin will be counted
E_PHASE_RISING	0x1	Set raising edge of external count pin will be counted

E_TIMER_TEX_EDGE

Enumeration Identifier	Value	Description
E_EDGE_FALLING	0x0	Set 1 to 0 transition on TEX will be detected
E_EDGE_RISING	0x1	Set 0 to 1 transition on TEX will be detected
E_EDGE_BOTH	0x2	Either 1 to 0 or 0 to 1 transition on TEX will be detected

E_TIMER_RSTCAP_MODE

Enumeration Identifier	Value	Description
E_CAPTURE	0x0	TEX transition is using as timer capture function
E_RESET	0x1	TEX transition is using as timer counter reset function

E_WDT_CMD

Enumeration Identifier	Value	Description
E_WDT_IOC_START_TIMER	0x0	Start WDT counting
E_WDT_IOC_STOP_TIMER	0x1	Stop WDT counting

E_WDT_IOC_ENABLE_INT	0x2	Enable WDT interrupt
E_WDT_IOC_DISABLE_INT	0x3	Disable WDT interrupt
E_WDT_IOC_ENABLE_WAKEUP	0x4	Enable WDT time-out wake up function
E_WDT_IOC_DISABLE_WAKEUP	0x5	Disable WDT time-out wake up function
E_WDT_IOC_RESET_TIMER	0x6	Reset WDT counter
E_WDT_IOC_ENABLE_RESET_FUNC	0x7	Enable WDT reset function when WDT time-out
E_WDT_IOC_DISABLE_RESET_FUNC	0x8	Disable WDT reset function when WDT time-out
E_WDT_IOC_SET_INTERVAL	0x9	Set the WDT time-out interval

E_WDT_INTERVAL

Enumeration Identifier	Value	Description
E_LEVEL0	0x0	Set WDT time-out interval is 2^4 WDT_CLK
E_LEVEL1	0x1	Set WDT time-out interval is 2^6 WDT_CLK
E_LEVEL2	0x2	Set WDT time-out interval is 2^8 WDT_CLK
E_LEVEL3	0x3	Set WDT time-out interval is 2^10 WDT_CLK
E_LEVEL4	0x4	Set WDT time-out interval is 2^12 WDT_CLK
E_LEVEL5	0x5	Set WDT time-out interval is 2^14 WDT_CLK
E_LEVEL6	0x6	Set WDT time-out interval is 2^16 WDT_CLK
E_LEVEL7	0x7	Set WDT time-out interval is 2^18 WDT_CLK

4.4. Functions

DrvTIMER_Init

Prototype

void DrvTIMER_Init (void)

Description

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

Parameter

None

Include

Driver/DrvTIMER.h

Return Value

None

Example:

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

DrvTIMER_Open

Prototype

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

Description

Open the specified timer channel with specified operation mode.

Parameter

ch [in]

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

uTicksPerSecond [in]

This value means how many timer interrupt ticks in one second

op_moode [in]

E_TIMER_OPMODE, E_ONESHOT_MODE / E_PERIODIC_MODE / E_TOGGLE_MODE / E_CONTINUOUS_MODE

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

E_DRVTIMER_CLOCK_RATE: Calculate initial value fail

Example

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

DrvTIMER_Close

Prototype

int32_t DrvTIMER_Close (E_TIMER_CHANNEL ch)

Description

The function is used to close the timer channel.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_SetTimerEvent

Prototype

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

Description

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

Parameter

ch [in]

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

uInterruptTicks [in]

Number of timer interrupt occurred

pTimerCallback [in]

The function pointer of the interrupt callback function

parameter [in]

A parameter of the callback function

Include

Driver/DrvTIMER.h

Return Value

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

Example

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

DrvTIMER_ClearTimerEvent

Prototype

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

Description

Clear the timer event of the specified timer channel.

Parameter

ch [in]

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

uTimerEventNo [in]

The timer event number

Include

Driver/DrvTIMER.h

Return Value

None

Example

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

DrvTIMER_EnableInt

Prototype

int32_t DrvTIMER_EnableInt (E_TIMER_CHANNEL ch)

Description

This function is used to enable the specified timer interrupt.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_DisableInt

Prototype

int32_t DrvTIMER_DisableInt (E_TIMER_CHANNEL ch)

Description

This function is used to disable the specified timer interrupt.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_GetIntFlag

Prototype

int32_t DrvTIMER_GetIntFlag (E_TIMER_CHANNEL ch)

Description

Get the interrupt flag status from the specified timer channel.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

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

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_ClearIntFlag

Prototype

int32_t DrvTIMER_ClearIntFlag (E_TIMER_CHANNEL ch)

Description

Clear the interrupt flag of the specified timer channel.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_Start

Prototype

int32_t DrvTIMER_Start (E_TIMER_CHANNEL ch)

Description

Start to count the specified timer channel.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_GetIntTicks

Prototype

uin32_t DrvTIMER_GetIntTicks (E_TIMER_CHANNEL ch)

Description

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

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

uTimerTick: Return the interrupt ticks

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_ResetIntTicks

Prototype

int32_t DrvTIMER_ResetIntTicks (E_TIMER_CHANNEL ch)

Description

This function is used to clear interrupt ticks to 0.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_Delay

Prototype

void DrvTIMER_Delay (E_TIMER_CHANNEL ch, uint32_t uIntTicks)

Description

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

Parameter

ch [in]

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

uIntTicks [in]

The delay ticks

Include

Driver/DrvTIMER.h

Return Value

None

Example

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

DrvTIMER_OpenCounter

Prototype

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

Description

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

Note

Only NUC1x0xxxBx and NUC1x0xxxCx series support this function, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

ch [in]

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

uCounterBoundary [in]

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

op_mode [in]

E_TIMER_OPMODE, it's included E_ONESHOT_MODE / E_PERIODIC_MODE / E_CONTINUOUS_MODE

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

E_DRVTIMER_EIO: Timer has not been initialized

Example

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

DrvTIMER_StartCounter

Prototype

in32_t DrvTIMER_StartCounter (E_TIMER_CHANNEL ch)

Description

Start counting of the specified timer channel.

Note

Only NUC1x0xxxBx and NUC1x0xxxCx series support this function, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicroTM NUC100 Series Products Selection Guide of Appendix in details.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

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

DrvTIMER_GetCounters

Prototype

uin32_t DrvTIMER_GetCounters (E_TIMER_CHANNEL ch)

Description

This function is used to get the current counters of the specified timer channel.
Only NUC1x0xxxBx and NUC1x0xxxCx series support this function, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicroTM NUC100 Series Products Selection Guide of Appendix in details.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

u32Counters: Return current counters

E_DRVTIMER_CHANNEL: Invalid timer channel

Example:

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

DrvTIMER_OpenCapture

Prototype

```
int32_t DrvTIMER_OpenCapture (
            E_TIMER_CHANNEL ch,
            E_TIMER_RSTCAP_MODE mode
  );
```

Description

This function is used to initial the external timer capture source and set to start catpure or reset specified timer counter.
The TIMER clock source should be set as HCLK.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

mode [in]

E_TIMER_RSTCAP_MODE,

E_CAPTURE : Run capture function
E_RESET : Reset counter value of specified timer channel

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Open external Timer-0 capture function */
DrvTIMER_OpenCapture (E_TMR0, E_CAPTURE);
```

DrvTIMER_CloseCapture

Prototype

int32_t DrvTIMER_CloseCapture (
E_TIMER_CHANNEL ch,
);

Description

This function is used to close the external timer capture source.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Close external Timer-0 capture function */
DrvTIMER_CloseCapture (E_TMR0);
```

DrvTIMER_SelectExternalMode

Prototype

int32_t DrvTIMER_SelectExternalMode (
E_TIMER_CHANNEL ch,
E_TIMER_RSTCAP_MODE mode
);

Description

This function is used to select to run capture function or reset the timer counter.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

mode [in]

E_TIMER_RSTCAP_MODE,

E_CAPTURE : Run capture function
E_RESET : Reset counter value of specified timer channel

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Select Timer-0 runs in capture function */
DrvTIMER_SelectExternalMode (E_TMR0, E_CAPTURE);
```

DrvTIMER_SelectCaptureEdge

Prototype

```
int32_t DrvTIMER_OpenCapture (
        E_TIMER_CHANNEL ch,
        E_TIMER_TEX_EDGE edge
);
```

Description

This function is used to configure the detect edge of timer capture mode.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

edge [in]

E_TIMER_TEX_EDGE,

E_EDGE_FALLING : 1 to 0 transition on TEX will be detected.
E_EDGE_RISING : 0 to 1 transition on TEX will be detected.
E_EDGE_BOTH : either 0 to 1 or 1 to 0 transition on TEX will be detected.

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

/* Configure timer-0 capture detect occurrd when 0 to 1 transition on external capture pin */
DrvTIMER_SelectCaptureEdge (E_TMR0, E_EDGE_RISING);

DrvTIMER_EnableCaptureInt

Prototype

```
int32_t DrvTIMER_EnableCaptureInt (
                    E_TIMER_CHANNEL ch,
    );
```

Description

This function is used to enable the timer external interrupt function.
If any transition on TEX pin and matched with the **E_TIMER_TEX_EDGE** settings, system will cause the external interrupt flag(TEXIF) to 1.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

/* Enable external timer-0 capture detect interrupt function */
DrvTIMER_EnableCaptureInt (E_TMR0);

DrvTIMER_DisableCaptureInt

Prototype

```
int32_t DrvTIMER_DisableCaptureInt (
                    E_TIMER_CHANNEL ch,
    );
```

Description

This function is used to disable the timer external interrupt function.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Disable external timer-0 capture detect interrupt function */
DrvTIMER_DisableCaptureInt (E_TMR0);
```

DrvTIMER_EnableCapture

Prototype

```
int32_t DrvTIMER_EnableCapture (
        E_TIMER_CHANNEL ch,
  );
```

Description

This function is used to enable the specified capture function.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Enable external timer-0 capture function */
DrvTIMER_EnableCapture (E_TMR0);
```

DrvTIMER_DisableCapture

Prototype

```
int32_t DrvTIMER_DisableCapture (
        E_TIMER_CHANNEL ch,
```

);

Description

This function is used to disable the specified capture function.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Disable external timer-0 capture function */
DrvTIMER_DisableCapture (E_TMR0);
```

DrvTIMER_GetCaptureData

Prototype

```
int32_t DrvTIMER_GetCaptureData (
                E_TIMER_CHANNEL ch,
    );
```

Description

This function is used to get the capture value of the specified timer channel.
And the return data is valid only if the capture interrupt flag set to 1 by H.W.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

Capture value: Return capture value

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Get the external timer-0 capture interrupt status */
uint32_t u32IntStatus, u32CurData;
u32IntStatus = DrvTIMER_GetCaptureIntFlag (E_TMR0);
if (u32IntStatus == 1)
{
    /* Get the current capture data from timer-0 */
    u32CurData = DrvTIMER_GetCaptureData (E_TMR0);
}
```

DrvTIMER_GetCaptureIntFlag

Prototype

```
int32_t DrvTIMER_GetCaptureIntFlag (
                E_TIMER_CHANNEL ch,
  );
```

Description

Get the external interrupt flag status from the specified timer channel.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

External interrupt flag: 0:No interrupt / 1:Interrupt occurred

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Get the external timer-0 capture interrupt status */
uint32_t u32IntStatus, u32CurData;
u32IntStatus = DrvTIMER_GetCaptureIntFlag (E_TMR0);
if (u32IntStatus == 1)
{
    /* Get the current capture data from timer-0 */
    u32CurData = DrvTIMER_GetCaptureData (E_TMR0);
}
```

DrvTIMER_ClearCaptureIntFlag

Prototype

```
int32_t DrvTIMER_GetCaptureIntFlag (
                    E_TIMER_CHANNEL ch,
    );
```

Description

Clear the external interrupt flag of the specified timer channel.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Get the external timer-0 capture interrupt status */
uint32_t u32IntStatus, u32CurData;
u32IntStatus = DrvTIMER_GetCaptureIntFlag (E_TMR0);
if (u32IntStatus == 1)
{
    /* Get the current capture data from timer-0 */
    u32CurData = DrvTIMER_GetCaptureData (E_TMR0);
    /* Clear capture interrupt status to receive the next valid capture value */
    DrvTIMER_ClearCaptureIntFlag (E_TMR0);
}
```

DrvTIMER_EnableCaptureDebounce

Prototype

```
int32_t DrvTIMER_EnableCaptureDebounce (
                    E_TIMER_CHANNEL ch,
    );
```

Description

Enable the debounce function of specified external capture input source.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Enable external timer-0 capture debounce function */
DrvTIMER_EnableCaptureDebounce (E_TMR0);
```

DrvTIMER_DisableCaptureDebounce

Prototype

```
int32_t DrvTIMER_DisableCaptureDebounce (
        E_TIMER_CHANNEL ch,
);
```

Description

Disable the debounce function of specified external capture input source.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Disable external timer-0 capture debounce function */
DrvTIMER_DisableCaptureDebounce (E_TMR0);
```

DrvTIMER_EnableCounterDebounce

Prototype

```
int32_t DrvTIMER_EnableCounterDebounce (
        E_TIMER_CHANNEL ch,
```

);

Description

Enable the debounce function of specified external counter input source.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

```
/* Enable external timer-0 counter debounce function */
DrvTIMER_EnableCounterDebounce (E_TMR0);
```

DrvTIMER_DisableCounterDebounce

Prototype

```
int32_t DrvTIMER_DisableCounterDebounce (
                    E_TIMER_CHANNEL ch,
    );
```

Description

Disable the debounce function of specified external counter input source.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

/* Disable external timer-0 counter debounce function */
DrvTIMER_DisableCounterDebounce (E_TMR0);

DrvTIMER_SelectCounterDetectPhase

Prototype

int32_t DrvTIMER_SelectCounterDetectPhase (
E_TIMER_CHANNEL ch,
E_TIMER_TX_PHASE phase
);

Description

This function is used to configure the counter detect phase of specified source.
Only NUC1x0xxxCx series support this function, ex:NUC140VE3CN.

Parameter

ch [in]

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

edge [in]

E_TIMER_TX_PHASE,

E_PHASE_FALLING : A falling edge of external counter pin will be counted.
E_PHASE_RISING : A rising edge of external counter pin will be counted.

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVTIMER_CHANNEL: Invalid timer channel

Example

/* Configure timer-0 counter detect phase is from low to high */
DrvTIMER_SelectCounterDetectPhase (E_TMR0, E_PHASE_RISING);

DrvTIMER_GetVersion

Prototype

uint32_t DrvTIMER_GetVersion (void)

Description

Get the version number of Timer/WDT driver.

Include

Driver/DrvTIMER.h

Return Value

Version number :

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

Example

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

DrvWDT_Open

Prototype

int32_t DrvWDT_Open (E_WDT_INTERVAL WDTlevel)

Description

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

Parameter

WDTlevel [in]

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

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_DRVWDT_OPEN: WDT open fail

Example

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

DrvWDT_Close

Prototype

void DrvWDT_Close (void)

Description

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

Parameter

None

Include

Driver/DrvTIMER.h

Return Value

None

Example

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

DrvWDT_InstallISR

Prototype

void DrvWDT_InstallISR (WDT_CALLBACK pvWDTISR)

Description

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

Parameter

pvWDTISR [in]

The function pointer of the interrupt service routine

Include

Driver/DrvTIMER.h

Return Value

None

Example

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

DrvWDT_Ioctl

Prototype

int32_T DrvWDT_Ioctl (E_WDT_CMD uWDTCmd, uint32_t uArgument)

Description

The function is used to operate more WDT applications, it could be the start/stop the WDT, enable/disable WDT interrupt function, enable/disable WDT time-out wake up function, enable/disable system reset when WDT time-out and set the WDT time-out

interval.
All bits in WDT register are write-protected. User must to check the REGWRPROT bit is enabled or disabled if write the specified WDT bit fail.

Parameter

uWDTCmd [in]

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

E_WDT_IOC_START_TIMER，
E_WDT_IOC_STOP_TIMER，
E_WDT_IOC_ENABLE_INT，
E_WDT_IOC_DISABLE_INT，
E_WDT_IOC_ENABLE _WAKEUP，
E_WDT_IOC_DISABLE_WAKEUP，
E_WDT_IOC_RESET_TIMER，
E_WDT_IOC_ENABLE_RESET_FUNC，
E_WDT_IOC_DISABLE_RESET_FUNC，
E_WDT_IOC_SET_INTERVAL

uArgument [in]

Set the argument for the specified WDT command

Include

Driver/DrvTIMER.h

Return Value

E_SUCCESS: Operation successful

E_ DRVWDT_CMD: Invalid WDT command

Example

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

5. GPIO Driver

5.1. GPIO introduction

NUC100 Medium Density Series has up to 80 General Purpose I/O pins can be shared with other function pins; it depends on the chip configuration. These 80 pins are arranged in 5 ports named with GPIOA, GPIOB, GPIOC, GPIOD and GPIOE. Each port equips maximum 16 pins.

NUC100 Low Density Series has up to 65 General Purpose I/O pins can be shared with other function pins; it depends on the chip configuration and package. These 65 pins are arranged in 5 ports. GPIOA, GPIOB, GPIOC and GPIOD with each port equips maximum 16 pins and GPIOE with 1 pin GPE[5].

5.2. GPIO Feature

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

5.3. Type Definition

E_DRVGPIO_PORT

Enumeration Identifier	Value	Description
E_GPA	0	Define GPIO Port A
E_GPB	1	Define GPIO Port B
E_GPC	2	Define GPIO Port C
E_GPD	3	Define GPIO Port D
E_GPE	4	Define GPIO Port E

E_DRVGPIO_IO

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

E_DRVGPIO_INT_TYPE

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

E_DRVGPIO_INT_MODE

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

E_DRVGPIO_DBCLKSRC

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

E_DRVGPIO_FUNC

Enumeration Identifier	Pins assignment	Description
E_FUNC_GPIO	All GPIO pins	Set all GPIO pins as GPIO functions
E_FUNC_CLKO	GPB.12	Enable Clock Driver Output function
E_FUNC_I2C0 / E_FUNC_I2C1	GPA.8~9 / GPA.10~11	Enable I2C0 and I2C1 functions
E_FUNC_I2S	GPA.15, GPC.0~3	Enable I2S function
E_FUNC_CAN0	GPD.6, GPD.7	Enable CAN0 function
E_FUNC_ACMP0 / E_FUNC_ACMP1	GPC.6~7 / GPC.14~15	Enable ACMP0 and ACMP1 function
E_FUNC_SPI0	GPC.0~3	Enable SPI0 SS0, CLK, MISO0 and MOSI0
E_FUNC_SPI0_SS1	GPB.10	Enable SPI0 SS1 function
E_FUNC_SPI0_2BIT_MODE	GPC.4 and GPC.5	Enable SPI0 MISO1 and MOSI1
E_FUNC_SPI1	GPC.8~11	Enable SPI1 SS0, CLK, MISO0 and MOSI0
E_FUNC_SPI1_SS1	GPB.9	Enable SPI1 SS1 function
E_FUNC_SPI1_2BIT_MODE	GPC.12 and GPC.13	Enable SPI1 MISO1 and MOSI1
E_FUNC_SPI2	GPD.0~3	Enable SPI2 SS0, CLK, MISO0 and MOSI0
E_FUNC_SPI2_SS1	GPA.7	Enable SPI2 SS1 function
E_FUNC_SPI2_2BIT_MODE	GPD.4 and GPC.5	Enable SPI2 MISO1 and MOSI1
E_FUNC_SPI3	GPD.8~11	Enable SPI3 SS0, CLK, MISO0 and MOSI0
E_FUNC_SPI3_SS1	GPB.14	Enable SPI3 SS1 function
E_FUNC_SPI3_2BIT_MODE	GPD.12 and GPD.13	Enable SPI3 MISO1 and MOSI1
E_FUNC_SPI0_QFN36PIN	GPC.0~3	Enable SPI0 SS0, CLK, MISO0 and MOSI0 for QFN36 package

E_FUNC_SPI0_SS1_QFN36PIN	GPD.1	Enable SPI0 SS1 for QFN36 package
E_FUNC_SPI0_2BIT_MODE_ QFN36PIN	GPD.2 and GPD.3	Enable SPI0 MISO1 and MOSI1 for QFN36 package
E_FUNC_ADC0 / E_FUNC_ADC1 / E_FUNC_ADC2 / E_FUNC_ADC3 / E_FUNC_ADC4 / E_FUNC_ADC5 / E_FUNC_ADC6 / E_FUNC_ADC7	GPA.0~7	Enable ADC0/ADC1/ADC2/ADC3/ ADC4/ADC5/ADC6/ADC7 functions
E_FUNC_EXTINT0 / E_FUNC_EXTINT1	GPB.14 / GPB.15	Enable External INT0/INT1 functions
E_FUNC_TMR0 / E_FUNC_TMR1 / E_FUNC_TMR2 / E_FUNC_TMR3	GPB.8~11	Enable TIMER0/TIMER1/TIMER2/ TIMER3 as Toggle/Counter mode
E_FUNC_T0EX / E_FUNC_T1EX / E_FUNC_T2EX / E_FUNC_T3EX	GPB.15, GPE.5, GPB.2 and GPB.3	Enable TIMER0/TIMER1/TIMER2/ TIMER3 as external Capture mode
E_FUNC_UART0	GPB.0~3	Enable UART0 RX, TX, RTS and CTS
E_FUNC_UART0_RX_TX	GPB.0~1	Enable UART0 RX, TX
E_FUNC_UART0_RTS_CTS	GPB.2~3	Enable UART0 RTS, CTS
E_FUNC_UART1	GPB.4~7	Enable UART1 RX, TX, RTS and CTS
E_FUNC_UART1_RX_TX	GPB.4~5	Enable UART1 RX, TX
E_FUNC_UART1_RTS_CTS	GPB.6~7	Enable UART1 RTS, CTS
E_FUNC_UART2	GPD.14~15	Enable UART2 RX, TX
E_FUNC_PWM01 / E_FUNC_PWM23 / E_FUNC_PWM45 / E_FUNC_PWM67	GPA.12~13 / GPA.14~15 / GPB.11, GPE.5 / GPE.0~1	Enable PWM01/PWM23/PWM45/ PWM67 functions
E_FUNC_PWM0 / E_FUNC_PWM1 / E_FUNC_PWM2 / E_FUNC_PWM3 / E_FUNC_PWM4 / E_FUNC_PWM5 / E_FUNC_PWM6 / E_FUNC_PWM7	GPA.12 / GPA.13 / GPA.14 / GPA.15 / GPB.11 / GPE.5 / GPE.0 / GPE.1	Enable PWM0/PWM1/PWM2/PWM3/ PWM4/PWM5/PWM6/PWM7 functions
E_FUNC_EBI_8B	GPB.12~13,GPC.14~15 GPC.6~7,GPA.6~7, GPB.6~7,GPA10~11	Enable EBI with 8 bit address width
E_FUNC_EBI_16B	GPB.12~13,GPC.14~15 GPC.6~7,GPA.6~7, GPA.5~1,GPA.12~14, GPB.6~7,GPA10~11 GPB.2~3	Enable EBI with 16 bit address width

5.4. Macros

_DRVGPIO_DOUT

Prototype

_DRVGPIO_DOUT (PortNum, PinNum)

Description

This macro is used to control I/O Bit Output/Input Control Register of the specified pin. User can set output data value of the specified pin by calling _DRVGPIO_DOUT macro, if the

GPIO pin is configured as output mode. Or get the input data value by calling _DRVGPIO_DOUT directly, if the GPIO pin is configured as input mode.

Note

Only NUC1x0xxxBx and NUC1x0xxxCx series support this function, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

PortNum [in]

Specify the GPIO port. It could be 0~4 to correspond to the GPIO-A/B/C/D/E.

PinNum [in]

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

Include

Driver/DrvGPIO.h

Example

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

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

Prototype

GPA_0~GPA_15 / GPB_0~GPB_15 / GPC_0~GPC_15 / GPD_0~GPD_15 / GPE_0~GPE_15

Description

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

Note

Only NUC1x0xxxBx and NUC1x0xxxCx series support this function, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

None

Include

Driver/DrvGPIO.h

Example

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

5.5. Functions

DrvGPIO_Open

Prototype

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

Description

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

Parameter

port [in]

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

i32Bit [in]

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

mode [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_Close

Prototype

int32_t DrvGPIO_Close (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_SetBit

Prototype

int32_t DrvGPIO_SetBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Set the specified GPIO pin to 1.

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetBit

Prototype

int32_t DrvGPIO_GetBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Get the pin value from the specified input GPIO pin.

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

The specified input pin value: 0 / 1

E_DRVGPIO_ARGUMENT: ncorrect argument

Example

```
int32_t i32BitValue;
/* Configure GPA-1 as GPIO input mode*/
DrvGPIO_Open (E_GPA, 1, E_IO_INPUT);
i32BitValue = DrvGPIO_GetBit (E_GPA, 1);
```

```
if    (u32BitValue == 1)
{
       printf("GPA-1 pin status is high.\n");
}
else
{
       printf("GPA-1 pin status is low.\n");
}
```

DrvGPIO_ClrBit

Prototype

int32_t DrvGPIO_ClrBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

Set the specified GPIO pin to 0.

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect arguments

Example

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

DrvGPIO_SetPortBits

Prototype

int32_tDrvGPIO_SetPortBits (E_DRVGPIO_PORT port, int32_t i32Data)

Description

Set the output port value to the specified GPIO port.

Parameter

port [in]

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

i32Data [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetPortBits

Prototype

int32_t DrvGPIO_GetPortBits (E_DRVGPIO_PORT port)

Description

Get the input port value from the specified GPIO port.

Parameter

port [in]

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

Include

Driver/DrvGPIO.h

Return Value

The specified input port value: 0 ~ 0xFFFF

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetDoutBit

Prototype

int32_t DrvGPIO_GetDoutBit (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified register: 0 / 1

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetPortDoutBits

Prototype

int32_t DrvGPIO_GetPortDoutBits (E_DRVGPIO_PORT port)

Description

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

Parameter

port [in]

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

Include

Driver/DrvGPIO.h

Return Value

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

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_SetBitMask

Prototype

int32_t DrvGPIO_SetBitMask (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

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

DrvGPIO_GetBitMask

Prototype

int32_t DrvGPIO_GetBitMask (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified register: 0 / 1

Example

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

DrvGPIO_ClrBitMask

Prototype

int32_t DrvGPIO_ClrBitMask (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

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

DrvGPIO_SetPortMask

Prototype

int32_t DrvGPIO_SetPortMask (E_DRVGPIO_PORT port, int32_t i32MaskData)

Description

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

Parameter

port [in]

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

i32MaskData [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetPortMask

Prototype

int32_t DrvGPIO_GetPortMask (E_DRVGPIO_PORT port)

Description

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

Parameter

port [in]

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

Include

Driver/DrvGPIO.h

Return Value

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

Example

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

DrvGPIO_ClrPortMask

Prototype

int32_t DrvGPIO_ClrPortMask (E_DRVGPIO_PORT port, int32_t i32MaskData)

Description

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

Parameter

port [in]

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

i32MaskData [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

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

DrvGPIO_EnableDigitalInputBit

Prototype

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

Description

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

Parameter

port [in]

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

pin [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example:

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

DrvGPIO_DisableDigitalInputBit

Prototype

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

Description

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

Parameter

port [in]

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

pin [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example:

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

DrvGPIO_EnableDebounce

Prototype

int32_t DrvGPIO_EnableDebounce (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

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

DrvGPIO_DisableDebounce

Prototype

int32_t DrvGPIO_DisableDebounce (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

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

DrvGPIO_SetDebounceTime

Prototype

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

Description

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

Parameter

u32CycleSelection [in]

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

ClockSource [in]

E_DRVGPIO_DBCLKSRC, it could be DBCLKSRC_HCLK or DBCLKSRC_10K.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetDebounceSampleCycle

Prototype

```
int32_t DrvGPIO_GetDebounceSampleCycle (void)
```

Description

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

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

Number of the sampling cycle selection: 0 ~ 15

Example

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

DrvGPIO_EnableInt

Prototype

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

Description

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

Parameter

port [in]

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

i32Bit [in]

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

TriggerType [in]

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

Mode [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_DisableInt

Prototype

int32_t DrvGPIO_DisableInt (E_DRVGPIO_PORT port, int32_t i32Bit)

Description

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

Parameter

port [in]

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

i32Bit [in]

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

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example

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

DrvGPIO_SetIntCallback

Prototype

```
void DrvGPIO_SetIntCallback (
    GPIO_GPAB_CALLBACK pfGPABCallback,
    GPIO_GPCDE_CALLBACK pfGPCDECallback
)
```

Description

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

Parameter

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

Include

Driver/DrvGPIO.h

Return Value

None

Example

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

DrvGPIO_EnableEINT0

Prototype

```
void DrvGPIO_EnableEINT0 (
    E_DRVGPIO_INT_TYPE TriggerType,
    E_DRVGPIO_INT_MODE Mode,
    GPIO_EINT0_CALLBACK pfEINT0Callback
)
```

Description

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

Parameter

TriggerType [in]

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

Mode [in]

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

pfEINT0Callback [in]

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

Include

Driver/DrvGPIO.h

Return Value

None

Example

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

DrvGPIO_DisableEINT0

Prototype

void DrvGPIO_DisableEINT0 (void)

Description

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

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

None

Example

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

DrvGPIO_EnableEINT1

Prototype

void DrvGPIO_EnableEINT1 (

E_DRVGPIO_INT_TYPE TriggerType,

E_DRVGPIO_INT_MODE Mode,

GPIO_EINT0_CALLBACK pfEINT0Callback

)

Description

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

Parameter

TriggerType [in]

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

Mode [in]

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

pfEINT1Callback [in]

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

Include

Driver/DrvGPIO.h

Return Value

None

Example

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

DrvGPIO_DisableEINT1

Prototype

void DrvGPIO_DisableEINT1 (void)

Description

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

Parameter

None

Include

Driver/DrvGPIO.h

Return Value

None

Example

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

DrvGPIO_GetIntStatus

Prototype

uint32_t DrvGPIO_GetIntStatus (E_DRVGPIO_PORT port)

Description

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

Parameter

port [in]

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

Include

Driver/DrvGPIO.h

Return Value

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

Example

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

DrvGPIO_InitFunction

Prototype

int32_t DrvGPIO_InitFunction (E_DRVGPIO_FUNC function)

Description

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

Note

Not all the chips support these functions. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Parameter

function [in]

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

E_FUNC_GPIO,
E_FUNC_CLKO,
E_FUNC_I2C0 / E_FUNC_I2C1,
E_FUNC_I2S,
E_FUNC_CAN0,
E_FUNC_ACMP0 / E_FUNC_ACMP1,
E_FUNC_SPI0 / E_FUNC_SPI0_SS1 / E_FUNC_SPI0_2BIT_MODE,
E_FUNC_SPI1 / E_FUNC_SPI1_SS1 / E_FUNC_SPI1_2BIT_MODE,
E_FUNC_SPI2 / E_FUNC_SPI2_SS1 / E_FUNC_SPI2_2BIT_MODE,
E_FUNC_SPI3 / E_FUNC_SPI3_SS1 / E_FUNC_SPI3_2BIT_MODE,
E_FUNC_SPI0_QFN36PIN / E_FUNC_SPI0_SS1_QFN36PIN /
E_FUNC_SPI0_2BIT_MODE_QFN36PIN,
E_FUNC_ADC0 / E_FUNC_ADC1 / E_FUNC_ADC2 / E_FUNC_ADC3 /
E_FUNC_ADC4 / E_FUNC_ADC5 / E_FUNC_ADC6 / FUNC_ADC7,
E_FUNC_EXTINT0 / E_FUNC_EXTINT1,
E_FUNC_TMR0 / E_FUNC_TMR1 / E_FUNC_TMR2 / E_FUNC_TMR3,
E_FUNC_T0EX / E_FUNC_T1EX / E_FUNC_T2EX / E_FUNC_T3EX,
E_FUNC_UART0 / E_FUNC_UART0_RX_TX / E_FUNC_UART0_RTS_CTS,
E_FUNC_UART1 / E_FUNC_UART1_RX_TX / E_FUNC_UART1_RTS_CTS,
E_FUNC_UART2 / E_FUNC_UART2_RX_TX / E_FUNC_UART2_RTS_CTS,
E_FUNC_PWM01 / E_FUNC_PWM23 / E_FUNC_PWM45 / E_FUNC_PWM67,
E_FUNC_PWM0 / E_FUNC_PWM1 / E_FUNC_PWM2 / E_FUNC_PWM3 /
E_FUNC_PWM4 / E_FUNC_PWM5 / E_FUNC_PWM6 / E_FUNC_PWM7,
E_FUNC_EBI_8B / E_FUNC_EBI_16B,

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

E_DRVGPIO_ARGUMENT: Incorrect argument

Example

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

DrvGPIO_GetVersion

Prototype

uint32_t DrvGPIO_GetVersion (void)

Description

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

Include

Driver/DrvGPIO.h

Return Value

The version number of GPIO driver:

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

Example

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

6. ADC Driver

6.1. ADC Introduction

NuMicro™ NUC100 series contain one 12-bit successive approximation analog-to-digital converters (SAR A/D converter) with 8 input channels. It takes about 27 ADC clock cycles to convert one sample, and the maximum input clock to ADC is 16MHz at 5.0V.The A/D converter supports three operation modes: single, single-cycle scan and continuous scan mode. The A/D converters can be started by software and external STADC/PB.8 pin. In this document, we will introduce how to use the ADC driver.

6.2. ADC Feature

The Analog to Digital Converter includes following features:

- Analog input voltage range: 0~Vref (Max to 5.0V).
- 12 bits resolution.
- Up to 8 analog input channels.
- Maximum ADC clock frequency is 16MHz.
- Three operating modes
  - Single mode
  - Single-cycle scan mode
  - Continuous scan mode
- An A/D conversion can be started by
  - Software write 1 to ADST bit
  - External pin STADC
- Conversion result can be compared with specify value and provide interrupt function when conversion result matches the compare register settings.
- The APIs include setting conditions and getting conversion data for ADC applications.
- Channel 7 supports 3 input sources: external analog voltage, internal fixed bandgap voltage and internal temperature sensor output.
- Support Self-calibration to minimize conversion error.
- Support single end and differential input signal.

6.3. Type Definition

E_ADC_INPUT_MODE

Enumeration Identifier	Value	Description
ADC_SINGLE_END	0	ADC single end input
ADC_DIFFERENTIAL	1	ADC differential input

E_ADC_OPERATION_MODE

Enumeration Identifier	Value	Description
ADC_SINGLE_OP	0	Single operation mode
ADC_SINGLE_CYCLE_OP	1	Single cycle scan mode
ADC_CONTINUOUS_OP	2	Continuous scan mode

E_ADC_CLK_SRC

Enumeration Identifier	Value	Description
EXTERNAL_12MHZ	0	External 12MHz clock
INTERNAL_PLL	1	Internal PLL clock
INTERNAL_HCLK	2	System clock
INTERNAL_RC22MHZ	3	Internal 22.1184MHz clock

E_ADC_EXT_TRI_COND

Enumeration Identifier	Value	Description
LOW_LEVEL	0	Low level trigger
HIGH_LEVEL	1	High level trigger
FALLING_EDGE	2	Falling edge trigger
RISING_EDGE	3	Rising edge trigger

E_ADC_CH7_SRC

Enumeration Identifier	Value	Description
EXTERNAL_INPUT_SIGNAL	0	External input signal
INTERNAL_BANDGAP	1	Internal bandgap voltage
INTERNAL_TEMPERATURE_SENSOR	2	Internal temperature sensor

E_ADC_CMP_CONDITION

Enumeration Identifier	Value	Description
LESS_THAN	0	Less than compare data
GREATER_OR_EQUAL	1	Greater or equal to compare data

E_ADC_DIFF_MODE_OUTPUT_FORMAT

Enumeration Identifier	Value	Description
UNSIGNED_OUTPUT	0	Unsigned format
TWOS_COMPLEMENT	1	2’s complement format

6.4. Macros

_DRVADC_CONV

Prototype

void _DRVADC_CONV (void);

Description

Inform ADC to start an A/D conversion.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Start an A/D conversion */
_DRVADC_CONV();
```

_DRVADC_GET_ADC_INT_FLAG

Prototype

uint32_t _DRVADC_GET_ADC_INT_FLAG (void);

Description

Get the status of ADC interrupt flag.

Include

Driver/DrvADC.h

Return Value

0: ADC interrupt does not occur.

1: ADC interrupt occurs.

Example

```
/* Get the status of ADC interrupt flag */
if(_DRVADC_GET_ADC_INT_FLAG())
    printf("ADC interrupt occurs.\n");
```

_DRVADC_GET_CMP0_INT_FLAG

Prototype

uint32_t _DRVADC_GET_CMP0_INT_FLAG (void);

Description

Get the status of ADC comparator 0 interrupt flag.

Include

Driver/DrvADC.h

Return Value

0: ADC comparator 0 interrupt does not occur.

1: ADC comparator 0 interrupt occurs.

Example

```
/* Get the status of ADC comparator 0 interrupt flag */
if(_DRVADC_GET_CMP0_INT_FLAG())
    printf("ADC comparator 0 interrupt occurs.\n");
```

_DRVADC_GET_CMP1_INT_FLAG

Prototype

uint32_t _DRVADC_GET_CMP1_INT_FLAG (void);

Description

Get the status of ADC comparator 1 interrupt flag.

Include

Driver/DrvADC.h

Return Value

0: ADC comparator 1 interrupt does not occur.

1: ADC comparator 1 interrupt occurs.

Example

```
/* Get the status of ADC comparator 1 interrupt flag */
if(_DRVADC_GET_CMP1_INT_FLAG())
    printf("ADC comparator 1 interrupt occurs.\n");
```

_DRVADC_CLEAR_ADC_INT_FLAG

Prototype

void _DRVADC_CLEAR_ADC_INT_FLAG (void);

Description

Clear the ADC interrupt flag.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Clear the ADC interrupt flag */
_DRVADC_CLEAR_ADC_INT_FLAG();
```

_DRVADC_CLEAR_CMP0_INT_FLAG

Prototype

void _DRVADC_CLEAR_CMP0_INT_FLAG (void);

Description

Clear the ADC comparator 0 interrupt flag.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Clear the ADC comparator 0 interrupt flag */
_DRVADC_CLEAR_CMP0_INT_FLAG();
```

_DRVADC_CLEAR_CMP1_INT_FLAG

Prototype

void _DRVADC_CLEAR_CMP1_INT_FLAG (void);

Description

Clear the ADC comparator 1 interrupt flag.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Clear the ADC comparator 1 interrupt flag */

_DRVADC_CLEAR_CMP1_INT_FLAG();

6.5. Functions

DrvADC_Open

Prototype

```
void DrvADC_Open (
    E_ADC_INPUT_MODE InputMode,
    E_ADC_OPERATION_MODE OpMode,
    uint8_t u8ChannelSelBitwise,
    E_ADC_CLK_SRC ClockSrc,
    uint8_t u8AdcDivisor
);
```

Description

Enable the ADC function and configure the related settings.

Parameters

InputMode [in]

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

ADC_SINGLE_END	: single-end input mode
ADC_DIFFERENTIAL	: differential input mode

OpMode [in]

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

ADC_SINGLE_OP	: single mode
ADC_SINGLE_CYCLE_OP	: single cycle scan mode
ADC_CONTINUOUS_OP	: continuous scan mode

u8ChannelSelBitwise [in]

Specify the input channels. If software enables more than one channel in single mode, only the channel with the lowest number will be converted and the other enabled channels will be ignored. For example, if user enable channel 2, 3 and 4 in single mode, only channel 2 will be converted. In differential input mode, only the even number of the two corresponding channels needs to be enabled. The conversion result will be placed to the corresponding data register of the selected channel. For example, in single-end input mode, 0x4 means the channel 2 is selected; in differential input mode, it means channel pair 1 is selected.

ClockSrc [in]

Specify the clock source of ADC clock.

EXTERNAL_12MHZ	: external 12MHz crystal
INTERNAL_PLL	: internal PLL output
INTERNAL_HCLK	: system clock
INTERNAL_RC22MHZ	: internal 22.1184MHz RC oscillator

u8AdcDivisor [in]

Determine the ADC clock frequency. The range of u8AdcDivisor is 0 ~ 0xFF.

ADC clock frequency = ADC clock source frequency / (u8AdcDivisor + 1)

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* single end input, single operation mode, channel 5 is selected, ADC clock frequency = 12MHz/(5+1) */
DrvADC_Open(ADC_SINGLE_END, ADC_SINGLE_OP, 0x20, EXTERNAL_12MHZ, 5);
```

DrvADC_Close

Prototype

void DrvADC_Close (void);

Description

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

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Close the ADC function */
DrvADC_Close();
```

DrvADC_SetADCChannel

Prototype

```
void DrvADC_SetADCChannel (
    uint8_t u8ChannelSelBitwise,
    E_ADC_INPUT_MODE InputMode
);
```

Description

Select ADC input channels.

Parameters

u8ChannelSelBitwise [in]

Specify the input channels. If software enables more than one channel in single mode, only the channel with the lowest number will be converted and the other enabled channels will be ignored. For example, if user enable channel 2, 3 and 4 in single mode, only channel 2 will be converted. In differential input mode, only the even number of the two corresponding channels needs to be enabled. The conversion result will be placed to the corresponding data register of the selected channel. For example, in single-end input mode, 0x4 means the channel 2 is selected; in differential input mode, it means channel pair 1 is selected.

InputMode [in]

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

ADC_SINGLE_END	: single-end input mode
ADC_DIFFERENTIAL	: differential input mode

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* In single-end input mode, this function select channel 0 and channel 2; In differential input mode, it select channel pair 0 and channel pair 1. */
DrvADC_SetADCChannel (0x5);
```

DrvADC_ConfigADCChannel7

Prototype

```
void DrvADC_ConfigADCChannel7 (E_ADC_CH7_SRC Ch7Src);
```

Description

Select the input signal source of ADC channel 7.

Parameters

Ch7Src [in]

Specify the analog input source.

EXTERNAL_INPUT_SIGNAL : external analog input

INTERNAL_BANDGAP : internal band gap voltage

INTERNAL_TEMPERATURE_SENSOR : internal temperature sensor

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Select the external analog input as the source of channel 7 */
DrvADC_ConfigADCChannel7(EXTERNAL_INPUT_SIGNAL);
```

DrvADC_SetADCInputMode

Prototype

void DrvADC_SetADCInputMode (E_ADC_INPUT_MODE InputMode);

Description

Set the ADC input mode.

Parameters

InputMode [in]

Specify the input mode.

ADC_SINGLE_END : single-end input mode

ADC_DIFFERENTIAL : differential input mode

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* The following statement indicates that the external analog input is a single-end input */
DrvADC_SetADCInputMode(ADC_SINGLE_END);
```

DrvADC_SetADCOperationMode

Prototype

void DrvADC_SetADCOperationMode (E_ADC_OPERATION_MODE OpMode);

Description

Set the ADC operation mode.

Parameters

OpMode [in]

Specify the operation mode.

ADC_SINGLE_OP	: single mode
ADC_SINGLE_CYCLE_OP	: single cycle scan mode
ADC_CONTINUOUS_OP	: continuous scan mode

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* The following statement configures the single mode as the operation mode */
DrvADC_SetADCOperationMode(ADC_SINGLE_OP);
```

DrvADC_SetADCClkSrc

Prototype

void DrvADC_SetADCClkSrc (E_ADC_CLK_SRC ClockSrc);

Description

Select the ADC clock source.

Parameters

ClockSrc [in]

Specify the ADC clock source.

EXTERNAL_12MHZ	: external 12MHz crystal
INTERNAL_PLL	: internal PLL output
INTERNAL_HCLK	: system clock
INTERNAL_RC22MHZ	: internal 22.1184MHz RC oscillator

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Select the external 12MHz crystal as the clock source of ADC */

DrvADC_SetADCClkSrc (EXTERNAL_12MHZ);

DrvADC_SetADCDivisor

Prototype

void DrvADC_SetADCDivisor (uint8_t u8AdcDivisor);

Description

Set the divisor value of ADC clock to determine the ADC clock frequency.

ADC clock frequency = ADC clock source frequency / (u8AdcDivisor + 1)

Parameters

u8AdcDivisor [in]

Specify the divisor value. The range of u8AdcDivisor is 0 ~ 0xFF.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* The clock source of ADC is from external 12MHz crystal. The ADC clock frequency is 2MHz. */

DrvADC_SetADCClkSrc (EXTERNAL_12MHZ);

DrvADC_SetADCDivisor (5);

DrvADC_EnableADCInt

Prototype

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

Description

Enable ADC interrupt and setup the callback function. As an ADC interrupt occurs, the callback function will be executed. When the ADC interrupt function is enabled and one of the following conditions happens, the ADC interrupt will be asserted.

- The A/D conversion of the specified channel is completed in single mode.
- The A/D conversions of all selected channels are completed in single cycle scan mode or continuous scan mode.

Parameters

Callback [in]

The callback function of the ADC interrupt.

u32UserData [in]

The parameter of the callback function.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* ADC interrupt callback function */
void AdcIntCallback(uint32_t u32UserData)
{
    gu8AdcIntFlag = 1;
}
/* Enable the ADC interrupt and setup the callback function. The parameter 0 will be passed to the callback function. */
DrvADC_EnableADCInt(AdcIntCallback, 0);
```

DrvADC_DisableADCInt

Prototype

void DrvADC_DisableADCInt (void);

Description

Disable the ADC interrupt.

Parameters

None

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Disable the ADC interrupt */
DrvADC_DisableADCInt();
```

DrvADC_EnableADCCmp0Int

Prototype

```
void DrvADC_EnableADCCmp0Int (
        DRVADC_ADCMP0_CALLBACK Callback,
        uint32_t u32UserData
);
```

Description

Enable the ADC comparator 0 interrupt and setup callback function. If the conversion result satisfies the compare conditions set in DrvADC_EnableADCCmp0(), a comparator 0 interrupt will be asserted and the callback function will be executed.

Parameters

Callback [in]

The callback function of the ADC comparator 0 interrupt.

u32UserData [in]

The parameter of the callback function.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* ADC comparator 0 interrupt callback function */
void Cmp0IntCallback(uint32_t u32UserData)
{
    gu8AdcCmp0IntFlag = 1;
}
int32_t main()
{
    …
```

```
/* Enable the ADC comparator 0 interrupt and setup the callback function. The parameter
0 will be passed to the callback function. */
DrvADC_EnableADCCmp0Int(Cmp0IntCallback, 0);
}
```

DrvADC_DisableADCCmp0Int

Prototype

void DrvADC_DisableAdcmp0Int (void);

Description

Disable the ADC comparator 0 interrupt.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Disable the ADC comparator 0 interrupt */
DrvADC_DisableADCCmp0Int();
```

DrvADC_EnableADCCmp1Int

Prototype

```
void DrvADC_EnableADCCmp1Int (
    DRVADC_ADCMP1_CALLBACK Callback,
    uint32_t u32UserData
);
```

Description

Enable the ADC comparator 1 interrupt and setup callback function. If the conversion result satisfies the compare conditions set in DrvADC_EnableADCCmp1(), a comparator 1 interrupt will be asserted and the callback function will be executed.

Parameters

Callback [in]

The callback function of the ADC comparator 1 interrupt.

u32UserData [in]

The parameter of the callback function.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* ADC comparator 1 interrupt callback function */
void Cmp1IntCallback(uint32_t u32UserData)
{
    gu8AdcCmp1IntFlag = 1;
}
int32_t main()
{
    …
    /* Enable the ADC comparator 1 interrupt and setup the callback function. The parameter
    0 will be passed to the callback function. */
    DrvADC_EnableADCCmp1Int(Cmp1IntCallback, 0);
}
```

DrvADC_DisableADCCmp1Int

Prototype

void DrvADC_DisableADCCmp1Int (void);

Description

Disable the ADC comparator 1 interrupt.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Disable the ADC comparator 1 interrupt */
DrvADC_DisableADCCmp1Int();
```

DrvADC_GetConversionRate

Prototype

uint32_t DrvADC_GetConversionRate (void);

Description

Get the A/D conversion rate. The ADC takes about 27 ADC clock cycles for converting one sample.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

Return the conversion rate. The unit is sample/second.

Example

```
/* The clock source of ADC is from external 12MHz crystal. The ADC clock frequency is
2MHz. The conversion rate is about 74K sample/second */
DrvADC_SetADCClkSrc (EXTERNAL_12MHZ);
DrvADC_SetADCDivisor (5);
/* Get the conversion rate */
printf("Conversion rate: %d samples/second\n", DrvADC_GetConversionRate());
```

DrvADC_EnableExtTrigger

Prototype

void DrvADC_EnableExtTrigger (E_ADC_EXT_TRI_COND TriggerCondition);

Description

Allow the external trigger pin (PB8) to be the trigger source of ADC. The external trigger pin must be configured as an input pin in advance.

Parameters

TriggerCondition [in]

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

LOW_LEVEL	: low level.
HIGH_LEVEL	: high level.
FALLING_EDGE	: falling edge.
RISING_EDGE	: rising edge.

Include

Driver/DrvADC.h

Return Value

None

Example

/* Use PB8 pin as the external trigger pin. The trigger condition is low level trigger. */

DrvADC_EnableExtTrigger(LOW_LEVEL);

DrvADC_DisableExtTrigger

Prototype

void DrvADC_DisableExtTrigger (void);

Description

Prohibit the external ADC trigger.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Disable the ADC external trigger source */

DrvADC_DisableExtTrigger ();

DrvADC_StartConvert

Prototype

void DrvADC_StartConvert(void);

Description

Clear the ADC interrupt flag (ADF) and start A/D converting.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Clear ADF bit and start converting */

DrvADC_StartConvert();

DrvADC_StopConvert

Prototype

void DrvADC_StopConvert(void);

Description

Stop A/D converting.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Stop converting */

DrvADC_StopConvert();

DrvADC_IsConversionDone

Prototype

uint32_t DrvADC_IsConversionDone (void);

Description

Check whether the conversion action is finished or not.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

TURE Conversion finished

FALSE In converting

Example

/* If the ADC interrupt is not enabled, user can call this function to check the state of conversion action */

/* Start A/D conversion */

DrvADC_StartConvert();

/* Wait conversion done */

while(!DrvADC_IsConversionDone());

DrvADC_GetConversionData

Prototype

int32_t DrvADC_GetConversionData (uint8_t u8ChannelNum);

Description

Get the conversion result of the specified ADC channel.

Parameters

u8ChannelNum [in]

Specify the ADC channel. The range of this value is 0~7.

Include

Driver/DrvADC.h

Return Value

A 32-bit conversion result. It is generated by extending the original 12 bits conversion result.

Example

/* Get the conversion result of ADC channel 3 */

printf("Conversion result of channel 3: %d\n", DrvADC_GetConversionData(3));

DrvADC_EnablePDMA

Prototype

void DrvADC_EnablePDMA (void);

Description

Enable PDMA transfer. User can transfer the A/D conversion result to user-specified memory space by PDMA without CPU intervention. In single mode, only the conversion result of the selected channel will be transferred. In single cycle scan mode or continuous scan mode, the conversion results of all enabled channels will be transferred by PDMA.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None

Example

/* Enable PDMA transfer */

DrvADC_EnablePDMA();

DrvADC_DisablePDMA

Prototype

void DrvADC_DisablePDMA (void);

Description

Disable PDMA transfer.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None

Example

/* Disable PDMA transfer */

DrvADC_DisablePDMA();

DrvADC_IsDataValid

Prototype

uint32_t DrvADC_IsDataValid (uint8_t u8ChannelNum);

Description

Check whether the conversion data is valid or not.

Parameters

u8ChannelNum [in]

Specify the ADC channel. The range of this value is 0~7.

Include

Driver/DrvADC.h

Return Value

TURE: data is valid

FALSE: data is invalid

Example

```
/* Check if the data of channel 3 is valid. */
If( DrvADC_IsDataValid(3) )
    u32ConversionData = DrvADC_GetConversionData(u8ChannelNum); /* Get the data */
```

DrvADC_IsDataOverrun

Prototype

uint32_t DrvADC_IsDataOverrun (uint8_t u8ChannelNum);

Description

Check whether the conversion data is overrun or not.

Parameters

u8ChannelNum [in]

Specify the ADC channel. The range of this value is 0~7.

Include

Driver/DrvADC.h

Return Value

TURE	overrun
FALSE	non-overrun

Example

```
/* Check if the data of channel 3 is overrun. */
If(DrvADC_IsDataOverrun(3) )
    printf("The data has been overwritten.\n");
```

DrvADC_EnableADCCmp0

Prototype

```
int32_t DrvADC_EnableADCCmp0 (
    uint8_t u8CmpChannelNum,
    E_ADC_CMP_CONDITION CmpCondition,
```

uint16_t u16CmpData,

uint8_t u8CmpMatchCount

);

Description

Enable the ADC comparator 0 and configure the necessary settings.

Parameters

u8CmpChannelNum [in]

Specify the channel number that wants to compare. The range of this value is 0~7.

CmpCondition [in]

Specify the compare condition.

LESS_THAN : less than the compare data.

GREATER_OR_EQUAL : greater or equal to the compare data.

u16CmpData [in]

Specify the compare data. The range is 0 ~ 0xFFF.

u8CmpMatchCount [in]

Specify the compare match count. The range is 0 ~ 15. When the specified A/D channel analog conversion result matches the compare condition, the internal match counter will increase 1. When the internal counter reaches the value to (u8CmpMatchCount +1), the comparator 0 interrupt flag will be set.

Include

Driver/DrvADC.h

Return Value

E_SUCCESS: Success. The compare function is enabled.

E_DRVADC_ARGUMENT: One of the input arguments is out of the range

Example

u8CmpChannelNum = 0;

u8CmpMatchCount = 5;

/* Enable ADC comparator0. Compare condition: conversion result < 0x800. */

DrvADC_EnableADCCmp0(u8CmpChannelNum, LESS_THAN, 0x800, u8CmpMatchCount);

DrvADC_DisableADCCmp0

Prototype

void DrvADC_DisableADCCmp0 (void);

Description

Disable the ADC comparator 0.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Disable the ADC comparator 0 */

DrvADC_DisableADCCmp0();

DrvADC_EnableADCCmp1

Prototype

```
int32_t DrvADC_EnableADCCmp1 (
    uint8_t u8CmpChannelNum,
    E_ADC_CMP_CONDITION CmpCondition,
    uint16_t u16CmpData,
    uint8_t u8CmpMatchCount
);
```

Description

Enable the ADC comparator 1 and configure the necessary settings.

Parameters

u8CmpChannelNum [in]

Specify the channel number that wants to compare. The range of this value is 0~7.

CmpCondition [in]

Specify the compare condition.

LESS_THAN : less than the compare data.

GREATER_OR_EQUAL : greater or equal to the compare data.

u16CmpData [in]

Specify the compare data. The range is 0 ~ 0xFFF.

u8CmpMatchCount [in]

Specify the compare match count. The range is 0 ~ 15. When the specified A/D channel analog conversion result matches the compare condition, the internal match counter will increase 1. When the internal counter reaches the value to (u8CmpMatchCount +1), the interrupt flag of comparator 1 will be set.

Include

Driver/DrvADC.h

Return Value

E_SUCCESS: Success. The compare function is enabled.

E_DRVADC_ARGUMENT: One of the input arguments is out of the range

Example

u8CmpChannelNum = 0;

u8CmpMatchCount = 5;

/* Enable ADC comparator1. Compare condition: conversion result < 0x800. */

DrvADC_EnableADCCmp1(u8CmpChannelNum, LESS_THAN, 0x800, u8CmpMatchCount);

DrvADC_DisableADCCmp1

Prototype

void DrvADC_DisableADCCmp1 (void);

Description

Disable the ADC comparator 1.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Disable the ADC comparator 1 */

DrvADC_DisableADCCmp1();

DrvADC_EnableSelfCalibration

Prototype

void DrvADC_EnableSelfCalibration (void);

Description

Enable the self calibration function for minimizing the A/D conversion error. When chip power on or software switches the ADC input type between single-end mode and differential mode, user needs to call this function to enable the self calibration. After call this function, user can call DrvADC_IsCalibrationDone() to check if the self calibration is done before any A/D conversion.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

```
/* Enable the self calibration function */
DrvADC_EnableSelfCalibration();
```

DrvADC_IsCalibrationDone

Prototype

uint32_t DrvADC_IsCalibrationDone (void);

Description

Check whether the self calibration action is finished or not.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

TURE: the self calibration action is finished.

FALSE: the self calibration action is in progress.

Example

```
if( DrvADC_IsCalibrationDone() )
    printf("Self calibration done.\n");
```

DrvADC_DisableSelfCalibration

Prototype

void DrvADC_DisableSelfCalibration (void);

Description

Disable the self calibration function.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

None.

Example

/* Disable the self calibration function */

DrvADC_DisableSelfCalibration();

DrvADC_DiffModeOutputFormat

Prototype

void DrvADC_DiffModeOutputFormat (

E_ADC_DIFF_MODE_OUTPUT_FORMAT OutputFormat

);

Description

Select the output format of differential input mode. Only NUC101 and low density version of NuMicro™ NUC100 series products support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix.

Parameters

OutputFormat [in]

Specify the output format. It could be unsigned format (UNSIGNED_OUTPUT) or 2's complement format (TWOS_COMPLEMENT.)

Include

Driver/DrvADC.h

Return Value

None

Example

/* 2's complement format */

DrvADC_DiffModeOutputFormat(TWOS_COMPLEMENT);

DrvADC_GetVersion

Prototype

uint32_t DrvADC_GetVersion (void);

Description

Return the current version number of ADC driver.

Parameters

None.

Include

Driver/DrvADC.h

Return Value

Version number:

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

Example

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

7. SPI Driver

7.1. SPI Introduction

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

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

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

The controller also supports two bits transfer mode which is defined in the SPI_CNTL[22]. When the TWOB bit, in SPI_CNTL[22], is enabled, it can transmit and receive two bit serial data via the transmission/reception buffers. The 1st bit channel transmits the data from TX0 and receives the data into RX0. The 2nd bit channel transmits the data from TX1 and receives the data into RX1.

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

7.2. SPI Feature

- Up to four sets of SPI controller.
- Support master/slave mode operation.
- Support 1- or 2-bit serial data IN/OUT.
- Configurable data length of transfer word up to 32 bits.
- Variable output serial clock frequency in master mode.
- Provide burst mode operation, transmit/receive can be executed up to two times in one transfer.
- MSB or LSB first data transfer.
- 2 slave/device select lines in the master mode.
- Support Byte Reorder function.
- Compatible with Motorola SPI and National Semiconductor Microwire Bus.

7.3. Type Definition

E_DRVSPI_PORT

Enumeration Identifier	Value	Description
eDRVSPI_PORT0	0	SPI port 0
eDRVSPI_PORT1	1	SPI port 1
eDRVSPI_PORT2	2	SPI port 2
eDRVSPI_PORT3	3	SPI port 3

E_DRVSPI_MODE

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

E_DRVSPI_TRANS_TYPE

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

E_DRVSPI_ENDIAN

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

E_DRVSPI_BYTE_REORDER

Enumeration Identifier	Value	Description
eDRVSPI_BYTE_REORDER_SUSPEND_DISABLE	0	Both Byte Reorder function and Byte Suspend function are disabled
eDRVSPI_BYTE_REORDER_SUSPEND	1	Both Byte Reorder function and Byte Suspend function are enabled

Enumeration Identifier	Value	Description
eDRVSPI_BYTE_REORDER	2	Enable the Byte Reorder function
eDRVSPI_BYTE_SUSPEND	3	Enable the Byte Suspend function

E_DRVSPI_SSLTRIG

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

E_DRVSPI_SS_ACT_TYPE

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

E_DRVSPI_SLAVE_SEL

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

E_DRVSPI_DMA_MODE

Enumeration Identifier	Value	Description
eDRVSPI_TX_DMA	0	Enable Tx DMA
eDRVSPI_RX_DMA	1	Enable Rx DMA

7.4. Functions

DrvSPI_Open

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eMode [in]

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

eType [in]

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

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

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

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

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

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

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

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

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

i32BitLength [in]

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

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_INIT : The specified SPI port has been opened before.

E_DRVSPI_ERR_BIT_LENGTH : The bit length is out of range.

E_DRVSPI_ERR_BUSY : The specified SPI port is in busy status.

Example

```
/* Configure SPI0 as a master, 32-bit transaction, not QFN 36-pin package */
DrvSPI_Open(eDRVSPI_PORT0, eDRVSPI_MASTER, eDRVSPI_TYPE1, 32);
```

DrvSPI_Close

Prototype

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

Description

Close the specified SPI module and disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Close SPI0 */
DrvSPI_Close(eDRVSPI_PORT0);
```

DrvSPI_Set2BitTransferMode

Prototype

```
void DrvSPI_Set2BitTransferMode (
    E_DRVSPI_PORT eSpiPort,
    uint8_t bEnable
);
```

Description

Set 2-bit transfer mode.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

bEnable [in]

Enable (TRUE) / Disable (FALSE)

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable 2-bit transfer mode of SPI0 */
DrvSPI_Set2BitTransferMode(eDRVSPI_PORT0, TRUE);
```

DrvSPI_SetEndian

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eEndian [in]

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

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* The transfer order of SPI0 is LSB first */
DrvSPI_SetEndian(eDRVSPI_PORT0, eDRVSPI_LSB_FIRST);
```

DrvSPI_SetBitLength

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

i32BitLength [in]

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

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BIT_LENGTH : The bit length is out of range.

Example

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

DrvSPI_SetByteReorder

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eOption [in]

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

eDRVSPI_BYTE_REORDER_SUSPEND_DISABLE:

Both Byte Reorder function and Byte Suspend function are disabled.

eDRVSPI_BYTE_REORDER_SUSPEND:

Both Byte Reorder function and Byte Suspend function are enabled.

eDRVSPI_BYTE_REORDER:

Only enable the Byte Reorder function.

eDRVSPI_BYTE_SUSPEND:

Only enable the Byte Suspend function.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BIT_LENGTH : The bit length MUST be 8/16/24/32.

Example

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

DrvSPI_SetSuspendCycle

Prototype

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

Description

Set the number of clock cycle of the suspend interval. In slave mode, executing this function is unmeaningful.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

i32Interval [in]

In FIFO mode and burst transfer mode, this value specified the delay clock number between successive transactions. If the Byte Suspend function is enabled, it specified the delay clock number among each byte. Please refer to TRM for the calculation of the suspend interval.

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_SUSPEND_INTERVAL : The suspend interval setting is out of range.

Example

/* The suspend interval is 10 SPI clock cycles */

DrvSPI_SetSuspendCycle (eDRVSPI_PORT0, 10);

DrvSPI_SetTriggerMode

Prototype

void DrvSPI_SetTriggerMode (

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SSLTRIG eSSTriggerMode

);

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eSSTriggerMode [in]

Specify the trigger mode.

eDRVSPI_EDGE_TRIGGER: edge trigger.

eDRVSPI_LEVEL_TRIGGER: level trigger.

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Level trigger */

DrvSPI_SetTriggerMode(eDRVSPI_PORT0, eDRVSPI_LEVEL_TRIGGER);

DrvSPI_SetSlaveSelectActiveLevel

Prototype

void DrvSPI_SetSlaveSelectActiveLevel (

E_DRVSPI_PORT eSpiPort,

E_DRVSPI_SS_ACT_TYPE eSSActType

);

Description

Set the active level of slave select.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eSSActType [in]

Select the active type of slave select pin.

eDRVSPI_ACTIVE_LOW_FALLING:

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

eDRVSPI_ACTIVE_HIGH_RISING:

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

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Configure the active level of SPI0 slave select pin */

DrvSPI_SetSlaveSelectActiveLevel(eDRVSPI_PORT0, eDRVSPI_ACTIVE_LOW_FALLING);

DrvSPI_GetLevelTriggerStatus

Prototype

uint8_t DrvSPI_GetLevelTriggerStatus (

E_DRVSPI_PORT eSpiPort

);

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

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

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

Example

/* Level trigger */

DrvSPI_SetTriggerMode(eDRVSPI_PORT0, eDRVSPI_LEVEL_TRIGGER);

…

/* Check the level-trigger transmission status */

If(DrvSPI_GetLevelTriggerStatus(eDRVSPI_PORT0))

DrvSPI_DumpRxRegister(eDRVSPI_PORT0, &au32DestinationData[u32DataCount], 1); /* Read Rx buffer */

DrvSPI_EnableAutoSS

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eSlaveSel [in]

Select the slave select pins which will be used.

eDRVSPI_NONE : no slave was selected.

eDRVSPI_SS0 : the SS0 was selected.

eDRVSPI_SS1 : the SS1 was selected.

eDRVSPI_SS0_SS1 : both SS0 and SS1 were selected.

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_DisableAutoSS

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_SetSS

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eSlaveSel [in]

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

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

eDRVSPI_NONE : no slave was selected.

eDRVSPI_SS0 : the SS0 was selected.

eDRVSPI_SS1 : the SS1 was selected.

eDRVSPI_SS0_SS1 : both SS0 and SS1 were selected.

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_ClrSS

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eSlaveSel [in]

Specify slave select pins.

eDRVSPI_NONE : no slave was selected.

eDRVSPI_SS0 : the SS0 was selected.

eDRVSPI_SS1 : the SS1 was selected.

eDRVSPI_SS0_SS1 : both SS0 and SS1 were selected.

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_IsBusy

Prototype

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

Description

Check the busy status of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

TURE: The SPI port is in busy.

FALSE: The SPI port is not in busy.

Example

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

DrvSPI_BurstTransfer

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

i32BurstCnt [in]

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

i32Interval [in]

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

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BURST_CNT : The burst count is out of range.

E_DRVSPI_ERR_SUSPEND_INTERVAL : The interval is out of range.

Example

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

DrvSPI_BurstTransfer(eDRVSPI_PORT0, 2, 10);

DrvSPI_SetClockFreq

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0	: SPI0
eDRVSPI_PORT1	: SPI1
eDRVSPI_PORT2	: SPI2
eDRVSPI_PORT3	: SPI3

u32Clock1 [in]

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

u32Clock2 [in]

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

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

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

Example

/* SPI0 clock rate of clock 1 is 2MHz; the clock rate of clock 2 is 1MHz */

DrvSPI_SetClockFreq(eDRVSPI_PORT0, 2000000, 1000000);

DrvSPI_GetClock1Freq

Prototype

uint32_t

DrvSPI_GetClock1Freq(

E_DRVSPI_PORT eSpiPort

);

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

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

Example

/* Get the engine clock rate of SPI0 */

printf("SPI clock rate: %d Hz\n", DrvSPI_GetClock1Freq(eDRVSPI_PORT0));

DrvSPI_GetClock2Freq

Prototype

uint32_t

DrvSPI_GetClock2Freq(

E_DRVSPI_PORT eSpiPort

);

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

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

Example

/* Get the clock rate of SPI0 variable clock 2 */

printf("SPI clock rate of variable clock 2: %d Hz\n", DrvSPI_GetClock2Freq(eDRVSPI_PORT0));

DrvSPI_SetVariableClockFunction

Prototype

void

DrvSPI_SetVariableClockFunction (

E_DRVSPI_PORT eSpiPort,

uint8_t bEnable,

uint32_t u32Pattern

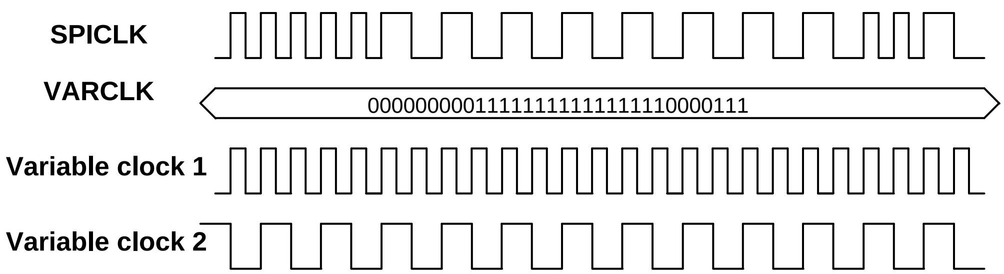
);

Description

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

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

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



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

In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

bEnable [in]

Enable (TRUE) / Disable (FALSE)

u32Pattern [in]

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

Include

Driver/DrvSPI.h

Return Value

None.

Example

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

DrvSPI_EnableInt

Prototype

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

Description

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

Parameters

u16Port [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pfnCallback [in]

The callback function of the corresponding SPI interrupt.

u32UserData [in]

The parameter which will be passed to the callback function.

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_DisableInt

Prototype

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

Description

Disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_GetIntFlag

Prototype

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

Description

Get the SPI interrupt flag.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

0: the SPI interrupt does not occur.

1: the SPI interrupt occurs.

Example

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

DrvSPI_ClrIntFlag

Prototype

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

Description

Clear the SPI interrupt flag.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None.

Example

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

DrvSPI_SingleRead

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pu32Data [out]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data is valid.

FALSE: The data stored in pu32Data is invalid.

Example

```
/* Read the previous retrieved data and trigger next transfer. */
uint32_t u32DestinationData;
DrvSPI_SingleRead(eDRVSPI_PORT0, &u32DestinationData);
```

DrvSPI_SingleWrite

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pu32Data [in]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data has been transferred.

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

Example

```
/* Write the data stored in u32SourceData to TX buffer of SPI0 and trigger SPI to start transfer. */
uint32_t u32SourceData;
DrvSPI_SingleWrite(eDRVSPI_PORT0, &u32SourceData);
```

DrvSPI_BurstRead

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pu32Buf [out]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf is valid.

FALSE: The data stored in pu32Buf is invalid.

Example

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

DrvSPI_BurstRead(eDRVSPI_PORT0, &au32DestinationData[u32DataCount]);

DrvSPI_BurstWrite

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pu32Buf [in]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf has been transferred.

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

Example

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

DrvSPI_BurstWrite(eDRVSPI_PORT0, &au32SourceData[u32DataCount]);

DrvSPI_DumpRxRegister

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pu32Buf [out]

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

u32DataCount [in]

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

Include

Driver/DrvSPI.h

Return Value

The count of data actually read from Rx registers.

Example

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

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

DrvSPI_SetTxRegister

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pu32Buf [in]

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

u32DataCount [in]

The count of data written to TX registers.

Include

Driver/DrvSPI.h

Return Value

The count of data actually written to SPI TX registers.

Example

```
/* Write one word of data stored in u32Buffer to SPI0 TX register. */
DrvSPI_SetTxRegister(eDRVSPI_PORT0, &u32Buffer, 1);
```

DrvSPI_SetGo

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_ClrGo

Prototype

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

Description

Stop a SPI data trasfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Stop a SPI data transfer */
```

DrvSPI_ClrGo(eDRVSPI_PORT0);

DrvSPI_SetPDMA

Prototype

```
void DrvSPI_SetPDMA (
    E_DRVSPI_PORT eSpiPort,
    E_DRVSPI_DMA_MODE eDmaMode,
    uint8_t bEnable
);
```

Description

Configure the DMA settings.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

eDmaMode [in]

Specify the DMA mode.

eDRVSPI_TX_DMA: DMA-Transmitting

eDRVSPI_RX_DMA: DMA-Receiving

eEnable [in]

True: Enable DMA.

False: Disable DMA.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable the SPI0 DMA-Receiving function */
DrvSPI_SetPDMA(eDRVSPI_PORT0, eDRVSPI_RX_DMA, TRUE);
```

DrvSPI_SetFIFOMode

Prototype

```
void
DrvSPI_SetFIFOMode (
    E_DRVSPI_PORT eSpiPort,
    uint8_t bEnable,
    int32_t i32Interval
);
```

Description

Enable/disable FIFO mode. If the caller enables FIFO mode, check the setting of suspend interval. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

bEnable [in]

Enable (TRUE) / Disable (FALSE)

i32Interval [in]

In FIFO mode, it could be 2~15 and 0. 0 indicates the maximum suspend interval; 2 indicates the minimum suspend interval. Please refer to NUC1xx TRM for the actual suspend interval.

Include

Driver/DrvSPI.h

Return Value

None.

Example

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

DrvSPI_IsRxEmpty

Prototype

```
uint8_t DrvSPI_IsRxEmpty(
    E_DRVSPI_PORT eSpiPort
);
```

Description

Check the status of the Rx buffer of the specified SPI port. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

TURE: Rx buffer is empty.

FALSE: Rx buffer is not empty.

Example

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

DrvSPI_IsRxFull

Prototype

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

Description

Check the status of the Rx buffer of the specified SPI port. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

TURE: Rx buffer is full.

FALSE: Rx buffer is not full.

Example

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

DrvSPI_IsTxEmpty

Prototype

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

Description

Check the status of the Tx buffer of the specified SPI port. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

TURE: Tx buffer is empty.

FALSE: Tx buffer is not empty.

Example

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

DrvSPI_IsTxFull

Prototype

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

Description

Check the status of the Tx buffer of the specified SPI port. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

TURE: Tx buffer is full.

FALSE: Tx buffer is not full.

Example

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

DrvSPI_ClrRxFIFO

Prototype

```
void DrvSPI_ClrRxFIFO (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Clear the Rx FIFO.
Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Clear the Rx FIFO.of SPI0 */
```

```
DrvSPI_ClrRxFIFO (eDRVSPI_PORT0);
```

DrvSPI_ClrTxFIFO

Prototype

```
void DrvSPI_ClrTxFIFO (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Clear the Tx FIFO.
Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Clear the Tx FIFO.of SPI0 */
DrvSPI_ClrTxFIFO (eDRVSPI_PORT0);
```

DrvSPI_EnableDivOne

Prototype

```
void DrvSPI_EnableDivOne (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Enable the DIV_ONE feature. The SPI clock rate will be equal to system clock rate. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Enable the DIV_ONE feature.of SPI0 */

DrvSPI_EnableDivOne (eDRVSPI_PORT0);

DrvSPI_DisableDivOne

Prototype

```
void DrvSPI_DisableDivOne (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Disable the DIV_ONE feature. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Disable the DIV_ONE feature.of SPI0 */

DrvSPI_DisableDivOne (eDRVSPI_PORT0);

DrvSPI_Enable3Wire

Prototype

```
void DrvSPI_Enable3Wire (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Enable the SPI 3-wire function. In master mode, executing this function is unmeaningful. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable the 3-wire SPI function.of SPI0 */
DrvSPI_Enable3Wire (eDRVSPI_PORT0);
```

DrvSPI_Disable3Wire

Prototype

```
void DrvSPI_Disable3Wire (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Disable the SPI 3-wire function. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Disable the 3-wire SPI function.of SPI0 */
DrvSPI_Disable3Wire (eDRVSPI_PORT0);
```

DrvSPI_3WireAbort

Prototype

```
void DrvSPI_3WireAbort (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Abort transfer when using 3-wire SPI. If using 3-wire SPI as slave, when slave start interrupt status is set but transfer done flag doesn't be set over a reasonable time, use this function to abort this transfer. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Abort current transfer.of SPI0 */

DrvSPI_ 3WireAbort (eDRVSPI_PORT0);

DrvSPI_Enable3WireStartInt

Prototype

```
void DrvSPI_Enable3WireStartInt (
    E_DRVSPI_PORT eSpiPort,
    PFN_DRVSPI_CALLBACK pfnCallback,
    uint32_t u32UserData
);
```

Description

Enable the 3-wire SPI start interrupt of the specified SPI port and install the callback function. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

u16Port [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

pfnCallback [in]

The callback function of the corresponding SPI interrupt.

u32UserData [in]

The parameter which will be passed to the callback function.

Include

Driver/DrvSPI.h

Return Value

None

Example

/* Enable the 3-wire SPI0 start interrupt and install the callback function. The parameter 0 will be passed to the callback function. */

DrvSPI_Enable3WireStartInt (eDRVSPI_PORT0, SPI0_Callback, 0);

DrvSPI_Disable3WireStartInt

Prototype

```
void DrvSPI_Disable3WireStartInt (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Disable the 3-wire SPI start interrupt. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Disable the 3-wire SPI0 start interrupt */
DrvSPI_Disable3WireStartInt (eDRVSPI_PORT0);
```

DrvSPI_Get3WireStartIntFlag

Prototype

```
uint32_t DrvSPI_Get3WireStartIntFlag (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Get the 3-wire SPI start interrupt status. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

0: the SPI start interrupt doesn't occur.

1: the SPI start interrupt occurs.

Example

/* Get the 3-wire SPI0 start interrupt flag */

DrvSPI_Get3WireStartIntFlag (eDRVSPI_PORT0);

DrvSPI_Clr3WireStartIntFlag

Prototype

```
void DrvSPI_Clr3WireStartIntFlag (
    E_DRVSPI_PORT eSpiPort
);
```

Description

Clear the 3-wire SPI start interrupt status. Only the chips with the part number NUC1x0xxxCx, ex: NUC140VE3CN, can support this function.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

Include

Driver/DrvSPI.h

Return Value

None.

Example

```
/* Clear the 3-wire SPI0 start interrupt flag */
DrvSPI_Clr3WireStartIntFlag (eDRVSPI_PORT0);
```

DrvSPI_GetVersion

Prototype

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

Description

Get the version number of SPI driver.

Include

Driver/DrvSPI.h

Return Value

Version number:

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

Example

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

8. I2C Driver

8.1. I2C Introduction

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

For NuMicro™ NUC100 Series, I2C device could act as master or slave and I2C driver can help user to use I2C functions easily.

8.2. I2C Feature

The I2C includes following features:

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

8.3. Type Definition

E_I2C_PORT

Enumeration identifier	Value	Description
I2C_PORT0	0	I2C port 0
I2C_PORT1	1	I2C port 1

E_I2C_CALLBACK_TYPE

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

8.4. Functions

DrvI2C_Open

Prototype

int32_t DrvI2C_Open (E_I2C_PORT port, uint32_t u32BusClock);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

u32BusClock [in]

To configure I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
/* Enable I2C0 and set I2C0 bus clock 100 KHz */
DrvI2C_Open (I2C_PORT0, 100000);
```

DrvI2C_Close

Prototype

int32_t DrvI2C_Close (E_I2C_PORT port);

Description

To close the I2C hardware.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

DrvI2C_Close (I2C_PORT0); /* Disable I2C0 */

DrvI2C_SetClockFreq

Prototype

int32_t DrvI2C_SetClockFreq (E_I2C_PORT port, uint32_t u32BusClock);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

u32BusClock [in]

To configure I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

/* Set I2C0 bus clock 200 KHz */

DrvI2C_SetClockFreq (I2C_PORT0, 200000);

DrvI2C_GetClockFreq

Prototype

uint32_t DrvI2C_GetClockFreq (E_I2C_PORT port);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

I2C bus clock

Example

```
uint32_t    u32clock;
u32clock = DrvI2C_GetClockFreq (I2C_PORT0);      /* Get I2C0 bus clock */
```

DrvI2C_SetAddress

Prototype

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

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

slaveNo [in]

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

slave_addr [in]

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

GC_Flag [in]

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

Include

Driver/DrvI2C.h

Return Value

0: Succeed

<0: Failed

Example

```
DrvI2C_SetAddress(I2C_PORT0, 0, 0x15, 0);    /* Set I2C0 1st slave address 0x15 */
DrvI2C_SetAddress(I2C_PORT0, 1, 0x35, 0);    /* Set I2C0 2nd slave address 0x35 */
DrvI2C_SetAddress(I2C_PORT0, 2, 0x55, 0);    /* Set I2C0 3rd slave address 0x55 */
DrvI2C_SetAddress(I2C_PORT0, 3, 0x75, 0);    /* Set I2C0 4th slave address 0x75 */
```

DrvI2C_SetAddressMask

Prototype

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

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

slaveNo [in]

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

slaveAddrMask [in]

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

Include

Driver/DrvI2C.h

Return Value

0: Succeed

<0: Failed

Example

```
DrvI2C_SetAddress (I2C_PORT0, 0, 0x15, 0);      /* Set I2C0 1st slave address 0x15 */
DrvI2C_SetAddress (I2C_PORT0, 1, 0x35, 0);      /* Set I2C0 2nd slave address 0x35 */
/* Set I2C0 1st slave address mask 0x01, slave address 0x15 and 0x14 would be addressed */
DrvI2C_SetAddressMask (I2C_PORT0, 0, 0x01);
/* Set I2C0 2nd slave address mask 0x04, slave address 0x35 and 0x31 would be addressed */
DrvI2C_SetAddressMask (I2C_PORT0, 1, 0x04);
```

DrvI2C_GetStatus

Prototype

uint32_t DrvI2C_GetStatus (E_I2C_PORT port);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

I2C status code

Example

```
uint32_t    u32status;
u32status = DrvI2C_GetStatus (I2C_PORT0);      /* Get I2C0 current status code */
```

DrvI2C_WriteData

Prototype

void DrvI2C_WriteData(E_I2C_PORT port, uint8_t u8data);

Description

To set a byte of data to be sent.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

u8data [in]

Byte data.

Include

Driver/DrvI2C.h

Return Value

None

Example

```
DrvI2C_WriteData (I2C_PORT0, 0x55); /* Set byte data 0x55 into I2C0 data register */
```

DrvI2C_ReadData

Prototype

uint8_t DrvI2C_ReadData(E_I2C_PORT port);

Description

To read the last data from I2C bus.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

Last byte data

Example

```
uint8_t    u8data;
u8data = DrvI2C_ReadData (I2C_PORT0); /* Read out byte data from I2C0 data register */
```

DrvI2C_Ctrl

Prototype

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

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

start [in]

To set STA bit or not. (1: set, 0: don't set). If the STA bit is set, a START or repeat START signal will be generated when I2C bus is free.

stop [in]

To set STO bit or not. (1: set, 0: don't set). If the STO bit is set, a STOP signal will be generated. When a STOP condition is detected, this bit will be cleared by hardward automatically.

intFlag [in]

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

ack [in]

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

Include

Driver/DrvI2C.h

Return Value

None

Example

DrvI2C_Ctrl (I2C_PORT0, 0, 0, 1, 0); /* Set I2C0 SI bit to clear SI flag */

DrvI2C_Ctrl (I2C_PORT0, 1, 0, 0, 0); /* Set I2C0 STA bit to send START signal */

DrvI2C_GetIntFlag

Prototype

uint8_t DrvI2C_GetIntFlag(E_I2C_PORT port);

Description

To get I2C interrupt flag status.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

Interrupt status (1 or 0)

Example

uint8_t u8flagStatus;

u8flagStatus = DrvI2C_GetIntFlag (I2C_PORT0); /* Get the status of I2C0 interrupt flag */

DrvI2C_ClearIntFlag

Prototype

void DrvI2C_ClearIntFlag (E_I2C_PORT port);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

None

Example

DrvI2C_ClearIntFlag (I2C_PORT0); /* Clear I2C0 interrupt flag (SI) */

DrvI2C_EnableInt

Prototype

int32_t DrvI2C_EnableInt (E_I2C_PORT port);

Description

To enable I2C interrupt function.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

DrvI2C_EnableInt (I2C_PORT0); /* Enable I2C0 interrupt */

DrvI2C_DisableInt

Prototype

int32_t DrvI2C_DisableInt (E_I2C_PORT port);

Description

To disable I2C interrupt function.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

DrvI2C_DisableInt (I2C_PORT0); /* Disable I2C0 interrupt */

DrvI2C_InstallCallBack

Prototype

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

Description

To install I2C call back function in I2C interrupt handler.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Type [in]

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

I2CFUNC: For normal I2C condition

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

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

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

callbackfn [in]

Call back function name for specified interrupt event.

Include

Driver/DrvI2C.h

Return Value

0: Succeed

<0: Failed

Example

/* Install I2C0 call back function ‘I2C0_Callback_Normal’ for I2C normal condition */

DrvI2C_InstallCallback (I2C_PORT0, I2CFUNC, I2C0_Callback_Normal);

/* Install I2C0 call back function ‘I2C0_Callback_BusErr’ for Bus Error condition */

DrvI2C_InstallCallback (I2C_PORT0, BUSERROR, I2C0_Callback_BusErr);

DrvI2C_UninstallCallBack

Prototype

int32_t DrvI2C_UninstallCallBack (E_I2C_PORT port, E_I2C_CALLBACK_TYPE Type);

Description

To uninstall I2C call back function in I2C interrupt handler.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Type [in]

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

I2CFUNC: For normal I2C condition

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

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

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

Include

Driver/DrvI2C.h

Return Value

0: Succeed

<0: Failed

Example

```
/* Uninstall I2C0 call back function for I2C normal condition */
DrvI2C_UninstallCallBack (I2C_PORT0, I2CFUNC);
/* Uninstall I2C0 call back function for Bus Error condition */
DrvI2C_UninstallCallBack (I2C_PORT0, BUSERROR);
```

DrvI2C_SetTimeoutCounter

Prototype

int32_t DrvI2C_SetTimeoutCounter (E_I2C_PORT port, int32_t i32enable, uint8_t u8div4);

Description

To configure 14-bit time-out counter.

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

i32enable [in]

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

u8div4 [in]

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

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

/* Enable I2C0 14-bit timeout counter and disable its DIV4 function */

DrvI2C_EnableTimeoutCount (I2C_PORT0, 1, 0);

DrvI2C_ClearTimeoutFlag

Prototype

void DrvI2C_ClearTimeoutFlag (E_I2C_PORT port);

Description

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

Parameter

port [in]

Specify I2C interface. (I2C_PORT0 / I2C_PORT1)

Include

Driver/DrvI2C.h

Return Value

None

Example

DrvI2C_ClearTimeoutFlag (I2C_PORT0); /* Clear I2C0 TIF flag */

DrvI2C_GetVersion

Prototype

uint32_t DrvI2C_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Version number:

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

9. RTC Driver

9.1. RTC Introduction

Real Time Clock (RTC) unit provides user the real time and calendar message .The RTC uses a 32.768 KHz external crystal. A built in RTC is designed to generate the periodic interrupt signal. The period can be 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second. And the RTC controller supports periodic Time Tick and Alarm Match interruptst.

9.2. RTC Features

- There is a time counter (second, minute, hour) and calendar counter (day, month, year) for user to check the time.
- 12-hour or 24-hour mode is selectable.
- Leap year compensation automatically.
- Day of week counter.
- Frequency compensate register.
- All time and calendar message is expressed in BCD code.
- Support periodic time tick interrupt with 8 period options 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second.
- Support RTC Time Tick and Alarm Match interrupt
- Support wake-up chip from power down mode by RTC Time Tick or Alarm Match interrupt.

Constant Definition

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

9.3. Type Definition

E_DRVRTC_INT_SOURCE

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

E_DRVRTC_TICK

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

E_DRVRTC_TIME_SELECT

Enumeration identifier	Value	Description
DRVRTC_CURRENT_TIME	0	Select current time option

DRVRTC_ALARM_TIME	1	Select alarm time option

E_DRVRTC_DWR_PARAMETER

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

9.4. Functions

DrvRTC_SetFrequencyCompensation

Prototype

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

Description

Set Frequency Compensation Data

Parameter

i32FrequencyX100 [in]

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FCR_VALUE: Wrong Compensation value

Example

/* If the measured RTC crystal frequency is 32773.65Hz.*/

DrvRTC_SetFrequencyCompensation (3277365) ;

DrvRTC_IsLeapYear

Prototype

int32_t

DrvRTC_IsLeapYear (void);

Description

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

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

1: This year is a leap year.

0: This year is not a leap year.

Example

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

DrvRTC_GetIntTick

Prototype

int32_t DrvRTC_GetIntTick (void);

Description

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

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

Software Tick Count in tick interrupt

Example

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

DrvRTC_ResetIntTick

Prototype

void DrvRTC_ResetTick (void);

Description

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

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

None

Example

DrvRTC_ResetTick ();

DrvRTC_WriteEnable

Prototype

int32_t

DrvRTC_WriteEnable (void);

Description

Access Password to AER to make access other register enable

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_FAILED : Failed.

Note

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

Example

/* Before you want to set the value in FCR / TAR / CAR / TTR register, using the function to open access account. */

DrvRTC_WriteEnable ();

DrvRTC_Init

Prototype

int32_t DrvRTC_Init (void);

Description

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

Parameter

None.

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO : Initial RTC Failed.

Example

/*In the beginning, call the function to initial RTC */

DrvRTC_Init() ;

DrvRTC_SetTickMode

Prototype

int32_t DrvRTC_SetTickMode(uint8_t ucMode);

Description

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

Parameter

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

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Access Enable failed

E_DRVRTC_ERR_ENOTTY: Parameter is wrong

Example

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

DrvRTC_EnableInt

Prototype

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

Description

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

Parameter

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

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

pfncallback [in] Callback function pointer

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_ENOTTY: Parameter is wrong

Example

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

DrvRTC_DisableInt

Prototype

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

Description

The function is used to disable specified interrupt and remove callback function.

Parameter

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

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_ENOTTY: Parameter is wrong

Example

```
/* Disable tick and alarm interrupt*/

DrvRTC_DisableInt(DRVRTC_ALL_INT);
```

DrvRTC_Open

Prototype

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

Description

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

Parameter

***sPt [in]**

Specify the time property and current time. It includes

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Initial RTC Failed.

Example

```
/* Start RTC count from 2009.Jan.19, 13:20:00 . */
S_DRVRTC_TIME_DATA_T sInitTime;
sInitTime.u32Year          = 2009;
sInitTime.u32cMonth        = 1;
sInitTime.u32cDay          = 19;
sInitTime.u32cHour         = 13;
sInitTime.u32cMinute       = 20;
sInitTime.u32cSecond       = 0;
sInitTime.u32cDayOfWeek = DRVRTC_MONDAY;
sInitTime.u8cClockDisplay = DRVRTC_CLOCK_24;
if (DrvRTC_Open(&sInitTime) !=E_SUCCESS)
{
    printf("RTC Open Fail!!\n");
}
```

DrvRTC_Read

Prototype

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

Description

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

Parameter

eTime [in]

Specify the current/alarm time to be read.

DRVRTC_CURRENT_TIME	: Current time
DRVRTC_ALARM_TIME	: Alarm time

***sPt [in]**

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

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Initial RTC Failed.

Example

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

DrvRTC_Read(DRVRTC_CURRENT_TIME, &sCurTime);
```

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

DrvRTC_Write

Prototype

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

Description

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

Parameter

eTime [in]

Specify the current/alarm time to be written.

DRVRTC_CURRENT_TIME : Current time

DRVRTC_ALARM_TIME : Alarm time

***sPt [in]**

Specify the data to write to RTC. It includes:

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

E_DRVRTC_ERR_EIO: Initial RTC Failed.

Example

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

DrvRTC_Read(DRVRTC_ALARM_TIME, &sCurTime);

sCurTime.u32cSecond = 0;

DrvRTC_Write(DRVRTC_ALARM_TIME, &sCurTime);
```

DrvRTC_Close

Prototype

int32_t

DrvRTC_Close (void);

Description

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

Include

Driver/DrvRTC.h

Return Value

E_SUCCESS: Success

Example

DrvRTC_Close();

DrvRTC_GetVersion

Prototype

int32_t

DrvRTC_GetVersion (void);

Description

Return the current version number of driver.

Include

Driver/DrvRTC.h

Return Value

Version number :

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

10. CAN Driver

10.1. CAN Introduction

The CAN Core performs communication according to the CAN protocol version 2.0 part A and B. The bit rate can be programmed to values up to 1MBit/s. For the connection to the physical layer, additional transceiver hardware is required.

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM. Only NuMicro™ 130/140 support the CAN application.

10.2. CAN Feature

Its main features are listed as following:

- Supports CAN protocol version 2.0 part A and B.
- Bit rates up to 1 MBit/s.
- 32 Message Objects.
- Each Message Object has its own identifier mask.
- Programmable FIFO mode (concatenation of Message Objects).
- Maskable interrupt.
- Disabled Automatic Re-transmission mode for Time Triggered CAN applications.
- Programmable loop-back mode for self-test operation.
- 16-bit module interfaces to the AMBA APB bus.
- Support wakeup function.

10.3. Constant Definition

Table 10-1: Callback function

Name	Value	Description
CALLBACK_RXOK	**0**	**RX OK Callback function pointer**
CALLBACK_TXOK	**1**	**TX OK Callback function pointer**
CALLBACK_EWARN	**2**	**Warning Callback function pointer**

Name	Value	Description
CALLBACK_BOFF	3	Bus Off Callback function pointer
CALLBACK_MSG	4	Message Callback function pointer
CALLBACK_WAKEUP	5	Wakeup Callback function pointer

Table 10-2: ID Type

Name	Value	Description
CAN_STD_ID	0	Standard ID (11-bits)
CAN_EXT_ID	1	Extended ID (29-bits)

Table 10-3: Frame Type

Name	Value	Description
REMOTE_FRAME	0	Remote Frame
DATA_FRAME	1	Data Frame

10.4. Functions

DrvCAN_Init

Prototype

void DrvCAN_Init(void);

Description

The function is used to reset and Initializes CAN IP

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Enable CAN IP clock */
DrvCAN_Init();
```

DrvCAN_Close

Prototype

void DrvCAN_Close(void);

Description

Reset and clear all CAN control and disable CAN IP

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Disable CAN IP clock , clear callback function pointer and reset CAN IP*/
DrvCAN_Close();
```

DrvCAN_Open

Prototype

int32_t DrvCAN_Open(uint32_t u32kbps);

Description

The function is used to set bus timing parameter according current clock and target bit rate.

Parameter

u32kbps [in]

The target CAN kilo bit rate per second.

The range of u32kbps is 1~1000Kbps.

Include

Driver/DrvCAN.h

Return Value

E_DRVCAN_ERR_BITRATE	Set target bit-rate fail
E_SUCCESS	Set bitrate successful.

Example

```
/* Set CAN bitrate is 500kbps */
DrvCAN_Open(500);
```

DrvCAN_SetTiming

Prototype

void DrvCAN_SetTiming(uint8_t u8Tseg2, uint8_t u8Tseg1, uint8_t u8Sjw, uint32_t u32Brp);

Description

Setups the CAN timing with specific parameters.

Parameter

u8Tseg1 [in]

specifies Time Segment before the sample point. This parameter must be a number between 1 and 16.

u8Tseg2 [in]

Time Segment after the sample point. This parameter must be a number between 1 and 8.

u8Sjw [in]

Synchronisation Jump Width. This parameter must be a number between 1 and 4.

u32Brp [in]

Baud Rate Prescaler. This parameter must be a number between 1 and 1024

Shown CAN bit-rate calculation equation as below:

$$\text{CAN speed (bps)} = = \frac{f_{APB\_CLK}}{(u8Tseg1 + u8Tseg2 + 3) \times (u32Bpr + 1)}$$

Whrer f_{APB_CLK} : System clock freq.

u8Tseg1 : The time segment 1

u8Tseg2: The time segment 2

u32Bpr : the baud-rate prescale

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Set CAN Bus timing according your desired. T2= 2, T1= 3,SJW=1, BRP =1*/
DrvCAN_EnterInitMode();
DrvCAN_SetTiming(2,3,1,1);
```

DrvCAN_LeaveInitMode();

If the system clock freq = 16MHz, so

CAN bit-rate = $\frac{16000000}{(2+3+3)\times(1+1)}$ = 1000kbps

DrvCAN_ResetMsgObj

Prototype

void DrvCAN_ResetMsgObj (uint8_t u8MsgObj);

Description

Configures the message object as default.

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: SUCCESS

E_DRVCAN_NO_USEFUL_INTERFACE: No useful interface

Example

/* Reset CAN Message Object No.5 information*/

DrvCAN_ResetMsgObj (5);

DrvCAN_ResetAllMsgObj

Prototype

void DrvCAN_ResetAllMsgObj (void);

Description

Configures all the message objects as default.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None.

Example

/* Reset all CAN Message Object */

DrvCAN_ ResetAllMsgObj ();

DrvCAN_SetTxMsgObj

Prototype

int32_t DrvCAN_SetTxMsgObj(uint8_t u8MsgObj, STR_CANMSG_T* pCanMsg);

Description

The function is used to configure a transmit object.

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31

pCanMsg [in]

A structure about CAN message object

idType:

specifies the identifier type of the frames that will be transmitted. using this message object. This parameter can be one of the following values:

- CAN_STD_ID (standard ID, 11-bit)
- CAN_EXT_ID (extended ID, 29-bit)

FrameType: DATA_FRAME or REMOTE_FRAME

Id: specifies the identifier used for acceptance filtering

Dlc: Desird data bytes you want to send. Maximun is 8.

Data[0]~ Data[7]: Data value

◆

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: SUCCESS

E_DRVCAN_NO_USEFUL_INTERFACE: No useful interface

Example

/* Configure tMsg structure content into Message Object 0 */

STR_CANMSG_T tMsg;

/* Send a 11-bits message */

```
tMsg.FrameType= DATA_FRAME;
tMsg.IdType     = CAN_STD_ID;
tMsg.Id         = 0x7FF;
tMsg.Dlc        = 0;
if(DrvCAN_SetTxMsgObj(MSG(0),&tMsg) < 0)
    printf("Set Tx Msg Object failed\n");
```

DrvCAN_SetMsgObjMask

Prototype

int32_t DrvCAN_SetMsgObjMask(uint8_t u8MsgObj,STR_CANMASK_T* MaskMsg);

Description

Configures Mask as the message object .

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

MaskMsg [in]

specifies the mask structure as message object.

The structure is including of

- u8Xtd (Mask IDE bit)
- u8Dir (Mask Direction)
- u32Id (Mask ID bit)
- u8IdType (Mask ID Type)

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: SUCCESS

E_DRVCAN_NO_USEFUL_INTERFACE: No useful interface

Example

```
/* Set CAN Message Object No.0 Mask ID is 0x7FF */
STR_CANMASK_T tMsg;
tMsg. u32Id      = 0x7FF;
if(DrvCAN_SetMsgObjMask(0, &tMsg); < 0)
    printf("Set Msg Object failed\n");
```

DrvCAN_SetRxMsgObj

Prototype

int32_t DrvCAN_SetRxMsgObj(uint8_t u8msgobj, uint8_t u8idType, uint32_t u32id, uint8_t u8singleOrFifoLast);

Description

The function is used to configure a receive message object..

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

idType [in]

specifies the identifier type of the frames that will be transmitted

using this message object. This parameter can be one of the following values:

- CAN_STD_ID (standard ID, 11-bit)

- CAN_EXT_ID (extended ID, 29-bit)

u32id [in]

specifies the identifier used for acceptance filtering

u8singleOrFifoLast [in]

specifies the end-of-buffer indicator.

This parameter can be one of the following values:

- TRUE: for a single receive object or a FIFO receive object that is the last one of the FIFO.

- FALSE: for a FIFO receive object that is not the last one

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: SUCCESS

E_DRVCAN_NO_USEFUL_INTERFACE: No useful interface

Example

```
/* Configure CAN Message Object No.0 only receive ID 0x123 */
STR_CANMSG_T rMsg;
if(DrvCAN_SetRxMsgObj(MSG(0),CAN_STD_ID, 0x123,TRUE) < 0)
    printf("Set Rx Msg Object failed\n");
```

DrvCAN_ClrIntPnd

Prototype

int32_t DrvCAN_ClrIntPnd (uint8_t u8msgobj);

Description

The function is used to reset IntPnd and TXRQSTNEWDAT bit in a Message Object.

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: SUCCESS

E_DRVCAN_NO_USEFUL_INTERFACE: No useful interface

Example

/* Clear CAN Message Object 0 interrupt pending */

DrvCAN_ ClrIntPnd (0);

DrvCAN_SetTxRqst

Prototype

uint32_t DrvCAN_SetTxRqst (uint8_t u8MsgObj);

Description

The function is used to set transmit request bit in the target message object.

Parameters

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: SUCCESS

Example

/* After call DrvCAN_SetTxMsg () to set up your message content into target message object , you can call this API and let Message Handler to send this message*/

/*Set the TxRqst bit of Message object No.0*/

DrvCAN_SetTxRqst (0);

DrvCAN_ReadMsgObj

Prototype

int32_t DrvCAN_ReadMsgObj(uint8_t u8MsgObj, uint8_t release, STR_CANMSG_T* pCanMsg);

Description

Gets the message, if received.

Parameters

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

u8Release [in]

specifies the message release indicator.

This parameter can be one of the following values:

- TRUE: the message object is released when getting the data.

- FALSE: the message object is not released.

pCanMsg [in]

pointer to the message structure where received data is copied.

Include

Driver/DrvCAN

Return Value

- E_SUCCESS: Success

- E_DRVCAN_NO_PENDING_MSG: No any message received

Example

/* Polling IIDR flag to wait specified message object status changed and receive information is stored as rMsg structure.*/

while(CAN->u32IIDR ==0);   /* Wait IIDR is changed */

DrvCAN_ ReadMsgObj (CAN->u32IIDR -1,TRUE,&rMsg);

DrvCAN_WaitEndOfTx

Prototype

int32_t DrvCAN_WaitEndOfTx(void);

Description

Waiting until current transmission is finished

Parameters

None

Include

Driver/DrvCAN.h

Return Value

- E_SUCCESS: Transmission ended

Example

```
/* Wait.Transmit OK*/
DrvCAN_WaitEndOfTx();
printf("Transmit successfully");
```

DrvCAN_BasicSendMsg

Prototype

int32_t DrvCAN_BasicSendMsg(STR_CANMSG_T* pCanMsg);

Description

The function is used to send CAN message in BASIC mode of test mode. Before call the API, the user should be call DrvCAN_EnterTestMode(CAN_TESTR_BASIC) and let CAN controller enter basic mode of test mode. Please notice IF1 Registers used as Tx Buffer in basic mode

Parameter

pCanMsg [in]

Pointer to the message structure containing data to transmit..

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: Transmission OK

E_DRVCAN_ERR_TIMEOUT: Check busy flag of interface 0 is timeout

Example

```
/* Use basic mode to send message without using message ram*/
STR_CANMSG_T msg1;
msg1.FrameType= DATA_FRAME;
msg1.IdType     = CAN_STD_ID;
msg1.Id         = 0x555;
msg1.Dlc        = 0;
```

DrvCAN_BasicSendMsg(&msg1);

DrvCAN_BasicReceiveMsg

Prototype

int32_t DrvCAN_BasicReceiveMsg(STR_CANMSG_T* pCanMsg);

Description

Get a message information in BASIC mode. This mode does not use the message RAM Using IF2 to get receive message information

Parameter

pCanMsg [in]

pointer to the message structure where message is copied.

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS: Reception OK

E_DRVCAN_NO_PENDING_MSG: No any message received

Example

/* Wait data in and stored in rmsg structure*/

STR_CANMSG_T rMsg;

DrvCAN_WaitMsg();

DrvCAN_BasicReceiveMsg(&rMsg);

DrvCAN_EnterInitMode

Prototype

void DrvCAN_EnterInitMode(void);

Description

This function is used to set CAN to enter initialization mode and enable access bit timing register. After bit timing configuration ready, user must call DrvCAN_LeaveInitMode()to leave initialization mode and lock bit timing register to let new configuration take effect.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

/* Enter init mode and user can changed bus timing settings.*/

DrvCAN_EnterInitMode();

DrvCAN_LeaveInitMode

Prototype

void DrvCAN_LeaveInitMode(void);

Description

This function is used to set CAN to leave initialization mode to let bit timing configuration take effect after configuration ready.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

/* leave init mode and to let the bit-timing configuration take effect.*/

DrvCAN_LeaveInitMode();

DrvCAN_EnterTestMode

Prototype

void DrvCAN_EnterTestMode(uint8_t u8TestMask);

Description

Switchs the CAN into test mode. There are four test mode (BASIC/SILENT/LOOPBACK/ LOOPBACK combined SILENT/CONTROL_TX_PIN)could be selected. After setting test mode,user must call DrvCAN_LeaveInitMode() to let the setting take effect.

Parameter

u8TestMask [in]

specifies the configuration in test modes

It could be

CAN_TESTR_BASIC : Enable basic mode of test mode

CAN_TESTR_SILENT : Enable silent mode of test mode

CAN_TESTR_LBACK : Enable Loop Back Mode of test mode

CAN_TESTR_TX0 : Set low bit of control CAN_TX pin bit field

CAN_TESTR_TX1 : Set high bit of control CAN_TX pin bit feild

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Enter basic mode of test mode*/
DrvCAN_ EnterTestMode (CAN_TEST_BASIC);
```

DrvCAN_LeaveTestMode

Prototype

void DrvCAN_LeaveTestMode(void);

Description

This function is used to Leaves the current test mode (switch into normal mode)..

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Leave test mode and then enter normal mode    */
DrvCAN_ LeaveTestMode ();
```

DrvCAN_IsNewDataReceived

Prototype

uint32_t DrvCAN_IsNewDataReceived (uint8_t u8MsgObj);

Description

This function is used to get the waiting status of a received message.

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

A non-zero value if the corresponding message object has a new data bit is set, else 0.

Example

```
/* Check message object 0 is no received new message */
if(!DrvCAN_IsNewDataReceived (0);
    return false;
```

DrvCAN_IsTxRqstPending

Prototype

uint32_t DrvCAN_IsTxRqstPending (uint8_t u8MsgObj);

Description

This function is used to get the request pending status of a transmitted message..

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

A non-zero value if the corresponding message has an tx request pending, else 0.

Example

```
/* Check message object 0 transmit request is sent or not */
if(!DrvCAN_IsTxRqstPending (0);
    return false;
```

DrvCAN_IsIntPending

Prototype

uint32_t DrvCAN_IsIntPending(uint8_t u8MsgoObj);

Description

This function is used to get the interrupt status of a message object.

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

A non-zero value if the corresponding message has an interrupt pending, else 0.

Example

```
/* Check message object 0 interrupt is pending or not */
if(!DrvCAN_ IsIntPending (0);
        return false;
```

DrvCAN_IsObjectValid

Prototype

uint32_t DrvCAN_IsObjectValid(uint8_t u8MsgObj);

Description

This function is used to test the validity of a message object (ready to use)..

Parameter

u8MsgObj [in]

specifies the Message object number, from 0 to 31.

Include

Driver/DrvCAN.h

Return Value

A non-zero value if the corresponding message object is valid, else 0.

Example

```
/* Check message object 0 is valied or not */
if(!DrvCAN_ IsObjectValid (0);
        return false;
```

DrvCAN_ResetIF

Prototype

void DrvCAN_ResetIF(uint8_t u8IF_Num);

Description

This function is used to reset message interface parameters..

Parameter

u8IF_Num [in]

specifies the Message Control Interface, 0 or 1

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Reset interface 0 all setting register value */
DrvCAN_ResetIF(0);
```

DrvCAN_WaitMsg

Prototype

void DrvCAN_WaitMsg(void);

Description

This function is used to wait message into message buffer in basic mode. Please notice the function is polling NEWDAT bit of MCON register by while loop and it is used in basic mode.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Wait new message into message ram */
DrvCAN_WaitMsg ();
printf("New Data In\n");
```

DrvCAN_EnableInt

Prototype

int32_t DrvCAN_EnableInt(uint16_t u16IntEnable);

Description

Enable CAN interrupt and NVIC corresponding to CAN.

Parameter

u16IntEnable [in]

Interrupt Enable (CAN_CON_IE or CAN_CON_SIE or CAN_CON_EIE).

It could be

CAN_CON_IE : Module Interrupt Enable

CAN_CON_SIE : Status Change Interrupt Enable

CAN_CON_EIE : Error Interrupt Enable

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS : Success

Example

```
/* Interrupt Enable */
DrvCAN_EnableInt(CAN_CON_IE);
```

DrvCAN_DisableInt

Prototype

int32_t DrvCAN_DisableInt(uint16_t u16IntEnable);

Description

Disable CAN interrupt and NVIC corresponding to CAN.

Parameter

u16IntEnable [in]

Interrupt Enable (CAN_CON_IE or CAN_CON_SIE or CAN_CON_EIE).

CAN_CON_IE : Module interrupt enable

CAN_CON_SIE : Status change interrupt enable

CAN_CON_EIE : Error interrupt enable

Include

Driver/DrvCAN.h

Return Value

None.

Example

/* Interrupt Disable */

DrvCAN_DisableInt(CAN_CON_IE);

DrvCAN_InstallCallback

Prototype

int32_t DrvCAN_InstallCallback(E_CAN_CALLBACK_TYPE Type, CAN_CALLBACK callbackfn);

Description

Install CAN call back function for CAN normal function MSG,RXOK,TXOK,EWARN,BOFF,WAKEUP.

Parameter

Type [in]

E_CAN_CALLBACK_TYPE (CALLBACK_RXOK or CALLBACK_TXOK or CALLBACK_EWARN or CALLBACK_BOFF or CALLBACK_MSG or CALLBACK_WAKEUP). More detail please ref Table 10.1

callback [in]

callback function pointer

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS : Success

E_ E_DRVCAN_ERR_ARGUMENT: Failed

Example

/* Install Message callback function "TestFnMsg" */

DrvCAN_InstallCallback(CALLBACK_MSG, (CAN_CALLBACK)TestFnMsg);

DrvCAN_UninstallCallback

Prototype

int32_t DrvCAN_UninstallCallback(E_CAN_CALLBACK_TYPE Type);

Description

The function is used to uninstall exist callback function pointer.

Parameter

Type [in]

E_CAN_CALLBACK_TYPE (CALLBACK_RXOK or CALLBACK_TXOK or CALLBACK_EWARN or CALLBACK_BOFF or CALLBACK_MSG or CALLBACK_WAKEUP). More detail please ref Table 10.1

Include

Driver/DrvCAN.h

Return Value

E_SUCCESS : Success

E_ E_DRVCAN_ERR_ARGUMENT: Failed

Example

/* Remove all message object callback function pointer */

DrvCAN_UninstallCallback(CALLBACK_MSG);

DrvCAN_EnableWakeUp

Prototype

void DrvCAN_EnableWakeUp(void);

Description

The function is used to enable wakeup function.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

/* Enable wake-up function */

DrvCAN_EnableWakeUp();

DrvCAN_DisableWakeUp

Prototype

void DrvCAN_DisableWakeUp(void);

Description

The function is used to disable wakeup function.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

None

Example

```
/* Disable wake-up function */
DrvCAN_DisableWakeUp()
```

DrvCAN_GetCANBitRate

Prototype

int32_t DrvCAN_GetCANBitRate(void);

Description

Return current CAN bitrate according to user bit-timing parameter settings.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

Current Bit-Rate (kilo bit per second)

Example

```
/* Get current CAN bit rate */
int32 i32bitrate;
i32bitrate    = DrvCAN_GetCANBitRate ();
```

DrvCAN_GetTxErrCount

Prototype

uint32_t DrvCAN_GetTxErrCount(void);

Description

The function is used to get current transmit error counter (TEC)

Parameter

None

Include

Driver/DrvCAN.h

Return Value

Current Transmit Error Counter(TEC)

Example

```
/* Get current transmit error counter(TEC) */
int32 i32TxErrCnt
i32TxErrCnt    = DrvCAN_ GetTxErrCount ();
```

DrvCAN_GetRxErrCount

Prototype

uint32_t DrvCAN_GetRxErrCount(void);

Description

The function is used to get current receive error counter (REC)

Parameter

None

Include

Driver/DrvCAN.h

Return Value

Current Receive Error Counter(REC)

Example

```
/* Get current receive error counter(REC) */
int32 i32RxErrCnt
i32RxErrCnt    = DrvCAN_ GetRxErrCount ();
```

DrvCAN_GetVersion

Prototype

uint32_t DrvCAN_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvCAN.h

Return Value

CAN driver current version number:

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

Example

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

int32_t i32CANVersionNum      ;

i32CANVersionNum = DrvCAN_GetVersion();
```

11. PWM Driver

11.1. PWM Introduction

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

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

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

11.2. PWM Features

The PWM controller includes following features:

- Up to two PWM group (PWMA/PWMB). Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix to know the number of PWM group.
- Each PWM group has two PWM generators. Each PWM generator supports one 8-bit prescaler, one clock divider, two PWM-timers (down counter), one dead-zone generator and two PWM outputs.
- One-shot or Auto-reload PWM mode.
- Up to eight capture input channels.
- Each capture input channel supports rising/falling latch register and capture interrupt flag.

11.3. Constant Definition

Constant Name	Value	Description
DRVPWM_TIMER0	0x00	PWM Timer 0
DRVPWM_TIMER1	0x01	PWM Timer 1
DRVPWM_TIMER2	0x02	PWM Timer 2
DRVPWM_TIMER3	0x03	PWM Timer 3
DRVPWM_TIMER4	0x04	PWM Timer 4
DRVPWM_TIMER5	0x05	PWM Timer 5
DRVPWM_TIMER6	0x06	PWM Timer 6
DRVPWM_TIMER7	0x07	PWM Timer 7
DRVPWM_CAP0	0x10	PWM Capture 0
DRVPWM_CAP1	0x11	PWM Capture 1
DRVPWM_CAP2	0x12	PWM Capture 2
DRVPWM_CAP3	0x13	PWM Capture 3
DRVPWM_CAP4	0x14	PWM Capture 4
DRVPWM_CAP5	0x15	PWM Capture 5
DRVPWM_CAP6	0x16	PWM Capture 6
DRVPWM_CAP7	0x17	PWM Capture 7
DRVPWM_CAP_ALL_INT	3	PWM Capture Rising and Falling Interrupt
DRVPWM_CAP_RISING_INT	1	PWM Capture Rising Interrupt
DRVPWM_CAP_FALLING_INT	2	PWM Capture Falling Interrupt
DRVPWM_CAP_RISING_FLAG	6	Capture rising interrupt flag
DRVPWM_CAP_FALLING_FLAG	7	Capture falling interrupt flag
DRVPWM_CLOCK_DIV_1	4	Input clock divided by 1
DRVPWM_CLOCK_DIV_2	0	Input clock divided by 2
DRVPWM_CLOCK_DIV_4	1	Input clock divided by 4
DRVPWM_CLOCK_DIV_8	2	Input clock divided by 8
DRVPWM_CLOCK_DIV_16	3	Input clock divided by 16
DRVPWM_EXT_12M	0	PWM clock source from external 12M crystal
DRVPWM_EXT_32K	1	PWM clock source from external 32K crystal
DRVPWM_HCLK	2	PWM clock source from HCLK
DRVPWM_INTERNAL_22M	3	PWM clock source from internal 22MHz oscillator
DRVPWM_AUTO_RELOAD_MODE	1	PWM Timer auto-reload mode
DRVPWM_ONE_SHOT_MODE	0	PWM Timer One-shot mode

11.4. Functions

DrvPWM_IsTimerEnabled

Prototype

int32_t DrvPWM_IsTimerEnabled(uint8_t u8Timer);

Description

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

Parameter

u8Timer [in]

Specify the timer.
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

Include

Driver/DrvPWM.h

Return Value

1: The specified timer is enabled.

0: The specified timer is disabled.

Example

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

DrvPWM_SetTimerCounter

Prototype

void DrvPWM_SetTimerCounter(uint8_t u8Timer, uint16_t u16Counter);

Description

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

Parameter

u8Timer [in]

Specify the timer.
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

u16Counter [in]

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

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Set 10000 to PWM timer 3 counter register. When the PWM timer 3 start to count down,
PWM timer 3 will count down from 10000 to 0. If PWM timer 3 is set to auto-reload mode,
the PWM timer 3 will reload 10000 to PWM timer 3 counter register after PWM timer 3
count down to 0 and PWM timer 3 will continue to count down from 10000 to 0 again. */
DrvPWM_SetTimerCounter(DRVPWM_TIMER3, 10000);
```

DrvPWM_GetTimerCounter

Prototype

uint32_t DrvPWM_GetTimerCounter(uint8_t u8Timer);

Description

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

Parameter

u8Timer [in]

Specify the timer.
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.

DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

Include

Driver/DrvPWM.h

Return Value

The specified timer counter value.

Example

```
/* Get PWM timer 5 counter value. */

uint32_t u32RetValTimer5CounterValue;
u32RetValTimer5CounterValue = DrvPWM_GetTimerCounter(DRVPWM_TIMER5);
```

DrvPWM_EnableInt

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

u8Int [in]

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

DRVPWM_CAP_RISING_INT : The capture rising interrupt.
DRVPWM_CAP_FALLING_INT : The capture falling interrupt.
DRVPWM_CAP_ALL_INT : All capture interrupt.

pfncallback [in]

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

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Enable PWM capture 5 falling edge interrupt and install DRVPWM_CapIRQHandler() as it's interrupt callback function.*/

DrvPWM_EnableInt(DRVPWM_CAP5, DRVPWM_CAP_FALLING_INT, DRVPWM_CapIRQHandler);

DrvPWM_DisableInt

Prototype

void DrvPWM_DisableInt(uint8_t u8Timer);

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Disable PWM capture 5 interrupts including rising and falling interrupt source and also uninstall PWM capture 5 rising and falling interrupt callback functions. */

DrvPWM_DisableInt(DRVPWM_CAP5);

/* Disable PWM timer 5 interrupt and uninstall PWM timer 5 callback function.*/

DrvPWM_DisableInt(DRVPWM_TIMER5);

DrvPWM_ClearInt

Prototype

void DrvPWM_ClearInt(uint8_t u8Timer);

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Clear PWM timer 1 interrupt flag.*/

DrvPWM_ClearInt(DRVPWM_TIMER1);

/* Clear PWM capture 0 interrupt flag. */

DrvPWM_ClearInt(DRVPWM_CAP0);

DrvPWM_GetIntFlag

Prototype

int32_t DrvPWM_GetIntFlag(uint8_t u8Timer);

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

Include

Driver/DrvPWM.h

Return Value

1: The specified interrupt occurs.

0: The specified interrupt doesn’t occur.

Example

/* Get PWM timer 6 interrupt flag.*/

```
if(DrvPWM_GetIntFlag(DRVPWM_TIMER6)==1)
printf(”PWM timer 6 interrupt occurs!\n);
else if(DrvPWM_GetIntFlag(DRVPWM_TIMER6)==0)
printf(”PWM timer 6 interrupt dosen’t occur!\n);
```

DrvPWM_GetRisingCounter

Prototype

uint16_t DrvPWM_GetRisingCounter(uint8_t u8Capture);

Description

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

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

Include

Driver/DrvPWM.h

Return Value

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

Example

```
/* Get PWM capture 7 rising latch register value. */

uint16_t u16RetValTimer7RisingLatchValue;

u16RetValTimer7RisingLatchValue = DrvPWM_GetRisingCounter (DRVPWM_CAP7);
```

DrvPWM_GetFallingCounter

Prototype

uint16_t DrvPWM_GetFallingCounter(uint8_t u8Capture);

Description

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

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

Include

Driver/DrvPWM.h

Return Value

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

Example

```
/* Get PWM capture 7 falling latch register value.*/

uint16_t u16RetValTimer7FallingLatchValue;

u16RetValTimer7FallingLatchValue = DrvPWM_GetFallingCounter (DRVPWM_CAP7);
```

DrvPWM_GetCaptureIntStatus

Prototype

int32_t DrvPWM_GetCaptureIntStatus(uint8_t u8Capture, uint8_t u8IntType);

Description

Check if there's a rising / falling transition

Parameter

u8Capture [in]

Specify the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

u8IntType [in]

Specify the Capture Latched Indicator.

DRVPWM_CAP_RISING_FLAG : The capture rising indicator flag.

DRVPWM_CAP_FALLING_FLAG : The capture falling indicator flag.

Include

Driver/DrvPWM.h

Return Value

TRUE: The specified transition occurs.

FALSE: The specified transition doesn't occur.

Example

```
/* Get PWM capture 5 rising transition flag.*/
if(DrvPWM_GetCaptureIntStatus(DRVPWM_CAP5, DRVPWM_CAP_RISING_FLAG)==TRUE)
printf("PWM capture 5 rising transition occurs!\n");
else if(DrvPWM_GetCaptureIntStatus(DRVPWM_CAP5, DRVPWM_CAP_RISING_FLAG)==FALSE)
printf("PWM capture 5 rising transition doesn't occur!\n");
```

DrvPWM_ClearCaptureIntStatus

Prototype

void DrvPWM_ClearCaptureIntStatus(uint8_t u8Capture, uint8_t u8IntType);

Description

Clear the rising / falling transition indicator flag

Parameter

u8Capture [in]

Specify the capture.

DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

u8IntType [in]

Specify the Capture Latched Indicator.

DRVPWM_CAP_RISING_FLAG : The capture rising indicator flag.

DRVPWM_CAP_FALLING_FLAG : The capture falling indicator flag.

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Clear PWM capture 5 falling transition flag.*/

DrvPWM_ClearCaptureIntStatus(DRVPWM_CAP5, DRVPWM_CAP_FALLING_FLAG);

DrvPWM_Open

Prototype

void DrvPWM_Open(void);

Description

Enable PWM engine clock and reset PWM.

Include

Driver/DrvPWM.h

Return Value

None

Example

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

DrvPWM_Open();

DrvPWM_Close

Prototype

void DrvPWM_Close(void);

Description

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

Include

Driver/DrvPWM.h

Return Value

None

Example

/* Disable PWM timer 0~7 output, PWM capture 0~7 output and disable PWM engine clock.*/

DrvPWM_Close ( );

DrvPWM_EnableDeadZone

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer

DRVPWM_TIMER0 or DRVPWM_TIMER1: PWM timer 0 & PWM timer 1.
DRVPWM_TIMER2 or DRVPWM_TIMER3: PWM timer 2 & PWM timer 3.
DRVPWM_TIMER4 or DRVPWM_TIMER5: PWM timer 4 & PWM timer 5.
DRVPWM_TIMER6 or DRVPWM_TIMER7: PWM timer 6 & PWM timer 7.

u8Length [in]

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

i32EnableDeadZone [in]

Enable DeadZone (1) / Diasble DeadZone (0)

Include

Driver/DrvPWM.h

Return Value

None

Example

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

uint8_t u8DeadZoneLength = 100;

DrvPWM_EnableDeadZone (DRVPWM_TIMER0, u8DeadZoneLength, 1);

Sample code

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

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

/* PWM Timer property */

sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;

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

sPt.i32Inverter = 0;

```
sPt.u32Duty = 1000;
sPt.u8ClockSelector = DRVPWM_CLOCK_DIV_1;
sPt.u8PreScale = 6;
u8Timer = DRVPWM_TIMER0;
/* Select PWM engine clock source */
DrvPWM_SelectClockSource(u8Timer, DRVPWM_EXT_12M);
/* Set PWM Timer0 Configuration */
DrvPWM_SetTimerClk(u8Timer, &sPt);
/* Enable Output for PWM Timer0 */
DrvPWM_SetTimerIO(u8Timer, 1);
/* Enable Output for PWM Timer1 */
DrvPWM_SetTimerIO(DRVPWM_TIMER1, 1);
/* Enable Timer0 and Time1 dead zone function and Set dead zone length to 10 */
DrvPWM_EnableDeadZone(u8Timer, u8DeadZoneLength, 1);
/* Enable the PWM Timer 0 */
DrvPWM_Enable(u8Timer, 1);
```

DrvPWM_Enable

Prototype

void DrvPWM_Enable(uint8_t u8Timer, int32_t i32Enable);

Description

This function is used to enable PWM timer / capture function

Parameter

u8Timer [in]

Specify the timer

DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.

DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.

DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

i32Enable [in]

Enable (1) / Disable (0)

Include

Driver/DrvPWM.h

Return Value

None

Example

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

DrvPWM_SetTimerClk

Prototype

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

Description

This function is used to configure the frequency/pulse/mode/inverter function. The function will set the frequency property automatically when user set a nonzero frequency value by *u32Frequency*. When the setting of frequency value (*u32Frequency*) is not specified, i.e set to 0, user needs to provide the setting of clock selector, prescale and duty to generate desired frequency.

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.

DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

***sPt [in]**

It includes the following parameter

Parameters	Description
u32Frequency	The timer/capture frequency (Hz)
u8HighPulseRatio	High pulse ratio (1~100)
u8Mode	DRVPWM_ONE_SHOT_MODE / DRVPWM_AUTO_RELOAD_MODE
bInverter	Inverter Enable (1) / Inverter Disable (0)
u8ClockSelector	Clock Selector DRVPWM_CLOCK_DIV_1: PWM input clock is divided by 1 DRVPWM_CLOCK_DIV_2: PWM input clock is divided by 2 DRVPWM_CLOCK_DIV_4: PWM input clock is divided by 4 DRVPWM_CLOCK_DIV_8: PWM input clock is divided by 8 DRVPWM_CLOCK_DIV_16: PWM input clock is divided by 16 (The parameter takes effect when *u32Frequency* = 0)
u8PreScale	Prescale (1 ~ 255). If the u8PreScale is set to 0, the timer will stop The PWM input clock = PWM source clock / (*u8PreScale* + 1) (The parameter takes effect when *u32Frequency* = 0)
u32Duty	Pulse duty (0x1 ~ 0x10000) (The parameter takes effect when *u32Frequency* = 0 or *u8Timer* = DRVPWM_CAP0/DRVPWM_CAP1/DRVPWM_CAP2/DRVPWM_CAP3/DRVPWM_CAP4/DRVPWM_CAP5/DRVPWM_CAP6/DRVPWM_CAP7)

Include

Driver/DrvPWM.h

Return Value

The actual specified PWM frequency (Hz).

Example

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

Method 1:

Fill sPt.u32Frequency = 1000 to determine the waveform frequency and DrvPWM_SetTimerClk() will set the frequency property automatically.

```
/* PWM Timer property */
sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;
sPt.u8HighPulseRatio = 20; /* High Pulse peroid : Total Pulse peroid = 20 : 100 */
sPt.i32Inverter = 0;
sPt.u32Frequency = 1000; // Set 1KHz to PWM timer output frequency
u8Timer = DRVPWM_TIMER0;
/* Select PWM engine clock */
DrvPWM_SelectClockSource(u8Timer, DRVPWM_HCLK);
```

/* Set PWM Timer0 Configuration */

DrvPWM_SetTimerClk(u8Timer, &sPt);

/* Enable Output for PWM Timer0 */

DrvPWM_SetTimerIO(u8Timer, 1);

/* Enable Interrupt Sources of PWM Timer 0 and install call back function */

DrvPWM_EnableInt(u8Timer, 0, DRVPWM_PwmIRQHandler);

/* Enable the PWM Timer 0 */

DrvPWM_Enable(u8Timer, 1);

Method 2:

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

Assume HCLK frequency is 22MHz.

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

/* PWM Timer property */

sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;

sPt.u8HighPulseRatio = 20; /* High Pulse peroid : Total Pulse peroid = 20 : 100 */

sPt.i32Inverter = 0;

sPt.u8ClockSelector = DRVPWM_CLOCK_DIV_1;

sPt.u8PreScale = 22;

sPt.u32Duty = 1000;

u8Timer = DRVPWM_TIMER0;

/* Select PWM engine clock and user must know the HCLK frequency*/

DrvPWM_SelectClockSource(u8Timer, DRVPWM_HCLK);

/* Set PWM Timer0 Configuration */

DrvPWM_SetTimerClk(u8Timer, &sPt);

/* Enable Output for PWM Timer0 */

DrvPWM_SetTimerIO(u8Timer, 1);

/* Enable Interrupt Sources of PWM Timer0 and install call back function */

DrvPWM_EnableInt(u8Timer, 0, DRVPWM_PwmIRQHandler);

/* Enable the PWM Timer 0 */

DrvPWM_Enable(u8Timer, 1);

DrvPWM_SetTimerIO

Prototype

void DrvPWM_SetTimerIO(uint8_t u8Timer, int32_t i32Enable);

Description

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

Parameter

u8Timer [in]

Specify the timer
DRVPWM_TIMER0: PWM timer 0.
DRVPWM_TIMER1: PWM timer 1.
DRVPWM_TIMER2: PWM timer 2.
DRVPWM_TIMER3: PWM timer 3.
DRVPWM_TIMER4: PWM timer 4.
DRVPWM_TIMER5: PWM timer 5.
DRVPWM_TIMER6: PWM timer 6.
DRVPWM_TIMER7: PWM timer 7.

or the capture.
DRVPWM_CAP0: PWM capture 0.
DRVPWM_CAP1: PWM capture 1.
DRVPWM_CAP2: PWM capture 2.
DRVPWM_CAP3: PWM capture 3.
DRVPWM_CAP4: PWM capture 4.
DRVPWM_CAP5: PWM capture 5.
DRVPWM_CAP6: PWM capture 6.
DRVPWM_CAP7: PWM capture 7.

i32Enable [in]

Enable (1) / Disable (0)

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Enable PWM timer 0 output.*/
DrvPWM_SetTimerIO(DRVPWM_TIMER0, 1);
/* Disable PWM timer 0 output. */
DrvPWM_SetTimerIO(DRVPWM_TIMER0, 0);
/* Enable PWM capture 3 input. */
DrvPWM_SetTimerIO(DRVPWM_CAP3, 1);
/* Disable PWM capture timer 3 input
```

DrvPWM_SetTimerIO(DRVPWM_CAP3, 0);

DrvPWM_SelectClockSource

Prototype

void DrvPWM_SelectClockSource(uint8_t u8Timer, uint8_t u8ClockSourceSelector);

Description

This function is used to select PWM0&PWM1, PWM2&PWM3, PWM4&PWM5 and PWM6&PWM7 engine clock source. It means PWM0/1 use one clock source. PWM2/3 can use another clock source and so on. In other words, if user change PWM timer 0 clock source from external 12MHz to internal 22MHz, the clock source of PWM timer 1 will also be changed from external 12MHz to internal 22MHz. Furthermore, it is possible to set the clock source of PWM1 to be external 12MHz and set the clock source of PWM2 to be external 32.768Hz.

Parameter

u8Timer [in]

Specify the timer

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

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

DRVPWM_TIMER4 or DRVPWM_TIMER5: PWM timer 4 & PWM timer 5.

DRVPWM_TIMER6 or DRVPWM_TIMER7: PWM timer 6 & PWM timer 7.

u8ClockSourceSelector [in]

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

Include

Driver/DrvPWM.h

Return Value

None

Example

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

DrvPWM_SelectClockSource(DRVPWM_TIMER0, DRVPWM_HCLK);

Select PWM timer 6 and PWM timer 7 engine clock source from external 12MHz.

DrvPWM_SelectClockSource(DRVPWM_TIMER7, DRVPWM_EXT_12M);

DrvPWM_SelectClearLatchFlagOption

Prototype

int32_t DrvPWM_SelectClearLatchFlagOption (int32_t i32option);

Description

This function is used to select how to clear Capture rising & falling Latch Indicator.

Parameter

i32option [in]

0: Select option to clear the Capture Latch Indicators by writing a '0'.

1: Select option to clear the Capture Latch Indicators by writing a '1'.

Include

Driver/DrvPWM.h

Return Value

0	Succeed
<0	Does NOT support this option

Note

Only NUC1x0xxxBx(Ex: NUC140RD2BN), NUC1x0xxxCx(Ex: NUC140VE3CN) and NUC101 of NuMicro™ NUC100 series support this function.Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

DrvPWM_GetVersion

Prototype

uint32_t DrvPWM_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvPWM.h

Return Value

PWM driver current version number:

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

Example

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

int32_t i32PWMVersionNum ;

i32PWMVersionNum = DrvPWM_GetVersion();
```

12. PS2 Driver

12.1. PS2 Introduction

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

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

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

12.2. PS2 Feature

The PS/2 device controller includes following features:

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

12.3. Constant Defination

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

12.4. Macros

_DRVPS2_OVERRIDE

Prototype

void _DRVPS2_OVERRIDE(bool state);

Description

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

Parameter

state **[in]**

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

Include

Driver/DrvPS2.h

Return Value

None.

Example

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

_DRVPS2_PS2CLK

Prototype

void _DRVPS2_PS2CLK(bool state);

Description

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

Parameter

state **[in]**

Specify PS2CLK line high or low

Include

Driver/DrvPS2.h

Return Value

None.

Note

The macro is meaningful only when DRVPS2_OVERRIDE has been called.

Example

/* Force PS2CLK pin high. */

_DRVPS2_PS2CLK(1);

/* Force PS2CLK pin low. */

_DRVPS2_PS2CLK(0);

_DRVPS2_PS2DATA

Prototype

void _DRVPS2_PS2DATA(bool state);

Description

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

Parameter

state **[in]**

Specify PS2DATA line high or low

Include

Driver/DrvPS2.h

Return Value

None.

Note

The macro is meaningful only when _DRVPS2_OVERRIDE has been called.

Example

/* Force PS2DATApin high. */

_DRVPS2_PS2DATA (1);

/* Force PS2DATA pin low. */

_DRVPS2_PS2DATA (0);

_DRVPS2_CLRFIFO

Prototype

void DRVPS2_CLRFIFO();

Description

The macro is used to clear TX FIFO.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

/* Clear TX FIFO. */

_DRVPS2_CLRFIFO();

_DRVPS2_ACKNOTALWAYS

Prototype

void _DRVPS2_ACKNOTALWAYS();

Description

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

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

/* Enable ackknowlwde NOT always. */

_DRVPS2_ACKNOTALWAYS()

_DRVPS2_ACKALWAYS

Prototype

void _DRVPS2_ACKALWAYS();

Description

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

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable ackknowlwde always. */
_DRVPS2_ACKALWAYS()
```

_DRVPS2_RXINTENABLE

Prototype

void _DRVPS2_RXINTENABLE();

Description

The macro is used to enable Rx interrupt. When acknowledge bit is sent from host to device, RX interrupt will happen

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable RX interrupt. */
_DRVPS2_RXINTENABLE();
```

_DRVPS2_RXINTDISABLE

Prototype

void _DRVPS2_RXINTDISABLE();

Description

The macro is used to disable Rx interrupt.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Disable RX interrupt. */
_DRVPS2_RXINTDISABLE ();
```

_DRVPS2_TXINTENABLE

Prototype

void _DRVPS2_TXINTENABLE();

Description

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

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable TX interrupt. */
_DRVPS2_TXINTENABLE();
```

_DRVPS2_TXINTDISABLE

Prototype

void _DRVPS2_TXINTDISABLE ();

Description

The macro is used to disable TX interrupt.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Disable TX interrupt. */
_DRVPS2_TXINTDISABLE();
```

_DRVPS2_PS2ENABLE

Prototype

void _RVPS2_PS2ENABLE ();

Description

The macro is used to enable PS/2 device controller.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Enable PS/2 device controller. */
_DRVPS2_PS2ENABLE ();
```

_DRVPS2_PS2DISABLE

Prototype

void _DRVPS2_PS2DISABLE ();

Description

The macro is used to disable PS/2 device controller.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Disable PS/2 device controller. */
_DRVPS2_PS2DISABLE ();
```

_DRVPS2_TXFIFO

Prototype

void _DRVPS2_TXFIFO(depth);

Description

The macro is used to set TX FIFO depth. The range of TX FIFO is [1,16]

Parameter

data [in] : Specify TX FIFO depth(1~16).

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Set TX FIFO depth to 16 bytes. */
_DRVPS2_TXFIFO(16);
/* Set TX FIFO depth to 1 bytes. */
_DRVPS2_TXFIFO(1);
```

_DRVPS2_SWOVERRIDE

Prototype

void _DRVPS2_SWOVERRIDE(bool data,   bool clk);

Description

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

_DRVPS2_PS2DATA(data);

_DRVPS2_PS2CLK(clk);

_DRVPS2_OVERRIDE(1);

Parameter

data [in]

Specify PS2DATA line high or low

clk [in]

Specify PS2CLK line high or low

Include

Driver/DrvPS2.h

Return Value

None.

Example

/* Set PS2DATA to high and set PS2CLK to low. */

_DRVPS2_SWOVERRIDE(1, 0);

/* Set PS2DATA to low and set PS2CLK to high. */

_DRVPS2_SWOVERRIDE(0, 1);

_DRVPS2_INTCLR

Prototype

void _DRVPS2_INTCLR(uint8_t intclr) ;

Description

The macro is used to clear interrup status.

Parameter

intclr [in]

Specify to clear TX or RX interrupt. Intclr=0x1 for clear RX interrupt; Intclr=0x2 for clear TX interrupt; Intclr=0x3 for clear RX and TX interrupt

Include

Driver/DrvPS2.h

Return Value

None.

Example

/* Clear RX interrupt. */

_DRVPS2_INTCLR(1);

```
/* Clear TX interrupt. */
_DRVPS2_INTCLR(2);
/* Clear TX and RX interrupt. */
_DRVPS2_INTCLR(3);
```

_DRVPS2_RXDATA

Prototype

uint8_t _DRVPS2_RXDATA();

Description

Reads 1 byte from the receive register.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

One byte data received.

Example

```
/* Read one byte from PS/2 receive data register. */
uint8_t u8ReceiveData;
u8ReceiveData = _DRVPS2_RXDATA();
```

_DRVPS2_TXDATAWAIT

Prototype

void _DRVPS2_TXDATAWAIT(uint32_t data, uint32_t len);

Description

The macro is used to wait TX FIFO EMPTY, set TX FIFO depth(length-1) and fill TX FIFO0-3(Register PS2TXDATA0). Data is sent immediately if bus is in IDLE state. The range of length is from 1 to 16 bytes. If the transfer size is more than 4 bytes, user should call DRVPS2_TXDATA1~3() after calling _DRVPS2_TXDATAWAIT() to transfer remind data.

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

Parameter

data [in]

Specify the data sent

len [in]

Specify the length of the data sent. Unit is byte. Range is [1, 16]

Include

Driver/DrvPS2.h

Return Value

None

Example

/* Wait TX FIFO empty and then write 16 bytes to TX FIFO. The sixteen bytes consist of 0x01 to 0x16. */

_DRVPS2_TXDATAWAIT(0x04030201, 16);

_DRVPS2_TXDATA1(0x08070605);

_DRVPS2_TXDATA2(0x0C0B0A09);

_DRVPS2_TXDATA3(0x100F0E0D);

/* Wait TX FIFO empty and then write 5 bytes to TX FIFO. The six bytes consist of 0x01 to 0x05. */

_DRVPS2_TXDATAWAIT(0x04030201, 5);

_DRVPS2_TXDATA1(0x05);

/* Wait TX FIFO empty and then write 3 bytes to TX FIFO. The three bytes consist of 0x01 to 0x03. */

_DRVPS2_TXDATAWAIT(0x030201, 3);

_DRVPS2_TXDATA

Prototype

void _DRVPS2_TXDATA(uint32_t data,   uint32_t len);

Description

The macro is used to set TX FIFO depth and fill TX FIFO0-3. But not wait TX FIFO EMPTY. Data is sent if bus is in IDLE state immediately. The range of len is [1, 16]

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

Parameter

data [in]

Specify the data sent

len [in]

Specify the length of the data sent. Unit is byte. Range is [1, 16]

Include

Driver/DrvPS2.h

Return Value

None

Note

If the transfer size is more than 4 bytes, user should issue _DRVPS2_TXDATA1~3() after issuing _DRVPS2_TXDATA();

Example

```
/*Write 16 bytes to TX FIFO. The sixteen bytes consist of 0x01 to 0x16. */
_DRVPS2_TXDATA(0x04030201, 16);
_DRVPS2_TXDATA1(0x08070605);
_DRVPS2_TXDATA2(0x0C0B0A09);
_DRVPS2_TXDATA3(0x100F0E0D);
/* Write 5 bytes to TX FIFO. The six bytes consist of 0x01 to 0x05. */
_DRVPS2_TXDATA(0x04030201, 5);
_DRVPS2_TXDATA1(0x05);
/* Write 3 bytes to TX FIFO. The three bytes consist of 0x01 to 0x03. */
_DRVPS2_TXDATA(0x030201, 3);
```

_DRVPS2_TXDATA0

Prototype

```
void _DRVPS2_TXDATA0(uint32_t data);
```

Description

The macro is used to fill TX FIFO0-3. But not wait TX FIFO EMPTY and not set TX FIFO depth. Data is sent if bus is in IDLE state immediately.

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

Parameter

data [in]

Specify the data that will be sent

Include

Driver/DrvPS2.h

Return Value

None.

Example

```
/* Write 16 bytes to TX FIFO. The sixteen bytes consist of 0x01 to 0x16. */
while(_DRVPS2_ISTXEMPTY()==0);
_DRVPS2_TXFIFO(16);
```

```
_DRVPS2_TXDATA0(0x04030201);
_DRVPS2_TXDATA1(0x08070605);
_DRVPS2_TXDATA2(0x0C0B0A09);
_DRVPS2_TXDATA3(0x100F0E0D);
```

_DRVPS2_TXDATA1

Prototype

void _DRVPS2_TXDATA1(uint32_t data);

Description

The macro is used to fill TX FIFO4-7. But not wait TX FIFO EMPTY and not set TX FIFO depth.

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

Parameter

data [in]

Specify the data that will be sent

Include

Driver/DrvPS2.h

Return Value

None

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_TXDATA2

Prototype

void _DRVPS2_TXDATA2(uint32_t data);

Description

The macro is used to fill TX FIFO8-11. But not wait TX FIFO EMPTY and not set TX FIFO depth.

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

Parameter

data [in]

Specify the data that will be sent

Include

Driver/DrvPS2.h

Return Value

None

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_TXDATA3

Prototype

void _DRVPS2_TXDATA3(uint32_t data);

Description

The macro is used to fill TX FIFO12-15. But not wait TX FIFO EMPTY and not set TX FIFO depth.

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

Parameter

data [in]

Specify the data that will be sent.

Include

Driver/DrvPS2.h

Return Value

None

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_ISTXEMPTY

Prototype

uint8_t _DRVPS2_ISTXEMPTY();

Description

The macro is used to check TX FIFO whether or not empty

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

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

TX FIFO empty status.

0: TX FIFO is empty.

1: TX FIFO is not empty.

Example

Please refer to _DRVPS2_TXDATA0() example.

_DRVPS2_ISFRAMEERR

Prototype

uint8_t _DRVPS2_ISFRAMEERR();

Description

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

Parameter

None

Include

Driver/DrvPS2.h

Return Value

Frame error status.

0: Not frame error.

1: Frame error.

Example

```
/* Check Frame error and print the result. */
if(_DRVPS2_ISFRAMEERR()==1)
    printf("Frame error happen!!\n");
else
    printf("Frame error not happen!!\n");
```

_DRVPS2_ISRXBUSY

Prototype

uint8_t _DRVPS2_ISRXBUSY();

Description

The macro is used to check whether or not Rx busy. If busy it indicates that PS/2 device is currently receiving data

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

RX busy flag.

0: RX is not busy,

1: RX is busy.

Example

```
/* Check RX is busy or not. */
if(_DRVPS2_ISRXBUSY()==1)
        printf("RX is busy!\n");
else
        printf("RX is not busy!\n");
```

12.5. Functions

DrvPS2_Open

Prototype

int32_t DrvPS2_Open();

Description

This function is used to init PS/2 IP. It includes enable PS2 clock, enable PS/2 controller, clear FIFO, set TX FIFO depth to default value zero.

Parameter

None

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS.

Example

```
/* Initialize PS/2 IP. */
DrvPS2_Open();
```

DrvPS2_Close

Prototype

void DrvPS2_Close();

Description

This function is used to disable PS2 controller, disable PS/2 clock and set TX FIFO depth to default value zero

Parameter

None

Include

Driver/ DrvPS2.h

Return Value

None

Example

```
/* Close PS2 IP. */
DrvPS2_Close ();
```

DrvPS2_EnableInt

Prototype

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

Description

This function is used to enable TX/RX interrupt and install interrupt call back function.

Parameter

u32InterruptFlag [in]

Specify TX/RX interrupt flag that will be enable. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT

pfncallback [in]

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

Include

Driver/ DrvPS2.h

Return Value

E_SUCCESS

Example

/* Enable TX/RX interrupt, install TX/RX call back function: PS2Mouse_IRQHandler(); */

DrvPS2_EnableInt(DRVPS2_TXINT| DRVPS2_RXINT, PS2Mouse_IRQHandler);

DrvPS2_DisableInt

Prototype

void DrvPS2_DisableInt(uint32_t u32InterruptFlag);

Description

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

Parameter

u32InterruptFlag [in]

Specify TX/RX interrupt flag that will be disabled. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT.

Include

Driver/ DrvPS2.h

Return Value

None

Example

/* Disable TX/RX interrupt and uninstall TX and RX call back function. */

DrvPS2_DisableInt(DRVPS2_TXINT| DRVPS2_RXINT);

DrvPS2_IsIntEnabled

Prototype

uint32_t DrvPS2_IsIntEnabled(uint32_t u32InterruptFlag);

Description

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

Parameter

u32InterruptFlag [in]

Specify TX/RX interrupt flag that will be checked. It can be DRVPS2_TXINT or DRVPS2_RXINT or DRVPS2_TXINT| DRVPS2_RXINT.

Include

Driver/DrvPS2.h

Return Value

0 : No interrupt be enable.

2 : TX interrupt be enable

4 : RX interrupt be enable

6 : TX and RX interrupt be enable.

Example

```
/* Check TX and RX interrupt enable or not enable. */
uint32_u32TXRXIntEnable
u32TXRXIntEnable = DrvPS2_IsIntEnabled(DRVPS2_TXINT| DRVPS2_RXINT)
if(u32TXRXIntEnable ==0)
printf("No interrupt be enable!!\n");
else if(u32TXRXIntEnable ==2)
printf("TX interrupt be enable!!\n");
else if(u32TXRXIntEnable ==4)
printf("RX interrupt be enable!!\n");
else if(u32TXRXIntEnable ==6)
printf("TX and RX interrupt be enable!!\n");
```

DrvPS2_ClearIn

Prototype

uint32_t DrvPS2_ClearInt(uint32_t u32InterruptFlag);

Description

This function is used to clear interrupt status.

Parameter

U32InterruptFlag [in]

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

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS: Success.

Example

```
/* Clear TX interrupt. */
DrvPS2_ClearInt(DRVPS2_TXINT);
/* Clear RX interrupt. */
DrvPS2_ClearInt(DRVPS2_RXINT);
/* Clear TX and RX interrupt. */
DrvPS2_ClearInt(DRVPS2_TXINT| DRVPS2_RXINT);
```

DrvPS2_GetIntStatus

Prototype

int8_t DrvPS2_GetIntStatus(uint32_t   u32InterruptFlag);

Description

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

Parameter

U32InterruptFlag [in]

Specify TX/RX interrupt flag that will be checked. It can be DRVPS2_TXINT or DRVPS2_RXINT

Include

Driver/DrvPS2.h

Return Value

TRUE: interrupt that be checked happens

FALSE: interrupt that be checked doesn't happen.

Example

```
/* Check TX interrupt status */
int8_t i8InterruptStatus;
i8InterruptStatus = DrvPS2_GetIntStatus(DRVPS2_TXINT);
if(i8InterruptStatus==TRUE)
    printf("TX interrupt that be checked happens"\n);
else
    printf("TX interrupt doesn't happen"\n);
```

DrvPS2_SetTxFIFODepth

Prototype

void DrvPS2_SetTxFIFODepth(uint16_t u16TxFIFODepth);

Description

This function is used to set TX FIFO depth. The function will call macro DRVPS2_TXFIFO to set TX FIFO depth

Parameter

u16TxFIFODepth [in]

Specify TX FIFO depth. The range can be [1, 16]

Include

Driver/DrvPS2.h

Return Value

None

Example

```
/* Set TX FIFO depth to 16 bytes. */
DrvPS2_SetTxFIFODepth(16);
/* Set TX FIFO depth to 1 byte. */
DrvPS2_SetTxFIFODepth(1);
```

DrvPS2_Read

Prototype

int32_t DrvPS2_Read(uint8_t *pu8RxBuf);

Description

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

Parameter

pu8RxBuf [out]

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

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS: Success.

Example

```
/* Read RX data and print it. */
uint8_t u8RXData;
DrvPS2_Read(&u8RXData);
printf("RX data is %x\n", u8RXData);
```

DrvPS2_Write

Prototype

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

Description

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

Parameter

pu32TxBuf [in]

the data that will be sent to host.

u32WriteBytes [in]

the length of data that will be sent to host.

Include

Driver/DrvPS2.h

Return Value

E_SUCCESS: Success.

Example

```
/* Write 64 bytes to TX buffer and TX buffer will send the 64 bytes out. */
uint32_t au32TXData[64];
DrvPS2_Write(au32TXData, 64);
```

DrvPS2_GetVersion

Prototype

```
int32_t DrvPS2_GetVersion(void);
```

Description

Return the current version number of driver.

Include

Driver/ DrvPS2.h

Return Value

PS2 driver current version number:

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

Example

```
/* Get PS/2 driver current version number */

int32_t i32Ps2VersionNum;

i32Ps2VersionNum = DrvPS2_GetVersion ();
```

13. FMC Driver

13.1. FMC Introduction

NuMicro™ NUC100 series equips with 128/64/32k bytes on chip embedded flash for application program memory (APROM), 4k bytes for ISP loader program memory (LDROM), and user configuration (Config0 & Config1). User configuration block provides several bytes to control system logic, like flash security lock, boot select, brown out voltage level, data flash base address, ..., and so on. NuMicro™ NUC100 series also provide additional 4k bytes data flash for user to store some application depended data before chip power off. For 128k bytes device, the data flash is shared with 128k program memory and its shared address is defined by user in Config1. The data flash size is defined by user depends on user application request.

13.2. FMC Feature

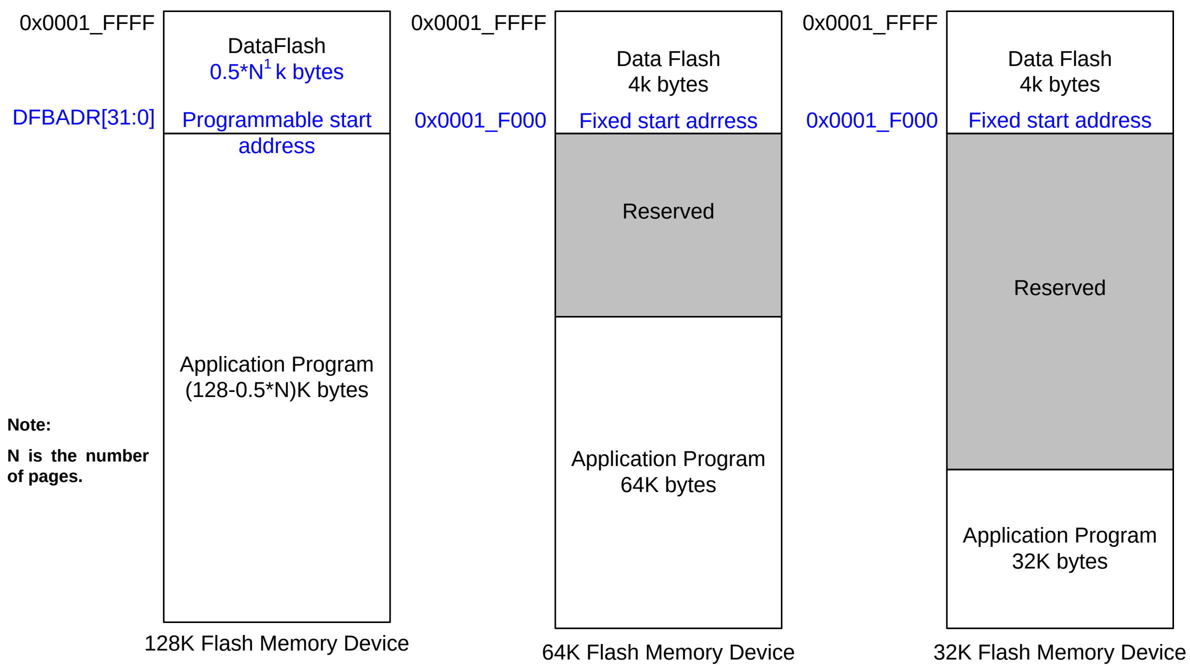
The FMC includes following features:

- 128/64/32kB application program memory (APROM).
- 4kB in system programming loader program memory (LDROM).
- 4kB data flash with 512 bytes page erase unit for user to store data
- Programmable data flash start address and memory size for 128KB program memory.
- Provide user configuration to control system logic.
- APROM cannot be updated when the MCU is running in APROM; LDROM can not be updated when the MCU is running in LDROM

Memory Address Map

Block Name	Size	Start Address	End Address
AP ROM	32 KB 64 KB 128 KB (128-0.5*N) KB	0x00000000	0x00007FFF 0x0000FFFF 0x0001FFFF if DFEN=0 for 128 KB (DFBADR-1) if DFEN=1 for 128 KB
Data Flash	4 KB 4 KB 0 KB (0.5*N) KB	0x0001F000 0x0001F000 None DFBADR	0x0001FFFF 0x0001FFFF None if DFEN=0 for 128 KB 0x0001FFFF if DFEN=1 for 128 KB
LD ROM	4KB	0x00100000	0x00100FFF
User Configuration	2 words	0x00300000	0x00300004

Flash Memory Structure



13.3. Type Definition

E_FMC_BOOTSELECT

Enumeration identifier	Value	Description
E_FMC_APROM	0	Boot from APROM
E_FMC_LDROM	1	Boot from LDROMl

13.4. Functions

DrvFMC_EnableISP

Prototype

void DrvFMC_EnableISP (void);

Description

To enable ISP function. This function will check if internal 22M oscillator is enabled or not. If not, this function will enable 22M oscillator automatically. User can disable 22M oscillator by using DrvSYS_SetOscCtrl ( ) if needed after ISP finished.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_EnableISP ( );      /* Enable ISP function */
```

DrvFMC_DisableISP

Prototype

```
void    DrvFMC_DisableISP (void);
```

Description

To disable ISP function.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_DisableISP ( );      /* Disable ISP function */
```

DrvFMC_BootSelect

Prototype

void DrvFMC_BootSelect(E_FMC_BOOTSELECT boot);

Description

To select next booting from APROM or LDROM.

Note

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

Parameter

boot [in]

Specify E_FMC_APROM or E_FMC_LDROM.

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_BootSelect (E_FMC_LDROM); /* Next booting from LDROM */

DrvFMC_BootSelect (E_FMC_APROM); /* Next booting from APROM */

DrvFMC_GetBootSelect

Prototype

E_FMC_BOOTSELECT DrvFMC_GetBootSelect(void);

Description

To get current boot select setting.

Parameter

None.

Include

Driver/DrvFMC.h

Return Value

E_FMC_APROM	The current boot select setting is in APROM
E_FMC_LDROM	The current boot select setting is in LDROM

Example

E_FMC_BOOTSELECT e_bootSelect

/* Check this booting is from APROM or LDROM */

e_bootSelect = DrvFMC_GetBootSelect ( );

DrvFMC_EnableLDUpdate

Prototype

void DrvFMC_EnableLDUpdate (void);

Description

To enable LDROM update function. LDROM can be updated if LDROM update function is enabled when the MCU runs in APROM.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_EnableLDUpdate ( ); /* Enable LDROM update function */

DrvFMC_DisableLDUpdate

Prototype

void DrvFMC_DisableLDUpdate (void);

Description

To disable LDROM update function.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_DisableLDUpdate ( ); /* Disable LDROM update function */

DrvFMC_EnableConfigUpdate

Prototype

void DrvFMC_EnableConfigUpdate (void);

Description

To enable Config update function. If Congif update function is enabled, the user configuration can be update regardless of MCU is running in APROM or LDROM.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_EnableConfigUpdate ( ); /* Enable Config update function */

DrvFMC_DisableConfigUpdate

Prototype

void DrvFMC_DisableConfigUpdate (void);

Description

To disable Config update function.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_DisableConfigUpdate ( );            /* Disable Config update function */
```

DrvFMC_EnableAPUpdate

Prototype

```
void    DrvFMC_EnableAPUpdate (void);
```

Description

To enable APROM update function. APROM can be updated if APROM update function is enabled when the MCU runs in APROM.

Note 1

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Note 2

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_EnableAPUpdate ( );          /* Enable APROM update function */
```

DrvFMC_DisableAPUpdate

Prototype

void DrvFMC_DisableAPUpdate (void);

Description

To disable APROM update function.

Note 1

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Note 2

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_DisableAPUpdate ( );          /* Disable APROM update function */
```

DrvFMC_EnablePowerSaving

Prototype

void DrvFMC_EnablePowerSaving (void);

Description

To enable flash access power saving function. If CPU clock is slower than 24 MHz, user can enable flash power saving function.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_EnablePowerSaving ( ); /* Enable flash power saving function */

DrvFMC_DisablePowerSaving

Prototype

void DrvFMC_DisablePowerSaving (void);

Description

To disable flash access power saving function.

Note

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_DisablePowerSaving ( ); /* Disable flash power saving function */

DrvFMC_Write

Prototype

int32_t DrvFMC_Write (uint32_t u32addr, uint32_t u32data);

Description

To write word data into APROM, LDROM, Data Flash or Config. The Memory Map of APROM and Data Flash are depended on the product of NuMicro™ NUC100 series. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix for Flash size. The corresponding function in Config0 and Config1 are described in FMC Section of TRM in details.

Note

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

Parameter

u32addr [in]

Word address of APROM, LDROM, Data Flash or Config.

u32data [in]

Word data to be programmed into APROM, LDROM, Data Flash or Config.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

/* Program word data 0x12345678 into address 0x1F000 */

DrvFMC_Write (0x1F000, 0x12345678);

DrvFMC_Read

Prototype

int32_t DrvFMC_Read (uint32_t u32addr, uint32_t * u32data);

Description

To read data from APROM, LDROM, Data Flash or Config. The Memory Map of APROM and Data Flash are depended on the product of NuMicro™ NUC100 series. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix for Flash size.

Note

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

Parameter

u32addr [in]

Word address of APROM, LDROM, Data Flash or Config.

u32data [in]

The word data to store data from APROM, LDROM, Data Flash or Config.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

uint32_t u32Data;

/* Read word data from address 0x1F000, and read data is stored to u32Data */

DrvFMC_Read (0x1F000, &u32Data);

DrvFMC_Erase

Prototype

int32_t   DrvFMC_Erase (uint32_t u32addr);

Description

To page erase APROM, LDROM, Data Flash or Config. The flash page erase unit is 512 bytes. The Memory Map of APROM and Data Flash are depended on the product of NuMicro™ NUC100 series. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix for Flash size.

Note

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

Parameter

u32addr [in]

Flash page base address of APROM, LDROM and Data Flash, or Config0 addrsss.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

/* Page Erase from 0x1F000 to 0x1F1FF */

DrvFMC_Erase (0x1F000);

DrvFMC_WriteConfig

Prototype

int32_t DrvFMC_WriteConfig(uint32_t u32data0, uint32_t u32data1);

Description

To erase Config and write data into Config0 and Config1. The corresponding functions in Config0 and Config1 are described in FMC Section of TRM in details.

Note

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

Parameter

u32data0 [in]

Word data to be programmed into Config0.

u32data1 [in]

Word data to be programmed into Config1.

Include

Driver/DrvFMC.h

Return Value

0: Succeed

<0: Failed

Example

/* Program word data 0xFFFFFFFE into Config0 and word data 0x1E000 into Config1 */

DrvFMC_Config (0xFFFFFFFE, 0x1E000);

DrvFMC_ReadDataFlashBaseAddr

Prototype

uint32_t DrvFMC_ReadDataFlashBaseAddr (void);

Description

To read data flash base address. For 128k bytes flash device, the base address of data flash is defined by user in Config1. For less 128k bytes flash device, the base address is fixed at 0x1F000.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

Data Flash base address

Example

```
uint32_t u32Data;
/* Read Data Flash base address */
u32Data = DrvFMC_ReadDataFlashBaseAddr ( );
```

DrvFMC_EnableLowFreqOptMode

Prototype

```
void    DrvFMC_EnableLowFreqOptMode (void);
```

Description

To enable flash access low frequency optimization mode. It can improve flash access performance when CPU runs at low frequency.

Note 1

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) and Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details. And, Set this bit only when HCLK ≤ 25MHz. If HCLK > 25MHz, CPU will fetch wrong code and cause fail result.

Note 2

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Enable flash access low frequency optimization mode */
DrvFMC_EnableLowFreqOptMode ( );
```

DrvFMC_DisableLowFreqOptMode

Prototype

```
void    DrvFMC_DisableLowFreqOptMode (void);
```

Description

To disable flash access low frequency optimization mode.

Note 1

Only NuMicro™ NUC1x0xxxCx series (Ex. NUC140VE3CN) and Low Density series support this function. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

Note 2

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

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
/* Disable flash access low frequency optimization mode */
DrvFMC_DisableLowFreqOptMode ( );
```

DrvFMC_GetVersion

Prototype

uint32_t DrvFMC_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

Version number:

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

14. USB Driver

14.1. Introduction

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

14.2. Feature

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

14.3. USB Framework

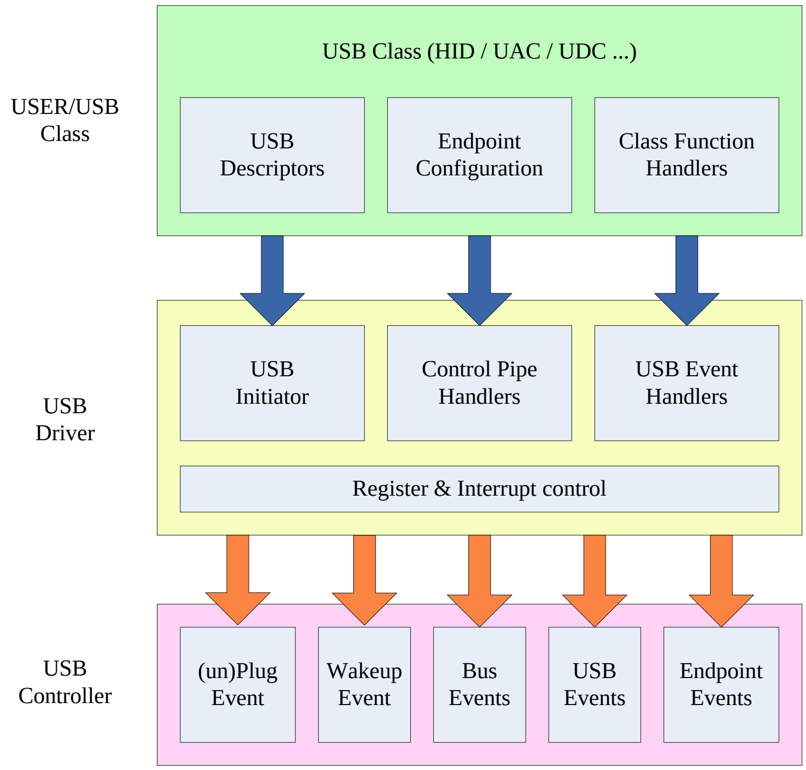


Figure 14-1: USB Framework

Above figure shows the framework of USB device library. The lowest layer is USB controller. The USB controller will raise different interrupt events according to USB, BUS and floating detection status. All the events are handled by USB driver by relative event handlers. USB driver also take care the basic handler of control pipe of USB protocol. Most function dependent handlers and USB descriptors must be provided by user applications or USB class definitions.

14.4. Call Flow

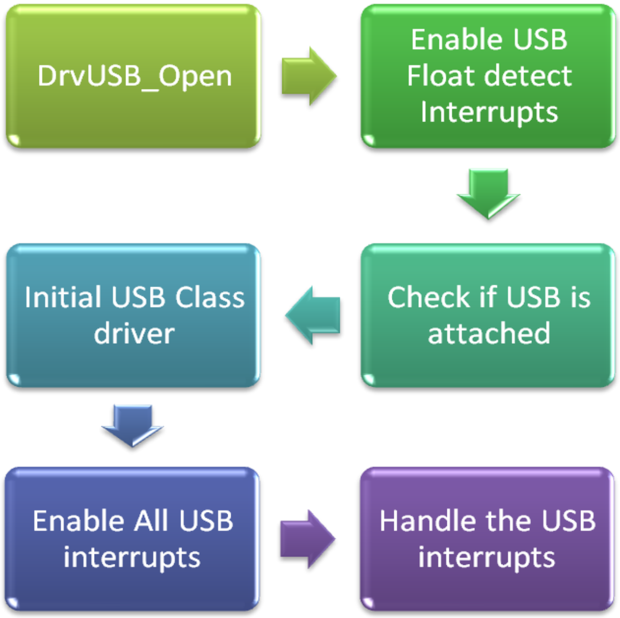


Figure 14-2: USB Driver Call Flow

The above figure shows the call flow of USB driver. The DrvUSB_Open is used to initial the USB device controller. Then USB floating detection is enabled to detect USB plug/un-plug events. If USB attached, it need to call the USB class driver to initial USB class specified descriptions, event handlers. Finally, all related USB interrupts are enabled to handle the USB events.

14.5. Constant Definition

USB Register Address

Constant Name	Value	Description
USBD_INTEN	0x40060000	USB Interrupt Enable Register Address
USBD_INTSTS	0x40060004	USB Interrupt Event Status Register Address
USBD_FADDR	0x40060008	USB Device Function Address Register Address
USBD_EPSTS	0x4006000C	USB Endpoint Status Register Address Address
USBD_ATTR	0x40060010	USB Bus Status and Attribution Register Address
USBD_FLDETB	0x40060014	USB Floating Detected Register Address
USBD_BUFSEG	0x40060018	Setup Token Buffer Segmentation Register Address
USBD_BUFSEG0	0x40060020	Endpoint 0 Buffer Segmentation Register Address
USBD_MXPLD0	0x40060024	Endpoint 0 Maximal Payload Register Address

Constant Name	Value	Description
USBD_CFG0	0x40060028	Endpoint 0 Configuration Register Address
USBD_CFGP0	0x4006002C	Endpoint 0 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG1	0x40060030	Endpoint 1 Buffer Segmentation Register Address
USBD_MXPLD1	0x40060034	Endpoint 1 Maximal Payload Register Address
USBD_CFG1	0x40060038	Endpoint 1 Configuration Register Address
USBD_CFGP1	0x4006003C	Endpoint 1 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG2	0x40060040	Endpoint 2 Buffer Segmentation Register Address
USBD_MXPLD2	0x40060044	Endpoint 2 Maximal Payload Register Address
USBD_CFG2	0x40060048	Endpoint 2 Configuration Register Address
USBD_CFGP2	0x4006004C	Endpoint 2 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG3	0x40060050	Endpoint 3 Buffer Segmentation Register Address
USBD_MXPLD3	0x40060054	Endpoint 3 Maximal Payload Register Address
USBD_CFG3	0x40060058	Endpoint 3 Configuration Register Address
USBD_CFGP3	0x4006005C	Endpoint 3 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG4	0x40060060	Endpoint 4 Buffer Segmentation Register Address
USBD_MXPLD4	0x40060064	Endpoint 4 Maximal Payload Register Address
USBD_CFG4	0x40060068	Endpoint 4 Configuration Register Address
USBD_CFGP4	0x4006006C	Endpoint 4 Set Stall and Clear In/Out Ready Control Register Address
USBD_BUFSEG5	0x40060070	Endpoint 5 Buffer Segmentation Register Address
USBD_MXPLD5	0x40060074	Endpoint 5 Maximal Payload Register Address
USBD_CFG5	0x40060078	Endpoint 5 Configuration Register Address
USBD_CFGP5	0x4006007C	Endpoint 5 Set Stall and Clear In/Out Ready Control Register Address
USBD_DRVSE0	0x40060090	USB Drive SE0 Control Register Address
USB_SRAM_BASE	0x40060100	USB PDMA Control Register Address

INTEN Register Bit Definition

Constant Name	Value	Description
INTEN_INNAK	0x00008000	Active NAK interrupt function and its status flag for IN token
INTEN_WAKEUP_EN	0x00000100	Wake Up Function Enable
INTEN_WAKEUP_IE	0x00000008	USB Wake Up Interrupt Enable
INTEN_FLDET_IE	0x00000004	Floating Detect Interrupt Enable
INTEN_USB_IE	0x00000002	USB Event Interrupt Enable

Constant Name	Value	Description
INTEN_BUS_IE	0x00000001	Bus Event Interrupt Enable

INTSTS Register Bit Definition

Constant Name	Value	Description
INTSTS_SETUP	0x80000000	Setup Event Status
INTSTS_EPEVT5	0x00200000	Endpoint 5's USB Event Status
INTSTS_EPEVT4	0x00100000	Endpoint 4's USB Event Status
INTSTS_ EPEVT 3	0x00080000	Endpoint 3's USB Event Status
INTSTS_ EPEVT 2	0x00040000	Endpoint 2's USB Event Status
INTSTS_ EPEVT 1	0x00020000	Endpoint 1's USB Event Status
INTSTS_ EPEVT 0	0x00010000	Endpoint 0's USB Event Status
INTSTS_WAKEUP_STS	0x00000008	Wakeup Interrupt Status
INTSTS_FLDET_STS	0x00000004	Floating Detected Interrupt Status
INTSTS_USB_STS	0x00000002	USB event Interrupt Status
INTSTS_BUS_STS	0x00000001	BUS Interrupt Status

ATTR Register Bit Definition

Constant Name	Value	Description
ATTR_BYTEM	0x00000400	CPU access USB RAM Size Mode Select
ATTR_PWRDN	0x00000200	Power down PHY, low active
ATTR_DPPU_EN	0x00000100	Pull-up resistor on D+ enable
ATTR_USB_EN	0x00000080	USB Controller Enable
ATTR_RWAKEUP	0x00000020	Remote Wake Up
ATTR_PHY_EN	0x00000010	PHY Function Enable
ATTR_TIMEOUT	0x00000008	Time Out Status
ATTR_RESUME	0x00000004	Resume Status
ATTR_SUSPEND	0x00000002	Suspend Status
ATTR_USBRST	0x00000001	USB Reset Status

Confiuration Register Bit Definition

Constant Name	Value	Description
CFG_CSTALL	0x00000200	Clear STALL Response

Constant Name	Value	Description
CFG_DSQ_SYNC	0x00000080	Data Sequence Synchronization
CFG_STATE	0x00000060	Endpoint STATE
CFG_EPT_IN	0x00000040	IN endpoint
CFG_EPT_OUT	0x00000020	Out endpoint
CFG_ISOCH	0x00000010	Isochronous Endpoint
CFG_EP_NUM	0x0000000F	Endpoint Number

Extera-Confiuration Register Bit Definition

Constant Name	Value	Description
CFGP_SSTALL	0x00000002	Set the device to respond STALL
CFGP_CLRRDY	0x00000001	Clear Ready

14.6. Macro

_DRVUSB_ENABLE_MISC_INT

Prototype

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

Description

Enable/Disable miscellaneous interrupts including USB event, Wakeup event, Float-detection event and bus event.

Parameter

u32Flags [in]

USB interrupt events. It can be following flags.

IEF_WAKEUP: Wakeup interrupt flag.

IEF_FLD: Float-detection interrupts flag.

IEF_USB: USB event interrupt flag.

IEF_BUS: Bus event interrupt flag.

u32Flag = 0 will disable all USB interrupts.

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_MISC_INT(0);   /* Disable All USB-related interrupts. */

_DRVUSB_ENABLE_MISC_INT(IEF_WAKEUP | IEF_WAKEUPEN | IEF_FLD | IEF_USB | IEF_BUS); /* Enable wakeup, float-detection, USB and bus interrupts */

_DRVUSB_ENABLE_WAKEUP

Prototype

void _DRVUSB_ENABLE_WAKEUP (void);

Description

Enable USB wakeup function. If USB wakeup function is enabled, any activity of USB bus could be used to wakeup CPU from power down.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_WAKEUP(); /* To enable the USB wakeup function */

_DRVUSB_DISABLE_WAKEUP

Prototype

void _DRVUSB_DISABLE_WAKEUP (void);

Description

Disable USB wakeup function. If USB wakeup function is disable, USB can’t used to wakeup up CPU from power down.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_DISABLE_WAKEUP(); /* To avoid wakeup CPU by USB */

_DRVUSB_ENABLE_WAKEUP_INT

Prototype

void _DRVUSB_ENABLE_WAKEUP_INT (void);

Description

Enable wakeup interrupt. USB will raise a wakeup event interrupt when wakeup interrupt is enabled.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

DRVUSB_ENABLE_WAKEUP_INT() /* To enable wakeup event interrupt */

_DRVUSB_DISABLE_WAKEUP_INT

Prototype

void _DRVUSB_DISABLE_WAKEUP_INT (void);

Description

Disable wakeup interrupt to avoid USB raise an interrupt when wakeup from power down.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

DRVUSB_DISABLE_WAKEUP_INT () /* To disable wakeup event interrupt */

_DRVUSB_ENABLE_FLDET_INT

Prototype

void _DRVUSB_ENABLE_FLDET_INT (void);

Description

Enable float-detection interrupt to raise an interrupt when USB plug-in or un-plug.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_FLDET_INT() /* To enable float-detection interrupt */

_DRVUSB_DISABLE_FLDET_INT

Prototype

void _DRVUSB_DISABLE_FLDET_INT (void);

Description

Disable float-detection interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_DISABLE_FLDET_INT() /* To disable float-detection interrupt */

_DRVUSB_ENABLE_USB_INT

Prototype

void _DRVUSB_ENABLE_USB_INT (void);

Description

Enable USB interrupt. It could be used to control USB interrupt only and _DRVUSB_ENABLE_MISC_INT() can be used to control all USB related interrupts at the same time.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_USB_INT () /* To enable USB interrupt */

_DRVUSB_DISABLE_USB_INT

Prototype

void _DRVUSB_DISABLE_USB_INT (void);

Description

Disable USB interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

DRVUSB DISABLE _USB_INT () /* To disable USB interrupt */

_DRVUSB_ENABLE_BUS_INT

Prototype

void _DRVUSB_ENABLE_BUS_INT (void);

Description

Enable USB bus interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_ENABLE_BUS_INT () /* To enable USB bus interrupt */

_DRVUSB_DISABLE_BUS_INT

Prototype

void _DRVUSB_DISABLE_BUS_INT (void);

Description

Disable bus interrupt.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_DISABLE_BUS_INT () /* To disable USB bus interrupt */

_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL

Prototype

```
void _DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL (
    uint32_t      u32EPId
);
```

Description

Clear specified USB endpoint hardware In/Out Ready and respond STALL,

Parameter

u32EPId[in]

EP Identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_CLEAR_EP_READY_AND_TRIG_STALL(3)   /* To clear ready flag of USB endpoint identity 3 and let it to response STALL. */

Notes

Here, EP (endpoint) identity means number of USB device hardware, not USB endpoint number defined by USB standard.

_DRVUSB_CLEAR_EP_READY

Prototype

```
void _DRVUSB_CLEAR_EP_READY (
    uint32_t      u32EPId
);
```

Description

Clear EP In/Out Ready.

Parameter

u32EPId[in]

EP Identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_CLEAR_EP_READY(1)   /* To clear ready flag of USB endpoint identity 1. */

_DRVUSB_SET_SETUP_BUF

Prototype

void _DRVUSB_SET_SETUP_BUF (

uint32_t   u32BufAddr

);

Description

Specify buffer address for Setup transaction. This buffer is used to store setup token data and its size is fixed to be 8 bytes according to USB standard. Therefore, the buffer address must be 8 bytes alignment.

Parameter

u32BufAddr [in]

Buffer address for setup token. It could be USB_BA+0x100 ~ USB_BA+0x2F8 where USB_BA is 0x40060000.

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_SET_SETUP_BUF(0x400602F8)   /* Set the setup packet address to 0x400602F8 */

_DRVUSB_SET_EP_BUF

Prototype

void _DRVUSB_SET_EP_BUF (

uint32_t   u32EPId,

uint32_t   u32BufAddr

);

Description

Specify buffer address for specified hardware endpoint identity and it must be 8 bytes alignment. This buffer would be used to buffer the data of IN/OUT USB transaction. The buffer size used by IN/OUT USB transaction is dependent on maximum payload of related endpoint identity.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

u32BufAddr [in]

Used to set buffer address and valid address is from 0x40060100 ~ 0x400602F8. Furthermore, buffer address + maximum payload size must less than 0x400602FF.

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_SET_EP_BUF(1, 0x40060100)   /* Set the buffer address of endpoint identity 1 to 0x40060100 */

_DRVUSB_TRIG_EP

Prototype

```
void _DRVUSB_TRIG_EP (
    uint32_t    u32EPId,
    uint32_t    u32TrigSize
);
```

Description

Trigger next transaction for specified endpoint identity and the transaction size is also defined at the same time.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5) for trigger Data In or Out transaction.

u32TrigSize [in]

For Data Out transaction, it means maximum data size transferred from Host; for Data In transaction, it means how many data transferred to Host.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Trigger the transaction of endpoint identity 1 and the transaction payload size is 64 bytes */
_DRVUSB_TRIG_EP (1, 64)

_DRVUSB_GET_EP_DATA_SIZE

Prototype

uint32_t
_DRVUSB_GET_EP_DATA_SIZE (
    uint32_t   u32EPId
);

Description

Length of data transmitted to or received from Host for specified endpoint identity.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

For IN endpoint: length of data transmitting to host in bytes.

For OUT endpoint: Actual length of data receiving from host in bytes.

Example

/* To get the size of received data of endpoint identity 1. */
size = _DRVUSB_GET_EP_DATA_SIZE(1);

_DRVUSB_SET_EP_TOG_BIT

Prototype

void   _DRVUSB_SET_EP_TOG_BIT (
    uint32_t   u32EPId,
    int32_t   bData0
)

Description

Specify Data0 or Data1 for specified endpoint identity. This bit will toggle automatically after Host ACK the IN token.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

bData0 [in]

Specify DATA0 or DATA1 for IN transaction. TRUE is for DATA0, FALSE is for DATA1.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To set the toggle bit as DATA0 for endpoint identity 1 */
_DRVUSB_SET_EP_TOG_BIT(1, TRUE);
```

_DRVUSB_SET_EVENT_FLAG

Prototype

```
void _DRVUSB_SET_EVENT_FLAG (
    uint32_t    u32Data
);
```

Description

Set Interrupt Event Flag to clear them. The interrupt event flags are write one clear.

Parameter

u32Data [in]

Specify the event to be clear. It could be

Events	Value	Description
EVF_SETUP	0x80000000	Got a setup token event
EVF_EPTF5	0x00200000	Got USB event from endpoint identity 5
EVF_EPTF4	0x00100000	Got USB event from endpoint identity 4
EVF_EPTF3	0x00080000	Got USB event from endpoint identity 3
EVF_EPTF2	0x00040000	Got USB event from endpoint identity 2
EVF_EPTF1	0x00020000	Got USB event from endpoint identity 1
EVF_EPTF0	0x00010000	Got USB event from endpoint identity 0
EVF_WAKEUP	0x00000008	Got a wakeup event
EVF_FLD	0x00000004	Got float-detection event
EVF_USB	0x00000002	Got USB event include endpoint events or setup event
EVF_BUS	0x00000001	Got USB bus event

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_SET_EVENT_FLAG(EVF_BUS); /* Clear USB bus event */

_DRVUSB_SET_EVENT_FLAG(EVF_BUS | EVF_FLD); /* Clear USB bus event and float-detection event */

_DRVUSB_GET_EVENT_FLAG

Prototype

uint32_t

_DRVUSB_GET_EVENT_FLAG (void);

Description

Get Interrupt Event Flags

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return EVF register value. Please refer to _DRVUSB_SET_EVENT_FLAG() for detail event information.

Example

u32Events = _DRVUSB_GET_EVF(); /* Get events */

_DRVUSB_CLEAR_EP_STALL

Prototype

```
void _DRVUSB_CLEAR_EP_STALL (
    uint32_t      u32EPId
);
```

Description

Stop to force specified endpoint identity to respond STALL to host.

Parameter

u32EPId [in]

EP identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_CLEAR_EP_STALL(1);/* Clear the STALL of endpoint identity 1 */

_DRVUSB_TRIG_EP_STALL

Prototype

```
void _DRVUSB_TRIG_EP_STALL (
    uint32_t     u32EPId
);
```

Description

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

Parameter

u32EPId[in]

EP identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

_DRVUSB_TRIG_EP_STALL (1); /* Force to STALL endpoint identity 1 */

_DRVUSB_CLEAR_EP_DSQ_SYNC

Prototype

```
void _DRVUSB_CLEAR_EP_DSQ_SYNC (
uint32_t     u32EPId
```

);

Description

Clear the endpoint toggle bit to DATA0,i.e force the toggle bit to be DATA0. This bit will toggle automatically after IN token ack from host.

Parameter

u32EPId [in]

EP Identity (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Force the toggle bit of endpoint identity 2 to be DATA0 */
_DRVUSB_CLEAR_EP_DSQ_SYNC (2);
```

_DRVUSB_SET_CFG

Prototype

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

Description

This macro is used to set USB CFG register.

Parameter

u32CFGNum [in]

CFG number (valid value: 0 ~ 5).

u32Data [in]

Specify the setting for CFG register.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Set USB CFG2 control register as 0x3 */
_DRVUSB_SET_CFG (2, 0x3);
```

_DRVUSB_GET_CFG

Prototype

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

Description

Get current setting of USB CFG register.

Parameter

u32CFGNum [in]

CFG number (valid value: 0 ~ 5).

Include

Driver/DrvUsb.h

Return Value

Return specified CFG register value

Example

```
/* Get the setting of USB CFG2 control register */
u32Cfg = _DRVUSB_GET_CFG (3);
```

_DRVUSB_SET_FADDR

Prototype

```
void _DRVUSB_SET_FADDR (
    uint32_t    u32Addr
)
```

Description

To set USB device address. The valid address is from 0 ~ 127.

Parameter

u32Addr [in]

The USB device address and it could be 0 ~ 127.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Set the USB devcie address as 3 */

_DRVUSB_SET_FADDR (3);

_DRVUSB_GET_FADDR

Prototype

uint32_t

_DRVUSB_GET_FADDR (void)

Description

To get USB device address.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return USB device address.

Example

/* Get USB devcie address */

u32Addr = _DRVUSB_GET_FADDR ();

_DRVUSB_GET_EPSTS

Prototype

uint32_t

_DRVUSB_GET_EPSTS (void)

Description

Get USB endpoint states register (EPSTS) value. The states register could be used to idendity the detail information of USB event. For detail information of EPSTS, please refere to NuMicro™ Technical Reference Manual.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Return STS register value

Example

```
/* Get USB STS register value */
u32Reg = _DRVUSB_GET_STS();
```

_DRVUSB_SET_CFGP

Prototype

```
void _DRVUSB_SET_CFGP(
    uint8_t     u8CFGPNum,
    uint32_t    u32Data
);
```

Description

To set extra configuration register (CFGP). The CFGP register could be used to STALL the endpoint and clear endpoint ready flag.

CFGP[1]: STALL control bit. Set ‘1’ to force the endpoint to response STALL to host.

CFGP[0]: Ready flag and it is write one clear.

Parameter

u8CFGPNum[in]

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

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL the endpoint identity 1. */
_DRVUSB_SET_CFGP(1, 0x2);
```

_DRVUSB_GET_CFGP

Prototype

```
uint32_t
_DRVUSB_GET_CFGP (
    uint32_t    u32CFGPNum
);
```

Description

Get the value of extra configuration register (CFGP)

Parameter

u32CFGPNum[in]

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

Include

Driver/DrvUsb.h

Return Value

Return CFGP register value

Example

```
/* Get the register value of CFG1 */
_DRVUSB_GET_CFGP(1);
```

_DRVUSB_ENABLE_USB

Prototype

```
void _DRVUSB_ENABLE_USB (void)
```

Description

Enable USB, PHY and use remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Enable USB, PHY and remote wakeup. */

_DRVUSB_ENABLE_USB();

_DRVUSB_DISABLE_USB

Prototype

void _DRVUSB_DISABLE_USB (void)

Description

Disable USB, PHY but still enable remote wake-up

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Disable USB, PHY but still enable remote wakeup. */

_DRVUSB_DISABLE_USB();

_DRVUSB_DISABLE_PHY

Prototype

void _DRVUSB_DISABLE_PHY (void)

Description

Disable PHY and remote wake-up.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Disable PHY and remote wakeup. */

_DRVUSB_DISABLE_PHY();

_DRVUSB_ENABLE_SE0

Prototype

void _DRVUSB_ENABLE_SE0 (void)

Description

Force USB to drive SE0 to bus. It can be used to simulate unplug event to let host re-connect to device. For more information about SE0, please refer to USB standard.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

/* Force bus to be SE0 state */

_DRVUSB_ENABLE_SE0();

_DRVUSB_DISABLE_SE0

Prototype

void _DRVUSB_DISABLE_SE0 (void)

Description

Stop to drive SE0 to USB bus.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* Stop to drive SE0 state to USB bus */
_DRVUSB_DISABLE_SE0();
```

_DRVUSB_SET_CFG_EP0

Prototype

```
void _DRVUSB_SET_CFG_EP0 (
    uint32_t     u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 0. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 0 */
_DRVUSB_SET_CFG_EP0(0x2);
```

_DRVUSB_SET_CFG_EP1

Prototype

```
void _DRVUSB_SET_CFG_EP1 (
    uint32_t    u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 1. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To STALL endpoint identity 1 */
_DRVUSB_SET_CFG_EP1(0x2);
```

_DRVUSB_SET_CFG_EP2

Prototype

```
void _DRVUSB_SET_CFG_EP2 (
    uint32_t    u32Data
)
```

Description

Stall control and clear In/out ready flag of endpoint identity 2. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* To STALL endpoint identity 2 */

_DRVUSB_SET_CFG_EP2(0x2);

_DRVUSB_SET_CFGP3

Prototype

void _DRVUSB_SET_CFG_EP3 (

uint32_t u32Data

)

Description

Stall control and clear In/out ready flag of endpoint identity 3. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* To STALL endpoint identity 3 */

_DRVUSB_SET_CFG_EP3(0x2);

_DRVUSB_SET_CFGP4

Prototype

void _DRVUSB_SET_CFG_EP4 (

uint32_t u32Data

)

Description

Stall control and clear In/out ready flag of endpoint identity 4. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* To STALL endpoint identity 4 */

_DRVUSB_SET_CFG_EP4(0x2);

_DRVUSB_SET_CFGP5

Prototype

void _DRVUSB_SET_CFG_EP5 (

uint32_t u32Data

)

Description

Stall control and clear In/out ready flag of endpoint identity 5. Please refer to _DRVUSB_SET_CFGP() for the bit definition of CFGP register.

Parameter

u32Data [in]

Specify data in CFGP register to STALL the endpoint or clear ready flag.

Include

Driver/DrvUsb.h

Return Value

None

Example

/* To STALL endpoint identity 5 */

_DRVUSB_SET_CFG_EP5(0x2);

14.7. Functions

DrvUSB_GetVersion

Prototype

uint32_t

DrvUSB_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

Version number :

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

Example

```
/* To get module's version */
u32Version = DrvUSB_GetVersion();
```

DrvUSB_Open

Prototype

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

Description

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

sEpDescription:

The structure type of sEpDescription is as follows:

```
typedef struct
{
    //bit7 is directory bit, 1: input; 0: output
    uint32_t    u32EPAddr;
    uint32_t    u32MaxPacketSize;
    uint8_t * u8SramBuffer;
}S_DRVUSB_EP_CTRL;
```

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

g_sBusOps:

The structure type of g_sBusOps is as follows:

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

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

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
    {NULL, NULL},                                   /* attach event callback */
    {NULL, NULL},                                   /* detach event callback */
    {DrvUSB_BusResetCallback, &g_HID_sDevice},      /* bus reset event callback */
    {NULL, NULL},                                   /* bus suspend event callback */
    {NULL, NULL},                                   /* bus resume event callback */
    {DrvUSB_CtrlSetupAck, &g_HID_sDevice},          /* setup event callback */
};
```

Parameter

pVoid

NULL	None
Callback function	If the pVoid is not NULL, it will be the callback function of USB

interrupt and it is called after DrvUSB_PreDispatchEvent in USB interrupt handler

Include

Driver/DrvUsb.h

Return Value

E_SUCCESS: Succeed

Example

```
/* To open USB device */
i32Ret = DrvUSB_Open(0);
if(i32Ret != E_SUCCESS)
    return i32Ret;
```

DrvUSB_Close

Prototype

void DrvUSB_Close (void);

Description

Close USB controller and disable USB interrupt.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* To close USB device */
DrvUSB_Close();
```

DrvUSB_PreDispatchEvent

Prototype

void DrvUSB_PreDispatchEvent(void);

Description

Pre-dispatch event base on EVF register.

Parameter

None

Include

Driver/DrvUsb.h

Return Value

None

Example

```
/* To pre dispatch USB device events at IRQ handler */
USBD_IRQHandler()
{
        DrvUSB_PreDispatchEvent();
}
```

DrvUSB_DispatchEvent

Prototype

void DrvUSB_DispatchEvent(void)

Description

Dispatch misc and endpoint event. Misc event include attach/detach/bus reset/bus suspend/bus resume and setup ACK, Misc event's handler is defined by g_sBusOps[]. The user must provide g_sBusOps[] before using USB driver.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* To dispatch USB events to handle them by related callback funcitons. */
DrvUSB_DispatchEvent();
```

DrvUSB_IsData0

Prototype

int32_t DrvUSB_IsData0(uint32_t u32EpId)

Description

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

Parameter

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

Include

Driver/DrvUSB.h

Return Value

TRUE The current data packet is DATA0
FALSE The current data packet is DATA1

Example

```
/* Get toggle bit of endpoint identity 2 */
if(DrvUSB_IsData0(2) )
{
    /* The toggle bit of endpoint identity 2 is DATA0 */
}
```

DrvUSB_GetUsbState

Prototype

E_DRVUSB_STATE DrvUSB_GetUsbState(void)

Description

Get current USB state E_DRVUSB_STATE. The status list as follows:

USB Status	Description
eDRVUSB_DETACHED	The USB has been detached.
eDRVUSB_ATTACHED	The USB has been attached.
eDRVUSB_POWERED	The USB is powered.
eDRVUSB_DEFAULT	The USB is in normal state.
eDRVUSB_ADDRESS	The USB is in ADDRESS state.
eDRVUSB_CONFIGURED	The USB is in CONFIGURATION state.
eDRVUSB_SUSPENDED	The USB is suspended.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

To return the current USB state.

Example

```
/* Get current USB state */
eUsbState = DrvUSB_GetUsbState();
```

```
if (eUsbState == eDRVUSB_DETACHED)
{
        /* USB unplug */
}
```

DrvUSB_SetUsbState

Prototype

void DrvUSB_SetUsbState(E_DRVUSB_STATE eUsbState)

Description

To change current USB state. Please refer to DrvUSB_GetUsbState for available states.

Parameter

eUsbState The USB state.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Set current USB state */
DrvUSB_SetUsbState(eDRVUSB_DETACHED);
```

DrvUSB_GetEpIdentity

Prototype

uint32_t DrvUSB_GetEpIdentity(uint32_t u32EpNum, uint32_t u32EpAttr)

Description

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

Parameter

u32EpNum	The endpoint number (0 ~ 15)
u32EpAttr	The endpoint number attribute. It could be EP_INPUT or EP_OUTPUT

Include

Driver/DrvUSB.h

Return Value

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

Example

/* Get the hardware endpoint identity of USB OUT endpoint 3 */

u32EpId = DrvUSB_GetEpIdentity(3, EP_OUTPUT);

DrvUSB_GetEpId

Prototype

uint32_t DrvUSB_GetEpId(uint32_t u32EpNum)

Description

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

Parameter

u32EpNum The endpoint address with direction information at bit 7.

Include

Driver/DrvUSB.h

Return Value

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

Example

/* Get the hardware endpoint identity of USB IN endpoint 4 */

u32EpId = DrvUSB_GetEpIdentity(0x84);

DrvUSB_DataOutTrigger

Prototype

int32_t DrvUSB_DataOutTrigger(uint32_t u32EpNum, uint32_t u32Size)

Description

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

Parameter

u32EpNum	The endpoint number (0~15)
u32Size	Maximum size want to receive from USB

Include

Driver/DrvUSB.h

Return Value

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

Example

/* Trigger endpoint number 2 to receive OUT packet of host and the maximum packet size is 64 bytes */

DrvUSB_DataOutTrigger(2, 64);

DrvUSB_GetOutData

Prototype

uint8_t * DrvUSB_GetOutData(uint32_t u32EpNum, uint32_t *u32Size)

Description

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

Parameter

u32EpNum	The endpoint number (0~15)
u32Size	Data size received from USB

Include

Driver/DrvUSB.h

Return Value

To return USB SRAM address.

Example

/* Get the buffer address and size of received data of endpoint number 2 */

pu8EpBuf = DrvUSB_GetOutData(2, &u32Size);

DrvUSB_DataIn

Prototype

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

Description

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

Parameter

u32EpNum	The endpoint number (0~15)
u8Buffer	The data buffer for DATA IN token
u32Size	The size of data buffer

Include

Driver/DrvUSB.h

Return Value

0	Successful
E_DRVUSB_SIZE_TOO_LONG	The size is larger than maximum packet size

Example

```
/* Prepare 2 bytes data for endpoint number 0 IN transaction. */
DrvUSB_DataIn(0, au8Data, 2);
```

DrvUSB_BusResetCallback

Prototype

void DrvUSB_BusResetCallback(void * pVoid)

Description

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

Parameter

pVoid	Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
    {NULL, NULL},                                    /* attach event callback */
    {NULL, NULL},                                    /* detach event callback */
    {DrvUSB_BusResetCallback, &g_HID_sDevice},    /* bus reset event callback */
    {NULL, NULL},                                    /* bus suspend event callback */
    {NULL, NULL},                                    /* bus resume event callback */
    {DrvUSB_CtrlSetupAck, &g_HID_sDevice},          /* setup event callback */
};
```

DrvUSB_InstallClassDevice

Prototype

void * DrvUSB_InstallClassDevice(S_DRVUSB_CLASS *sUsbClass)

Description

Register USB class device to USB driver.

Parameter

sUsbClass USB class structure pointer.

Include

Driver/DrvUSB.h

Return Value

Return USB driver pointer

Example

```
/* Register USB class device to USB driver. */
g_HID_sDevice.device = (void *)DrvUSB_InstallClassDevice(&sHidUsbClass);
```

DrvUSB_InstallCtrlHandler

Prototype

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

Description

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

Parameter

device	USB driver device pointer.
psCtrlCallbackEntry	Handler structure pointer.
u32RegCnt	Handler structure size.

Include

Driver/DrvUSB.h

Return Value

0	Success
E_DRVUSB_NULL_POINTER	Null function pointer

Example

```
/* Register ctrl pipe handler. */
i32Ret = DrvUSB_InstallCtrlHandler(g_HID_sDevice.device, g_asCtrlCallbackEntry,
sizeof(g_asCtrlCallbackEntry) / sizeof(g_asCtrlCallbackEntry[0]));
```

DrvUSB_CtrlSetupAck

Prototype

void DrvUSB_CtrlSetupAck(void * pArgu)

Description

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

Parameter

pArgu	Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* bus event call back */
S_DRVUSB_EVENT_PROCESS g_sBusOps[6] =
{
    {NULL, NULL},                    /* attach event callback */
    {NULL, NULL},                    /* detach event callback */
```

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

DrvUSB_CtrlDataInAck

Prototype

void DrvUSB_CtrlDataInAck(void * pArgu)

Description

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

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* USB event call back */
S_DRVUSB_EVENT_PROCESS g_sUsbOps[12] =
{
    {DrvUSB_CtrlDataInAck, &g_HID_sDevice},/* ctrl pipe0 (EP address 0) In ACK callback */
    {DrvUSB_CtrlDataOutAck, &g_HID_sDevice},/* ctrl pipe0 (EP address 0) Out ACK callback */
    {HID_IntInCallback, &g_HID_sDevice},/* EP address 1 In ACK callback */
    {NULL, NULL},                        /* EP address 1 Out ACK callback */
    {NULL, NULL},                        /* EP address 2 In ACK callback */
    {HID_IntOutCallback, &g_HID_sDevice},/* EP address 2 Out ACK callback */
    {NULL, NULL},                        /* EP address 3 In ACK callback */
    {NULL, NULL},                        /* EP address 3 Out ACK callback */
    {NULL, NULL},                        /* EP address 4 In ACK callback */
    {NULL, NULL},                        /* EP address 4 Out ACK callback */
    {NULL, NULL},                        /* EP address 5 In ACK callback */
    {NULL, NULL},                        /* EP address 5 Out ACK callback */
};
```

DrvUSB_CtrlDataOutAck

Prototype

void DrvUSB_CtrlDataOutAck(void * pArgu)

Description

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

Parameter

pArgu Parameter passed by g_sBusOps[].

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* USB event call back */
S_DRVUSB_EVENT_PROCESS g_sUsbOps[12] =
{
    {DrvUSB_CtrlDataInAck      , &g_HID_sDevice},/* ctrl pipe0 (EP address 0) In ACK callback */
    {DrvUSB_CtrlDataOutAck    , &g_HID_sDevice},/* ctrl pipe0 (EP address 0) Out ACK callback */
    {HID_IntInCallback           , &g_HID_sDevice},/* EP address 1 In ACK callback */
    {NULL, NULL},                                          /* EP address 1 Out ACK callback */
    {NULL, NULL},                                          /* EP address 2 In ACK callback */
    {HID_IntOutCallback              , &g_HID_sDevice},/* EP address 2 Out ACK callback */
    {NULL, NULL},                                          /* EP address 3 In ACK callback */
    {NULL, NULL},                                          /* EP address 3 Out ACK callback */
    {NULL, NULL},                                          /* EP address 4 In ACK callback */
    {NULL, NULL},                                          /* EP address 4 Out ACK callback */
    {NULL, NULL},                                          /* EP address 5 In ACK callback */
    {NULL, NULL},                                          /* EP address 5 Out ACK callback */
};
```

DrvUSB_CtrlDataInDefault

Prototype

void DrvUSB_CtrlDataInDefault(void * pVoid)

Description

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

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* If no control data IN callback installed, just use default one */
if (psEntry->pfnCtrlDataInCallback == NULL)
    psEntry->pfnCtrlDataInCallback = DrvUSB_CtrlDataInDefault;
```

DrvUSB_CtrlDataOutDefault

Prototype

void DrvUSB_CtrlDataOutDefault(void * pVoid)

Description

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

Parameter

pVoid	Parameter passed by DrvUSB_InstallCtrlHandler.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* If no control data OUT callback installed, just use default one */
if (psEntry->pfnCtrlDataOutCallback == NULL)
    psEntry->pfnCtrlDataOutCallback = DrvUSB_CtrlDataOutDefault;
```

DrvUSB_Reset

Prototype

void DrvUSB_Reset(uint32_t u32EpNum)

Description

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

Parameter

u32EpNum	The endpoint number to reset

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Reset endpoint number 2 */
DrvUSB_Reset(2);
```

DrvUSB_ClrCtrlReady

Prototype

void DrvUSB_ClrCtrlReady(void)

Description

Clear ctrl pipe ready flag that was set by MXPLD.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear control endpoint ready flag */
DrvUSB_ClrCtrlReady();
```

DrvUSB_ClrCtrlReadyAndTrigStall

Prototype

void DrvUSB_ClrCtrlReadyAndTrigStall(void);

Description

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

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear control pipe ready flag that was set by MXPLD and send STALL. */
DrvUSB_ClrCtrlReadyAndTrigStall();
```

DrvUSB_GetSetupBuffer

Prototype

uint32_t DrvUSB_GetSetupBuffer(void)

Description

Get setup buffer address of USB SRAM to read the received setup packet data.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Setup buffer address

Example

```
/* Get setup buffer address of USB SRAM. */
SetupBuffer = (uint8_t *)DrvUSB_GetSetupBuffer();
```

DrvUSB_GetFreeSRAM

Prototype

uint32_t DrvUSB_GetFreeSRAM(void)

Description

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

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Free USB SRAM address

Example

```
/* Get the base address of free USB SRAM */
u32BaseAddr = DrvUSB_GetFreeSRAM();
```

DrvUSB_EnableSelfPower

Prototype

void DrvUSB_EnableSelfPower(void)

Description

Enable self-power attribution of USB device.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Set a flag to note the USB device is self-power */
DrvUSB_EnableSelfPower();
```

DrvUSB_DisableSelfPower

Prototype

void DrvUSB_DisableSelfPower(void)

Description

Disable self-power attribution of USB device.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear the flag to note the USB device is not self-power */
DrvUSB_ DisableSelfPower ();
```

DrvUSB_IsSelfPowerEnabled

Prototype

int32_t DrvUSB_IsSelfPowerEnabled(void)

Description

Self-power is enable or disable.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

TRUE	The device is self-powered.
FALSE	The device is bus-powered.

Example

```
/* Check if the USB device is self-power */
if(DrvUSB_IsSelfPowerEnabled())
{
    /* The USB device is self-power */
}
```

DrvUSB_EnableRemoteWakeup

Prototype

void DrvUSB_EnableRemoteWakeup(void)

Description

Enable remote wakeup attribution of USB device.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Set the flag to note the USB device supports remote wakeup */
```

DrvUSB_EnableRemoteWakeup();

DrvUSB_DisableRemoteWakeup

Prototype

void DrvUSB_DisableRemoteWakeup(void)

Description

Disable remote wakeup attribution.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Clear the flag to note the USB device doesn't support remote wakeup */
DrvUSB_DisableRemoteWakeup();
```

DrvUSB_IsRemoteWakeupEnabled

Prototype

int32_t DrvUSB_IsRemoteWakeupEnabled (int32_t * pbVoid)

Description

Return remote wakeup is enabling or disable.

Parameter

None

Include

Driver/DrvUSB.h

Return Value

TRUE	Support remote wakeup
FALSE	Not support remote wakeup

Example

```
/* Check if the USB device supports remote wakeup. */
```

```
if(DrvUSB_ IsRemoteWakeupEnabled ())
{
    /* Remote wakeup enable flag is set */
}
```

DrvUSB_SetMaxPower

Prototype

int32_t DrvUSB_SetMaxPower(uint32_t u32MaxPower)

Description

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

Parameter

u32MaxPower Maximum power value

Include

Driver/DrvUSB.h

Return Value

0: Successful

<0: Wrong maximum value

Example

```
/* Set the maximum power is 150mA */
DrvUSB_SetMaxPower(75);
```

DrvUSB_GetMaxPower

Prototype

int32_t DrvUSB_GetMaxPower(void)

Description

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

Parameter

None

Include

Driver/DrvUSB.h

Return Value

Return the maximum power. (2mA unit)

Example

```
/* Get the maximum power */
i32Power = DrvUSB_GetMaxPower();
```

DrvUSB_EnableUSB

Prototype

void DrvUSB_EnableUSB(S_DRVUSB_DEVICE *psDevice)

Description

Enable USB, PHY and remote wakeup.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Enable USB, PHY and remote wakeup function. */
DrvUSB_EnableUSB(psDevice);
```

DrvUSB_DisableUSB

Prototype

void DrvUSB_DisableUSB(S_DRVUSB_DEVICE * psDevice)

Description

Disable USB, PHY but keep remote wakeup function on.

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Enable USB, PHY and remote wakeup function. */
DrvUSB_DisableUSB(psDevice);
```

DrvUSB_PreDispatchWakeupEvent

Prototype

void DrvUSB_PreDispatchWakeupEvent(S_DRVUSB_DEVICE *psDevice)

Description

Pre-dispatch wakeup event. This function does nothing and reserves for further usage

Parameter

psDevice   USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

N/A

DrvUSB_PreDispatchFDTEvent

Prototype

void DrvUSB_PreDispatchFDTEvent(S_DRVUSB_DEVICE * psDevice)

Description

Pre-dispatch plug-in and plug-out event

Parameter

psDevice   USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Pre-dispatch float-detection event. */

DrvUSB_PreDispatchFDTEvent(&gsUsbDevice);

DrvUSB_PreDispatchBusEvent

Prototype

void DrvUSB_PreDispatchBusEvent(S_DRVUSB_DEVICE *psDevice)

Description

Pre-dispatch BUS event

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Pre-dispatch bus event. */

DrvUSB_PreDispatchBusEvent(&gsUsbDevice);

DrvUSB_PreDispatchEPEvent

Prototype

void DrvUSB_PreDispatchEPEvent(S_DRVUSB_DEVICE * psDevice)

Description

Pre-dispatch EP event including IN ACK/IN NAK/OUT ACK/ISO end. This function is used to recognize endpoint events and record them for further processing of DrvUSB_DispatchEPEvent(). All EP event handlers are defined at g_sUsbOps[].

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Clear USB events individually instead of in total. Otherwise, incoming USB events may be cleared mistakenly. Pre-dispatch USB event. */

DrvUSB_PreDispatchEPEvent(&gsUsbDevice);

DrvUSB_DispatchWakeupEvent

Prototype

void DrvUSB_DispatchWakeupEvent(S_DRVUSB_DEVICE *psDevice)

Description

Dispatch wakeup event. This function does nothing and reserves for further usage.

Parameter

psDevice   USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

N/A

DrvUSB_DispatchMiscEvent

Prototype

void DrvUSB_DispatchMiscEvent(S_DRVUSB_DEVICE * psDevice)

Description

Dispatch Misc event. The event is set by attach/detach/bus reset/bus suspend/bus resume and setup ACK. Misc event's handler is defined at g_sBusOps[].

Parameter

psDevice   USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Parsing the MISC events and call relative handles */

DrvUSB_DispatchMiscEvent(&gsUsbDevice);

DrvUSB_DispatchEPEvent

Prototype

void DrvUSB_DispatchEPEvent(S_DRVUSB_DEVICE * psDevice)

Description

Dispatch EP event, the event is set by DrvUSB_PreDispatchEPEvent() including IN ACK/IN NAK/OUT ACK/ISO end. The EP event's handler is defined at g_sUsbOps[].

Parameter

psDevice USB driver device pointer

Include

Driver/DrvUSB.h

Return Value

None

Example

/* Parsing the endpoint events and call relative handlers */

DrvUSB_ DispatchEPEvent (&gsUsbDevice);

DrvUSB_CtrlSetupSetAddress

Prototype

void DrvUSB_CtrlSetupSetAddress(void * pVoid)

Description

Setup ACK handler for set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

/*ctrl pipe call back.*/

```
/*it will be call by DrvUSB_CtrlSetupAck, DrvUSB_CtrlDataInAck and DrvUSB_CtrlDataOutAck*/
/*if in ack handler and out ack handler is 0, default handler will be called */
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, SET_ADDRESS, DrvUSB_CtrlSetupSetAddress,
DrvUSB_CtrlDataInSetAddress, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupClearSetFeature

Prototype

void DrvUSB_CtrlSetupClearSetFeature(void * pVoid)

Description

Setup ACK handler for Clear feature command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, CLEAR_FEATURE, DrvUSB_CtrlSetupClearSetFeature, 0, 0,
&g_HID_sDevice}
};
```

DrvUSB_CtrlSetupGetConfiguration

Prototype

void DrvUSB_CtrlSetupGetConfiguration(void * pVoid)

Description

Setup ACK handler for Get configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, GET_CONFIGURATION, DrvUSB_CtrlSetupGetConfiguration, 0, 0,
&g_HID_sDevice}
};
```

DrvUSB_CtrlSetupGetStatus

Prototype

void DrvUSB_CtrlSetupGetStatus(void * pVoid)

Description

Setup ACK handler for Get status command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, GET_STATUS, DrvUSB_CtrlSetupGetStatus, 0, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupGetInterface

Prototype

void DrvUSB_CtrlSetupGetInterface(void * pVoid)

Description

Setup ACK handler for Get interface command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, GET_INTERFACE, DrvUSB_CtrlSetupGetInterface, 0, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupSetInterface

Prototype

void DrvUSB_CtrlSetupSetInterface(void * pVoid)

Description

Setup ACK handler for Set interface command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, SET_INTERFACE, DrvUSB_CtrlSetupSetInterface, 0, 0, &g_HID_sDevice}
};
```

DrvUSB_CtrlSetupSetConfiguration

Prototype

void DrvUSB_CtrlSetupSetConfiguration(void * pVoid)

Description

Setup ACK handler for Set configuration command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, SET_CONFIGURATION, DrvUSB_CtrlSetupSetConfiguration, 0, 0,
&g_HID_sDevice}
};
```

DrvUSB_CtrlDataInSetAddress

Prototype

void DrvUSB_CtrlDataInSetAddress(void * pVoid)

Description

Setup ACK handler for Set address command.

Parameter

pVoid Parameter passed by DrvUSB_InstallCtrlHandler

Include

Driver/DrvUSB.h

Return Value

None

Example

```
S_DRVUSB_CTRL_CALLBACK_ENTRY g_asCtrlCallbackEntry[] =
{ //request type,command,setup ack handler, in ack handler,out ack handler, parameter
    {REQ_STANDARD, SET_ADDRESS, DrvUSB_CtrlSetupSetAddress,
DrvUSB_CtrlDataInSetAddress, 0, &g_HID_sDevice}
};
```

DrvUSB_memcpy

Prototype

void DrvUSB_memcpy(uint8_t *pi8Dest, uint8_t *pi8Src, uint32_t u32Size)

Description

The USB buffer is recommended to be byte access thus this function is implemented by byte access.

Parameter

pi8Dest: Destination pointer

pi8Src: Source pointer

u32Size: Data size. The unit is byte.

Include

Driver/DrvUSB.h

Return Value

None

Example

```
/* Copy 64 bytes data from USB SRAM to SRAM */
DrvUSB_memcpy(0x20000800, 0x40060100, 64);
```

15. PDMA Driver

15.1. PDMA Introduction

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

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

15.2. PDMA Feature

The PDMA includes following features:

- Advanced Microcontroller Bus Architecture Advanced High-performance Bus (AMBA AHB) master/slave interface compatible, for data transfer and register read/write.
- PDMA support 32-bit source and destination addressing range address increment and fixed.
- Up to 9 channels of DMA. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix to know the number of DMA channel.

15.3. Constant Definition

Constant Name	Value	Description
CHANNEL_OFFSET	0x100	PDMA channel register offset

15.4. Type Definition

E_DRVPDMA_CHANNEL_INDEX

Enumeration identifier	Value	Description
eDRVPDMA_CHANNEL_0	0	PDMA channel 0

eDRVPDMA_CHANNEL_1	1	PDMA channel 1
eDRVPDMA_CHANNEL_2	2	PDMA channel 2
eDRVPDMA_CHANNEL_3	3	PDMA channel 3
eDRVPDMA_CHANNEL_4	4	PDMA channel 4
eDRVPDMA_CHANNEL_5	5	PDMA channel 5
eDRVPDMA_CHANNEL_6	6	PDMA channel 6
eDRVPDMA_CHANNEL_7	7	PDMA channel 7
eDRVPDMA_CHANNEL_8	8	PDMA channel 8

E_DRVPDMA_DIRECTION_SELECT

Enumeration identifier	Value	Description
eDRVPDMA_DIRECTION_INCREMENTED	0	Source/Destination Address Direction is incremented.
eDRVPDMA_DIRECTION_FIXED	2	Source/Destination Address Direction is fixed.

E_DRVPDMA_TRANSFER_WIDTH

Enumeration identifier	Value	Description
eDRVPDMA_WIDTH_32BITS	0	One word is transferred for every PDMA operation in IP-to-Memory/Memory-to-IP mode.
eDRVPDMA_WIDTH_8BITS	1	One byte is transferred for every PDMA operation in IP-to-Memory/Memory-to-IP mode.
eDRVPDMA_WIDTH_16BITS	2	Half word is transferred for every PDMA operation in IP-to-Memory/Memory-to-IP mode.

E_DRVPDMA_INT_ENABLE

Enumeration identifier	Value	Description
eDRVPDMA_TABORT	1	Target abort interrupt/flag
eDRVPDMA_BLKD	2	Transferred done interrupt/flag

E_DRVPDMA_APB_DEVICE

Enumeration identifier	Value	Description
eDRVPDMA_SPI0	0	PDMA source/destination APB device is SPI0
eDRVPDMA_SPI1	1	PDMA source/destination APB device is SPI1
eDRVPDMA_SPI2	2	PDMA source/destination APB device is SPI2
eDRVPDMA_SPI3	3	PDMA source/destination APB device is SPI3
eDRVPDMA_UART0	4	PDMA source/destination APB device is UART0
eDRVPDMA_UART1	5	PDMA source/destination APB device is UART1
eDRVPDMA_ADC	7	PDMA source/destination APB device is ADC
eDRVPDMA_I2S	8	PDMA source/destination APB device is I2S

E_DRVPDMA_APB_RW

Enumeration identifier	Value	Description
eDRVPDMA_READ_APB	0	Read data from APB device to memory
eDRVPDMA_WRITE_APB	1	Write data from memory to APB device

E_DRVPDMA_MODE

Enumeration identifier	Value	Description
eDRVPDMA_MODE_MEM2MEM	0	PDMA mode is Memory-to-Memory
eDRVPDMA_MODE_APB2MEM	1	PDMA mode is APB device-to-Memory
eDRVPDMA_MODE_MEM2APB	2	PDMA mode is Memory-to-APB device

15.5. Functions

DrvPDMA_Init

Prototype

void

DrvPDMA_Init (void);

Description

The function is used to enable AHB PDMA engine clock.

Parameter

None

Include

Driver/DrvPDMA.h

Return Value

None

Example

```
/* Enable AHB PDMA engine clock */
DrvPDMA_Init();
```

DrvPDMA_Close

Prototype

void DrvPDMA_Close (void);

Description

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

Parameter

None

Include

Driver/DrvPDMA.h

Return Value

None

Example

```
/* Disable all PDMA channel clock and AHB PDMA clock */
DrvPDMA_Close();
```

DrvPDMA_CHEnableTransfer

Prototype

```
int32_t
DrvPDMA_CHEnablelTransfer(
    E_DRVPDMA_CHANNEL_INDEX eChannel
);
```

Description

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

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

E_DRVPDMA_ERR_PORT_INVALID: Invalid port number

Example

```
/* Enable PDMA channel0 and enable channel0 data read/write transfer */
DrvPDMA_CHEnableTransfer(eDRVPDMA_CHANNEL_0);
```

DrvPDMA_CHSoftwareReset

Prototype

int32_t

DrvPDMA_CHSoftwareReset(

E_DRVPDMA_CHANNEL_INDEX eChannel

);

Description

The function is used to do software reset specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success.

E_DRVPDMA_ERR_PORT_INVALID: Invalid port number

Note

The function will reset the specified channel internal state machine and pointers. The contents of control register will not be cleared.

Example

/* Software reset PDMA channel0 and get returned value */

int32_t i32RetVal_CH0SoftwareReset;

i32RetVal_CH0SoftwareReset = DrvPDMA_CH0SoftwareReset(eDRVPDMA_CHANNEL_0);

DrvPDMA_Open

Prototype

int32_t

DrvPDMA_Open(

E_DRVPDMA_CHANNEL_INDEX sChannel,

STR_PDMA_T *sParam

);

Description

The function configures PDMA setting

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

sParam [in]

The struct parameter to configure PDMA,

It includes

sSrcCtrl.u32Addr : Source Address.**(must be word alignment)**

sSrcCtrl.eAddrDirection: Source Address Direction

eDRVPDMA_DIRECTION_INCREMENTED: Source address direction is incremented

eDRVPDMA_DIRECTION_FIXED: Source address direction is fixed

sDestCtrl.u32Addr: Destination Address.**(must be word alignment)**

sDestCtrl.eAddrDirection: Destination Address Direction. It could be eDRVPDMA_DIRECTION_INCREMENTED / eDRVPDMA_DIRECTION_FIXED.

eDRVPDMA_DIRECTION_INCREMENTED: Destination address direction is incremented

eDRVPDMA_DIRECTION_FIXED: Destination address direction is fixed

u8TransWidth: Peripheral Transfer Width. This field is meaningful only when the operation mode setting are APB to memory or memory to APB. It could be eDRVPDMA_WIDTH_8BITS / eDRVPDMA_WIDTH_16BITS / eDRVPDMA_WIDTH_32BITS.

eDRVPDMA_WIDTH_8BITS: One byte (8 bits) is transferred for every PDMA operation.

eDRVPDMA_WIDTH_16BITS: One half-word (16 bits) is transferred for every PDMA operation.

eDRVPDMA_WIDTH_32BITS: One word (32 bits) is transferred for every PDMA operation.

u8Mode: Operation Mode

eDRVPDMA_MODE_MEM2MEM: Memory to memory mode.

eDRVPDMA_MODE_APB2MEM: APB to memory mode.

eDRVPDMA_MODE_MEM2APB: memory to APB mode.

i32ByteCnt: PDMA Transfer Byte Count

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: Invalid port number

Example

```
/*---------------------------------------------------------------------------------------*/
```

```
/* Set PDMA channel1 to UART1 TX------------------------------------------ */
/* Set PDMA transfer done callback function and trigger PDMA function. */
/*--------------------------------------------------------------------------------------*/
/* PDMA Setting */
UARTPort = UART1_BASE;
DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_1,eDRVPDMA_UART1,eDR
VPDMA_WRITE_APB);
/* CH1 TX Setting */
sPDMA.sSrcCtrl.u32Addr = (uint32_t)SrcArray;
sPDMA.sDestCtrl.u32Addr = UARTPort;
sPDMA.u8TransWidth = eDRVPDMA_WIDTH_8BITS;
sPDMA.u8Mode = eDRVPDMA_MODE_MEM2APB;
sPDMA.sSrcCtrl.eAddrDirection = eDRVPDMA_DIRECTION_INCREMENTED;
sPDMA.sDestCtrl.eAddrDirection = eDRVPDMA_DIRECTION_FIXED;
sPDMA.i32ByteCnt = UART_TEST_LENGTH;
DrvPDMA_Open(eDRVPDMA_CHANNEL_1,&sPDMA);

/* Enable INT */
DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_1, eDRVPDMA_BLKD );
/* Install Callback function */
DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_1,eDRVPDMA_BLKD,(PFN_DRV
PDMA_CALLBACK));
/* Enable UART PDMA and Trigger PDMA specified Channel */
DrvPDMA_CHEnableTransfer(eDRVPDMA_CHANNEL_1);
```

DrvPDMA_ClearIntFlag

Prototype

void

DrvPDMA_ClearIntFlag(

E_DRVPDMA_CHANNEL_INDEX eChannel,

E_DRVPDMA_INT_FLAG eIntFlag

) ;

Description

The function is used to clear interrupt status for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntFlag [in] Interrupt source:

eDRVPDMA_TABORT: Read/Write Target Abort

eDRVPDMA_BLKD: Block Transfer Done

Include

Driver/DrvPDMA.h

Return Value

None

Example

/* Clear channel0 block transfer done interrupt flag. */

DrvPDMA_ClearIntFlag(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD_FLAG);

/* Clear channel1 read/write target abort interrupt flag */

DrvPDMA_ClearIntFlag(eDRVPDMA_CHANNEL_1, eDRVPDMA_TABORT);

DrvPDMA_PollInt

Prototype

int32_t

DrvPDMA_PollInt(

E_DRVPDMA_CHANNEL_INDEX eChannel,

E_DRVPDMA_INT_FLAG eIntFlag

) ;

Description

The function is used to polling channel interrupt status

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntFlag [in] Intterrupt source:

eDRVPDMA_TABORT: Read/Write Target Abort

eDRVPDMA_BLKD: Block Transfer Done

Include

Driver/DrvPDMA.h

Return Value

True: Interrupt status is set.

False: Interrupt status is clear.

Example

/* Get Channel 5 transfer done interrupt status */

int32_t i32Channel5TransferDone;

/* Enable INT */

```
DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_5, eDRVPDMA_BLKD );
…
/* Check channel5 transfer done interrupt flag */
if(DrvPDMA_PollInt(eDRVPDMA_CHANNEL_5, eDRVPDMA_BLKD_FLAG)==TRUE)
    printf("Channel5 block transfer done interrupt flag is set!!\n")
else
    printf("Channel5 block transfer done interrupt flag is not set!!\n")
```

DrvPDMA_SetAPBTransferWidth

Prototype

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

Description

The function is used to set APB transfer width for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eTransferWidth [in]

eDRVPDMA_WIDTH_32BITS: One word (32 bits) is transferred for every PDMA operation.

eDRVPDMA_WIDTH_8BITS: One byte (8 bits) is transferred for every PDMA operation.

eDRVPDMA_WIDTH_16BITS: One half-word (16 bits) is transferred for every PDMA operation.

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: invalid port number

Note

This function is meaningful only when PDMA mode select is APB-to-Memory or Memory-to-APB mode.

Example

/* Set chaneel 7 peripheral bus width to 8 bits.*/

DrvPDMA_SetAPBTransferWidth(eDRVPDMA_CHANNEL_7, eDRVPDMA_WIDTH_8BITS)

DrvPDMA_SetCHForAPBDevice

Prototype

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

Description

The function is used to select PDMA channel for APB device

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eDevice [in]

Channel for APB device. It includes of

eDRVPDMA_SPI0~3,eDRVPDMA_UART0~1, eDRVPDMA_ADC, eDRVPDMA_I2S

eRWAPB [in] : PDMA transfer data direction

eDRVPDMA_WRITE_APB: PDMA transfer data from memory to specified APB.

eDRVPDMA_READ_APB: PDMA transfer data from specified APB to memory.

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: Invalid port

E_DRVPDMA_FALSE_INPUT: Invalid APB device

Example

/*Set PDMA channel1 to UART1 TX port*/

DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_1,eDRVPDMA_UART1,eDRVPDMA_WRITE_APB);

/*Set PDMA channel0 to SPI0 RX port*/

DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_0,eDRVPDMA_SPI0,eDRVPDMA_READ_APB);

DrvPDMA_SetSourceAddress

Prototype

int32_t

DrvPDMA_SetSourceAddress(

E_DRVPDMA_CHANNEL_INDEX eChannel,

uint32_t u32SourceAddr

) ;

Description

The function is used to set source address for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

u32SourceAddress [in]

Source address

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: Invalid port number

Example

/* Set channel 0 source address to specified address.*/

DrvPDMA_ SetSourceAddress (eDRVPDMA_CHANNEL_0, 0x20001000);

DrvPDMA_SetDestAddress

Prototype

int32_t

DrvPDMA_SetDestAddress(

E_DRVPDMA_CHANNEL_INDEX eChannel,

uint32_t u32DestAddr

) ;

Description

The function is used to set destination address for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

u32DestAddress [in]

Destination address

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: Invalid port number

Example

/* Set channel 0 destination address to specified address.*/

DrvPDMA_ SetDestAddress (eDRVPDMA_CHANNEL_0, 0x20001200);

DrvPDMA_DisableInt

Prototype

int32_t

DrvPDMA_DisableInt(

E_DRVPDMA_CHANNEL_INDEX eChannel,

E_DRVPDMA_INT_ENABLE eIntSource

);

Description

The function is used to disable interrupt for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]: Interrupt source

eDRVPDMA_TABORT: Read/Write Target Abort

eDRVPDMA_BLKD: Block Transfer Done

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: invalid port number

Example

/*Disable channel3 read/write target abort interrupt*/

DrvPDMA_DisableInt(eDRVPDMA_CHANNEL_3, eDRVPDMA_TABORT);

DrvPDMA_EnableInt

Prototype

int32_t

DrvPDMA_EnableInt(

E_DRVPDMA_CHANNEL_INDEX eChannel,

E_DRVPDMA_INT_ENABLE eIntSource

);

Description

The function is used to enable Interrupt for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]: Interrupt source

eDRVPDMA_TABORT: Read/Write Target Abort

eDRVPDMA_BLKD: Block Transfer Done

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

E_DRVPDMA_ERR_PORT_INVALID: invalid port number

Example

/*Enable channel0 block transfer done interrupt.*/

DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD);

DrvPDMA_GetAPBTransferWidth

Prototype

int32_t

DrvPDMA_GetAPBTransferWidth(

E_DRVPDMA_CHANNEL_INDEX eChannel);

Description

The function is used to get peripheral transfer width from specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

0: One word (32 bits) is transferred for every PDMA operation.

1: One byte (8 bits) is transferred for every PDMA operation.

2: One half-word (16 bits) is transferred for every PDMA operation.

E_DRVPDMA_ERR_PORT_INVALID: invalid port number

Note

This function is meaningful only when PDMA mode selection is APB-to-Memory/Memory-to-APB mode.

Example

```
/*Get peripheral transfer width from channel3*/
int32_t i32Channel3APBTransferWidth;
i32Channel3APBTransferWidth = DrvPDMA_GetAPBTransferWidth(eDRVPDMA_CHANNEL_3);
```

DrvPDMA_GetCHForAPBDevice

Prototype

```
Int32_t
DrvPDMA_GetCHForAPBDevice(
    E_DRVPDMA_APB_DEVICE eDevice,
    E_DRVPDMA_APB_RW eRWAPB
);
```

Description

The function is used to get PDMA channel for specified APB device

Parameter

eDevice [in]

Channel for APB device. It includes of

eDRVPDMA_SPI0~3,eDRVPDMA_UART0~1, eDRVPDMA_ADC, eDRVPDMA_I2S

eRWAPB [in] : Specify APB direction

eDRVPDMA_READ_APB: APB to memory

eDRVPDMA_WRITE_APB: memory to APB

Include

Driver/DrvPDMA.h

Return Value

0: channel 0

1: channel 1

2: channel 2

3: channel 3

4: channel 4

5: channel 5

6: channel 6

7: channel 7

8: channel 8

E_DRVPDMA_FALSE_INPUT: Wrong parameter

Others: Reserved

Note

If APBDevice don’t be assigned to any channel, the default return value will be 15(0xF).

Example

/* Get UART0 RX PDMA channel*/

int32_t i32GetChannel4APBDevice;

i32GetChannel4APBDevice = DrvPDMA_GetCHForAPBDevice(eDRVPDMA_UART0, eDRVPDMA_READ_APB);

DrvPDMA_GetCurrentDestAddr

Prototype

uint32_t

DrvPDMA_GetCurrentDestAddr(

E_DRVPDMA_CHANNEL_INDEX eChannel

);

Description

The function is used to get current destination address from specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Current destination address

Note

Current destination address indicates the destination address where the PDMA transfer is just occurring.

Example

```
/*Get Channel5 current destination address;*/
uint32_t u32Channel5CurDestAddr;
u32Channel5CurDestAddr = DrvPDMA_GetCurrentDestAddr(eDRVPDMA_CHANNEL_5);
```

DrvPDMA_GetCurrentSourceAddr

Prototype

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

Description

The function is used to get current source address from specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Current source address register indicates the source address where the PDMA transfer is just occurring.

Example

```
/*Get channel7 current source address.*/
```

uint32_t u32Channel7CurrentSourceAddress;

u32Channel7CurrentSourceAddress =
DrvPDMA_GetCurrentSourceAddr(eDRVPDMA_CHANNEL_7);

DrvPDMA_GetRemainTransferCount

Prototype

uint32_t

DrvPDMA_GetRemainTransferCount(

E_DRVPDMA_CHANNEL_INDEX eChannel

);

Description

The function is used to get current remained byte count of specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Current remained byte count.

Note

If user set transfer byte to 64 bytes, the current byte count will be 64bytes in the beginning of transfer. After PDMA transferred 4 bytes to memory, user can issue this API and will get current remained byte count value which is 60 bytes.

Example

Get Channel0 Current remained byte count

uint32_t u32CurrentRemainedByteCount;

u32CurrentRemainedByteCount =
DrvPDMA_GetRemainTransferCount(eDRVPDMA_CHANNEL_0);

DrvPDMA_GetInternalBufPointer

Prototype

uint32_t

DrvPDMA_GetInternalBufPointer(

E_DRVPDMA_CHANNEL_INDEX eChannel

);

Description

The function is used to get internal buffer pointer for specified channel

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

E_DRVPDMA_ERR_PORT_INVALID : invalid port

0x01 : internal pointer point to byte1(one byte remained in PDMA buffer)

0x03 : internal pointer point to byte2(two byte remained in PDMA buffer)

0x07: internal pointer point to byte3(three byte remained in PDMA buffer)

0x0F: internal pointer point to byte4(There is no more data remained in PDMA buffer)

Example

```
/*Get channel0 internal buffer data point to know how many bytes remained in PDMA shared
buffer and print the internal buffer values.*/


uint32_t u32PdmaInternalBufferPoint ;
uint32_t u32PdmaSharedBufferData;
uint8_t au8EffectiveSharedBufferData[4] ;
u32PdmaInternalBufferPoint = DrvPDMA_GetInternalBufPointer(eDRVPDMA_CHANNEL_0)
if(u32PdmaInternalBufferPoint==0x01)
{
   printf(''Because the Pdma Internal bufer point is 0x01 which indicates that there is only one
   byte data remained in PDMA buffer!'')
   u32PdmaSharedBufferData = DrvPDMA_GetSharedBufData(eDRVPDMA_CHANNEL_0) ;
   au8EffectiveSharedBufferData [0] = (uint8_t)(u32PdmaSharedBufferData&0x000000FF) ;
   printf(''PDMA Shared buffer data is %x\n'', au8EffectiveSharedBufferData [0]) ;
   }
else if(u32PdmaInternalBufferPoint==0x03)
{
   printf(''Because the Pdma Internal bufer point is 0x03 which indicates that there is two bytes
   data remained in PDMA buffer!'')
   u32PdmaSharedBufferData = DrvPDMA_GetSharedBufData(eDRVPDMA_CHANNEL_0) ;
   au8EffectiveSharedBufferData [0] = (uint8_t)(u32PdmaSharedBufferData&0x000000FF) ;
   au8EffectiveSharedBufferData [1] = (uint8_t)(u32PdmaSharedBufferData&0x0000FF00) ;
```

printf(‘’PDMA Shared buffer data are %x and %x\n’’, au8EffectiveSharedBufferData [0], au8EffectiveSharedBufferData [1]) ;

}

else if(u32PdmaInternalBufferPoint==0x07)

{

printf("Because the Pdma Internal bufer point is 0x07 which indicates that there is three bytes data remained in PDMA buffer!’’)

u32PdmaSharedBufferData = DrvPDMA_GetSharedBufData(eDRVPDMA_CHANNEL_0) ;

au8EffectiveSharedBufferData [0] = (uint8_t)(u32PdmaSharedBufferData&0x000000FF) ;

au8EffectiveSharedBufferData [1] = (uint8_t)(u32PdmaSharedBufferData&0x0000FF00) ;

au8EffectiveSharedBufferData [2] = (uint8_t)(u32PdmaSharedBufferData&0x00FF0000) ;

printf("PDMA Shared buffer data are %x,%x and%x\n’’,au8EffectiveSharedBufferData[0], au8EffectiveSharedBufferData [1] , au8EffectiveSharedBufferData [2]) ;

}

else if(u32PdmaInternalBufferPoint==0x0F)

{

printf(‘’Because the Pdma Internal bufer point is 0x0F which indicates that there is no data in PDMA buffer!’’)

}

DrvPDMA_GetSharedBufData

Prototype

uint32_t

DrvPDMA_GetSharedBufData(

E_DRVPDMA_CHANNEL_INDEX eChannel,

)

Description

The function is used to get shared buffer content from specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Shared buffer data

Example

Please refer to DrvPDMA_GetInternalBufPointer() example.

DrvPDMA_GetTransferLength

Prototype

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

Description

The function is used to get channel transfer length setting. The unit of * pu32TransferLength is byte.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

pu32TransferLength [in]

The data pointer to save transfer length

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

Example

```
/* Get the transfer byte count setting of channel0.*/
uint32_t u32GetTransferByteCountSetting;
DrvPDMA_GetTransferLength(eDRVPDMA_CHANNEL_0,
&u32GetTransferByteCountSetting);
```

DrvPDMA_GetSourceAddress

Prototype

```
uint32_t
```

DrvPDMA_GetSourceAddress (

    E_DRVPDMA_CHANNEL_INDEX eChannel,

)

Description

The function is used to get source address for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Source address

Example

```
/* Get the source address of channel0 */
uint32_t u32GetSourceAddress;
u32GetSourceAddress = DrvPDMA_GetSourceAddress(eDRVPDMA_CHANNEL_0);
```

DrvPDMA_GetDestAddress

Prototype

uint32_t

DrvPDMA_GetDestAddress (

    E_DRVPDMA_CHANNEL_INDEX eChannel,

)

Description

The function is used to get destination address for specified channel.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

Destination address

Example

/* Get the destination address of channel0 */

uint32_t u32GetDestAddress;

u32GetDestAddress = DrvPDMA_GetDestAddress(eDRVPDMA_CHANNEL_0);

DrvPDMA_InstallCallBack

Prototype

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

Description

The function is used to install call back function for specified channel and interrupt source.

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]: Interrupt source

eDRVPDMA_TABORT: read/write target abort

eDRVPDMA_BLKD: block transfer done

pfncallback [in]

The callback function pointer

Include

Driver/DrvPDMA.h

Return Value

E_SUCCESS: Success

Example

Please refer to DrvPDMA_Open() sample code.

DrvPDMA_IsCHBusy

Prototype

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

Description

The function is used to Get Channel Enable/Disable status

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

Include

Driver/DrvPDMA.h

Return Value

TRUE: The channel is busy

FALSE: The channel is un-used.

E_DRVPDMA_ERR_PORT_INVALID: invalid port number

Example

```
/* Get channel0 bus status.*/
int32_t i32Channel0BusStatus;
i32Channel0BusStatus = if(DrvPDMA_IsCHBusy(eDRVPDMA_CHANNEL_0);
if(i32Channel0BusStatus== TRUE)
printf("Channel0 bus is busy!!\n");
else if(i32Channel0BusStatus== FALSE)
printf("Channel0 bus is not busy!!\n");
else if(i32Channel0BusStatus== E_DRVPDMA_ERR_PORT_INVALID)
printf("invalid port!!\n");
```

DrvPDMA_IsIntEnabled

Prototype

```
int32_t
```

DrvPDMA_IsIntEnabled(

E_DRVPDMA_CHANNEL_INDEX eChannel,

E_DRVPDMA_INT_ENABLE eIntSource

);

Description

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

Parameter

eChannel [in]

Specify eDRVPDMA_CHANNEL_0~8

eIntSource [in]

Interrupt source: eDRVPDMA_TABORT/eDRVPDMA_BLKD

Include

Driver/DrvPDMA.h

Return Value

TRUE: The specified interrupt source of specified channel is enable.

FALSE: The specified interrupt source of specified channel is disable.

Include

Driver/DrvPDMA.h

Example

```
int32_t i32IsIntEnable;
i32IsIntEnable=DrvPDMA_IsIntEnabled(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD)
if(i32IsIntEnable == TRUE )
printf("Channel0 Block transfer Done interrupt is enable!\n");
else if(i32IsIntEnable == FALSE )
printf("Channel0 Block transfer Done interrupt is disable!\n");
```

DrvPDMA_GetVersion

Prototype

int32_t

DrvPDMA_GetVersion (void);

Description

Return the current version number of driver.

Include

Driver/DrvPDMA.h

Return Value

PDMA driver current version number:

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

Example

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

int32_t i32PDMAVersionNum;

i32PDMAVersionNum = DrvPDMA_GetVersion();
```

16. I2S Driver

16.1. I2S Introduction

This I2S controller consists of IIS protocol to interface with external audio CODEC. Two 8 word deep FIFO for read path and write path respectively and is capable of handling 8-bit, 16-bit, 24-bit and 32-bit data size. DMA controller handles the data movement between FIFO and memory.

16.2. I2S Feature

- Operate as either master mode or slave mode.
- Capable of handling 8, 16, 24, and 32 bit data size.
- Support mono and stereo audio data.
- Support I2S and MSB justified data format.
- Two 8 word FIFO data buffers are provided. One for transmit and one for receive.
- Generate interrupt request when Tx/Rx FIFO level crosses a programmable boundary.
- Two DMA requests. One for transmit and one for receive.

16.3. Constant Definition

Constant Name	Value	Description
DRVI2S_DATABIT_8	0x00	Data size is 8 bit
DRVI2S_DATABIT_16	0x01	Data size is 16 bit
DRVI2S_DATABIT_24	0x02	Data size is 24 bit
DRVI2S_DATABIT_32	0x03	Data size is 32 bit
DRVI2S_MONO	0x01	Data is mono format
DRVI2S_STEREO	0x00	Data is stereo format
DRVI2S_FORMAT_MSB	0x01	MSB justified data format
DRVI2S_FORMAT_I2S	0x00	I2S data format
DRVI2S_MODE_SLAVE	0x01	I2S operates as slave mode
DRVI2S_MODE_MASTER	0x00	I2S operates as master mode
DRVI2S_FIFO_LEVEL_WORD_0	0x00	FIFO threshold level is 0 word
DRVI2S_FIFO_LEVEL_WORD_1	0x01	FIFO threshold level is 1 word
DRVI2S_FIFO_LEVEL_WORD_2	0x02	FIFO threshold level is 2 word
DRVI2S_FIFO_LEVEL_WORD_3	0x03	FIFO threshold level is 3 word
DRVI2S_FIFO_LEVEL_WORD_4	0x04	FIFO threshold level is 4 word
DRVI2S_FIFO_LEVEL_WORD_5	0x05	FIFO threshold level is 5 word
DRVI2S_FIFO_LEVEL_WORD_6	0x06	FIFO threshold level is 6 word
DRVI2S_FIFO_LEVEL_WORD_7	0x07	FIFO threshold level is 7 word
DRVI2S_FIFO_LEVEL_WORD_8	0x08	FIFO threshold level is 8 word
DRVI2S_EXT_12M	0	I2S clock source is from external 12MHz crystal clock
DRVI2S_PLL	1	I2S clock source is from PLL clock
DRVI2S_HCLK	2	I2S clock source is from HCLK
DRVI2S_INTERNAL_22M	3	I2S clock source is from internal 22MHz RC clock

16.4. Type Definition

E_I2S_CHANNEL

Enumeration identifier	Value	Description
I2S_LEFT_CHANNEL	0	I2S for left channel
I2S_RIGHT_CHANNEL	1	I2S for right channel

E_I2S_CALLBACK_TYPE

Enumeration identifier	Value	Description
I2S_RX_UNDERFLOW	0	For RX FIFO underflow interrupt
I2S_RX_OVERFLOW	1	For RX FIFO overflow interrupt
I2S_RX_FIFO_THRESHOLD	2	For RX FIFO threshold level interrupt
I2S_TX_UNDERFLOW	8	For TX FIFO underflow interrupt
I2S_TX_OVERFLOW	9	For TX FIFO overflow interrupt
I2S_TX_FIFO_THRESHOLD	10	For TX FIFO threshold level interrupt
I2S_TX_RIGHT_ZERO_CROSS	11	For TX right channel zero cross interrupt
I2S_TX_LEFT_ZERO_CROSS	12	For TX left channel zero cross interrupt

16.5. Macro Functions

_DRVI2S_WRITE_TX_FIFO

Prototype

```
static __inline
void _DRVI2S_WRITE_TX_FIFO (
    uint32_t    u32Data
);
```

Description

Write word data to Tx FIFO.

Parameter

u32Data [in]

Word data to Tx FIFO.

Include

Driver/DrvI2S.h

Return Value

None

Example

```
/* Write word data 0x12345678 into I2S Tx FIFO */
_DRVI2S_WRITE_TX_FIFO (0x12345678);
```

_DRVI2S_READ_RX_FIFO

Prototype

```
static __inline
uint32_t
_DRVI2S_READ_RX_FIFO (
    void
);
```

Description

Read out word data from Rx FIFO.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

Word data from Rx FIFO.

Example

```
uint32_t u32data;
/* Read word data from I2S Rx FIFO */
u32data = _DRVI2S_READ_RX_FIFO ();
```

_DRVI2S_READ_TX_FIFO_LEVEL

Prototype

```
static __inline
uint32_t
_DRVI2S_READ_TX_FIFO_LEVEL (
    void
);
```

Description

Get word data number in Tx FIFO.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

0~8: word data in Tx FIFO

Example

```
uint32_t u32len;
/* Get word data number in Tx FIFO */
u32len = _DRVI2S_READ_TX_FIFO_LEVEL ();
```

_DRVI2S_READ_RX_FIFO_LEVEL

Prototype

```
static __inline
uint32_t
_DRVI2S_READ_RX_FIFO_LEVEL (
    void
);
```

Description

Get word data number in Rx FIFO.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

0~8: word data in Rx FIFO

Example

```
uint32_t u32len;
/* Get word data number in Rx FIFO */
u32len = _DRVI2S_READ_RX_FIFO_LEVEL ( );
```

16.6. Functions

DrvI2S_Open

Prototype

int32_t DrvI2S_Open (S_DRVI2S_DATA_T *sParam);

Description

This function is used to enable I2S clock and function, and configure the data length/data format/FIFO threshold level/BCLK (Bit Clock). The data and audio formats are shown in I2S Operation and FIFO Operation of I2S Section in TRM. For master mode, *I2S_BCLK* and *I2S_LRCLK* pins are output mode; for slave mode, *I2S_BCLK* and *I2S_LRCLK* pins are input mode. Also, the I2S signals (*I2S_BCLK* and *I2S_LRCLK*) are shown in I2S Block Diagram of I2S Section in TRM.

Parameter

***sParam [in]**

It includes the following parameter

u32SampleRate: Sampling rate. The setting takes effect when I2S operates as master mode.

u8WordWidth: 8, 16, 24, or 32 bit data size - DRVI2S_DATABIT_8 / DRVI2S_DATABIT_16 / DRVI2S_DATABIT_24 / DRVI2S_DATABIT_32

u8AudioFormat: Support mono or stereo audio data - DRVI2S_MONO / DRVI2S_STEREO

u8DataFormat: Support I2S and MSB justified data format - DRVI2S_FORMAT_I2S / DRVI2S_FORMAT_MSB

u8Mode: Operate as master or slave mode - DRVI2S_MODE_MASTER / DRVI2S_MODE_SLAVE

u8TxFIFOThreshold: Tx FIFO threshold level - DRVI2S_FIFO_LEVEL_WORD_0 /
DRVI2S_FIFO_LEVEL_WORD_1 /
DRVI2S_FIFO_LEVEL_WORD_2 /
DRVI2S_FIFO_LEVEL_WORD_3 /
DRVI2S_FIFO_LEVEL_WORD_4 /
DRVI2S_FIFO_LEVEL_WORD_5 /
DRVI2S_FIFO_LEVEL_WORD_6 /
DRVI2S_FIFO_LEVEL_WORD_7

u8RxFIFOThreshold: Rx FIFO threshold level - DRVI2S_FIFO_LEVEL_WORD_1 /
DRVI2S_FIFO_LEVEL_WORD_2 /
DRVI2S_FIFO_LEVEL_WORD_3 /
DRVI2S_FIFO_LEVEL_WORD_4 /
DRVI2S_FIFO_LEVEL_WORD_5 /
DRVI2S_FIFO_LEVEL_WORD_6 /
DRVI2S_FIFO_LEVEL_WORD_7 /
DRVI2S_FIFO_LEVEL_WORD_8

Include

Driver/DrvI2S.h

Return Value

0 Success

Example

```
S_DRVI2S_DATA_T st;

st.u32SampleRate = 16000; /* Sampling rate is 16ksps */
st.u8WordWidth = DRVI2S_DATABIT_16; /* Data length is 16-bit */
st.u8AudioFormat = DRVI2S_STEREO; /* Stereo format */
st.u8DataFormat = DRVI2S_FORMAT_I2S; /* I2S data format */
st.u8Mode = DRVI2S_MODE_MASTER; /* Operate as master mode */
/* Tx FIFO threshold level is 0 word data */
st.u8TxFIFOThreshold = DRVI2S_FIFO_LEVEL_WORD_0;
/* Rx FIFO threshold level is 8 word data */
st.u8RxFIFOThreshold = DRVI2S_FIFO_LEVEL_WORD_8;
/* Enable I2S and configure its settings */
DrvI2S_Open (&st);
```

DrvI2S_Close

Prototype

void DrvI2S_Close (void);

Description

Close I2S controller and disable I2S clock.

Include

Driver/DrvI2S.h

Return Value

None

Example

```
DrvI2S_Close ( );     /* Disable I2S */
```

DrvI2S_EnableInt

Prototype

int32_t DrvI2S_EnableInt (E_I2S_CALLBACK_TYPE Type, I2S_CALLBACK callbackfn);

Description

To enable I2S interrupt function and install relative call back function in I2S interrupt handler.

Parameter

Type [in]

There are eight types for call back function.

I2S_RX_UNDERFLOW: Rx FIFO underflow

I2S_RX_OVERFLOW: Rx FIFO overflow.

I2S_RX_FIFO_THRESHOLD: Data word in Rx FIFO is higher than Rx threshold level.

I2S_TX_UNDERFLOW: Tx FIFO underflow.

I2S_TX_OVERFLOW: Tx FIFO overflow

I2S_TX_FIFO_THRESHOLD: Data word in Tx FIFO is less than Tx threshold level.

I2S_TX_RIGHT_ZERO_CROSS: Tx right channel zero cross.

I2S_TX_LEFT_ZERO_CROSS: Tx left channel zero cross.

callbackfn [in]

Call back function name for specified interrupt event.

Include

Driver/DrvI2S.h

Return Value

0: Succeed

<0: Failed

Example

/* Enable Rx threshold level interrupt and install its callback function */

DrvI2S_EnableInt (I2S_RX_FIFO_THRESHOLD, Rx_thresholdCallbackfn);

/* Enable Tx threshold level interrupt and install its callback function */

DrvI2S_EnableInt (I2S_TX_FIFO_THRESHOLD, Tx_thresholdCallbackfn);

DrvI2S_DisableInt

Prototype

int32_t DrvI2S_DisableInt (E_I2S_CALLBACK_TYPE Type);

Description

To disable I2S interrupt function and uninstall relative call back function in I2S interrupt handler.

Parameter

Type [in]

There are eight types for call back function.

I2S_RX_UNDERFLOW: Rx FIFO underflow

I2S_RX_OVERFLOW: Rx FIFO overflow

I2S_RX_FIFO_THRESHOLD: Data word in Rx FIFO is higher than Rx threshold level.

I2S_TX_UNDERFLOW: Tx FIFO underflow.

I2S_TX_OVERFLOW: Tx FIFO overflow

I2S_TX_FIFO_THRESHOLD: Data word in Tx FIFO is less than Tx threshold level.

I2S_TX_RIGHT_ZERO_CROSS: Tx right channel zero cross.

I2S_TX_LEFT_ZERO_CROSS: Tx left channel zero cross.

Include

Driver/DrvI2S.h

Return Value

0: Succeed

<0: Failed

Example

```
/* Disable Rx threshold level interrupt and uninstall its callback function */
DrvI2S_DisableInt (I2S_RX_FIFO_THRESHOLD);
/* Disable Tx threshold level interrupt and uninstall its callback function */
DrvI2S_DisableInt (I2S_TX_FIFO_THRESHOLD);
```

DrvI2S_GetBCLKFreq

Prototype

uint32_t DrvI2S_GetBCLKFreq (void);

Description

To get the I2S BCLK (Bit Clock) frequency.

BCLK = I2S source clock / (2 x (BCLK divider + 1))

Parameter

None

Include

Driver/DrvI2S.h

Return Value

I2S BCLK frequency. The unit is Hz.

Example

```
uint32_t u32clock;
u32clock = DrvI2S_GetBCLKFreq ( );      /* Get I2S BCLK clock frequency */
```

DrvI2S_SetBCLKFreq

Prototype

```
void DrvI2S_SetBCLKFreq (uint32_t u32Bclk);
```

Description

To configure BCLK (Bit Clock) clock. The BCLK will work when I2S operates in master mode. *BCLK = I2S source clock / (2 x BCLK divider + 1))*

Parameter

u32Bclk [in]

I2S BCLK frequency. The unit is Hz.

Include

Driver/DrvI2S.h

Return Value

None

Example

```
DrvI2S_SetBCLKFreq (512000);      /* Set I2S BCLK clock frequency 512 KHz */
```

DrvI2S_GetMCLKFreq

Prototype

```
uint32_t DrvI2S_GetMCLKFreq (void);
```

Description

To get the I2S MCLK (Master Clock) frequency.

MCLK = I2S source clock / (2 x MCLK divider))

Parameter

None

Include

Driver/DrvI2S.h

Return Value

I2S MCLK frequency. The unit is Hz.

Example

```
uint32_t u32clock;
u32clock = DrvI2S_GetMCLKFreq ( );      /* Get I2S MCLK clock frequency */
```

DrvI2S_SetMCLKFreq

Prototype

```
void DrvI2S_SetMCLKFreq (uint32_t u32Mclk);
```

Description

To configure MCLK (Master Clock) clock.

MCLK = I2S source clock / (2 x (MCLK divider))

Parameter

u32Mclk [in]

I2S MCLK frequency. The unit is Hz.

Include

Driver/DrvI2S.h

Return Value

None

Example

```
DrvI2S_SetMCLKFreq (12000000);      /* Set I2S MCLK clock frequency 12MHz */
```

DrvI2S_SetChannelZeroCrossDetect

Prototype

```
int32_t DrvI2S_SetChannelZeroCrossDetect (E_I2S_CHANNEL channel, int32_t i32flag);
```

Description

To enable or disable right/left channel zero cross detect function.

Parameter

channel [in]

I2S_LEFT_CHANNEL / I2S_RIGHT_CHANNEL

i32flag [in]

To enable or disable zero cross detect function. (1: enable 0: disable)

Include

Driver/DrvI2S.h

Return Value

0: Succeed

<0: Failed

Example

```
/* Enable left channel zero cross detect */
DrvI2S_SetChannelZeroCrossDetect (I2S_LEFT_CHANNEL, 1);
/* Disable right channel zero cross detect */
DrvI2S_SetChannelZeroCrossDetect (I2S_RIGHT_CHANNEL, 0);
```

DrvI2S_EnableTxDMA

Prototype

```
void DrvI2S_EnableTxDMA (void);
```

Description

To enable I2S Tx DMA function. I2S requests DMA to transfer data to Tx FIFO.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

```
/* Enable I2S Tx DMA function */
DrvI2S_EnableTxDMA ( );
```

DrvI2S_DisableTxDMA

Prototype

```
void DrvI2S_DisableTxDMA (void);
```

Description

To disable I2S Tx DMA function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Disable I2S Tx DMA function */

DrvI2S_DisableTxDMA ( );

DrvI2S_EnableRxDMA

Prototype

void DrvI2S_EnableRxDMA (void);

Description

To enable I2S Rx DMA function. I2S requests DMA to transfer data from Rx FIFO.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Enable I2S Rx DMA function */

DrvI2S_EnableRxDMA ( );

DrvI2S_DisableRxDMA

Prototype

void DrvI2S_DisableRxDMA (void);

Description

To disable I2S Rx DMA function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Disable I2S Rx DMA function */

DrvI2S_DisableRxDMA ( );

DrvI2S_EnableTx

Prototype

void DrvI2S_EnableTx (void);

Description

To enable I2S Tx function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Enable I2S Tx function */

DrvI2S_EnableTx ( );

DrvI2S_DisableTx

Prototype

void DrvI2S_DisableTx (void);

Description

To disable I2S Tx function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Disable I2S Tx function */

DrvI2S_DisableTx ( );

DrvI2S_EnableRx

Prototype

void DrvI2S_EnableRx (void);

Description

To enable I2S Rx function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Enable I2S Rx function */

DrvI2S_EnableRx ( );

DrvI2S_DisableRx

Prototype

void DrvI2S_DisableRx (void);

Description

To Disable I2S Rx function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Disable I2S Rx function */

DrvI2S_DisableRx ( );

DrvI2S_EnableTxMute

Prototype

void DrvI2S_EnableTxMute (void);

Description

To enable I2S Tx Mute function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

```
/* Enable I2S Tx Mute function */
DrvI2S_EnableTxMute ( );
```

DrvI2S_DisableTxMute

Prototype

void DrvI2S_DisableTxMute (void);

Description

To disable I2S Tx Mute function.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

```
/* Disable I2S Tx Mute function */
DrvI2S_DisableTxMute ( );
```

DrvI2S_EnableMCLK

Prototype

void DrvI2S_EnableMCLK (void);

Description

To enable I2S MCLK output from GPIOA Pin15.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Enable MCLK output */

DrvI2S_EnableMCLK ( );

DrvI2S_DisableMCLK

Prototype

void DrvI2S_DisableMCLK (void);

Description

To disable I2S MCLK output from GPIOA Pin15.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

/* Disable MCLK output */

DrvI2S_DisableMCLK ( );

DrvI2S_ClearTxFIFO

Prototype

void DrvI2S_ClearTxFIFO (void);

Description

To clear Tx FIFO. The internal pointer of Tx FIFO is reset to start point.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

DrvI2S_ClearTxFIFO ( );   /* Clear Tx FIFO */

DrvI2S_ClearRxFIFO

Prototype

void   DrvI2S_ClearRxFIFO (void);

Description

To clear Rx FIFO. The internal pointer of Rx FIFO is reset to start point.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

None

Example

DrvI2S_ClearRxFIFO ( );   /* Clear Rx FIFO */

DrvI2S_SelectClockSource

Prototype

void   DrvI2S_SelectClockSource (uint8_t u8ClkSrcSel);

Description

To select I2S clock source, including external 12M, PLL clock, HCLK and internal 22M.

Parameter

u8ClkSrcSel [in]

To select I2S clock source. There are four sources for I2S:

DRVI2S_EXT_12M: external 12MHz crystal clock

DRVI2S_PLL: PLL clock

DRVI2S_HCLK: HCLK.

DRVI2S_INTERNAL_22M: internal 22MHz oscillator clock

Include

Driver/DrvI2S.h

Return Value

None

Example

DrvI2S_SelClockSource (DRVI2S_EXT_12M); /* I2S clock source from external 12M */

DrvI2S_SelClockSource (DRVI2S_PLL); /* I2S clock source from PLL clock */

DrvI2S_SelClockSource (DRVI2S_HCLK); /* I2S clock source from HCLK */

DrvI2S_GetSourceClockFreq

Prototype

uint32_t DrvI2S_GetSourceClockFreq (void);

Description

To get I2S source clock frequency.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

I2S clock source frequency. The unit is Hz.

Example

uint32_t u32clock;
u32clock = DrvI2S_GetSourceClock ( );   /* Get I2S source clock frequency */

DrvI2S_GetVersion

Prototype

uint32_t DrvI2S_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvI2S.h

Return Value

Version number:

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

17. EBI Driver

17.1. EBI Introduction

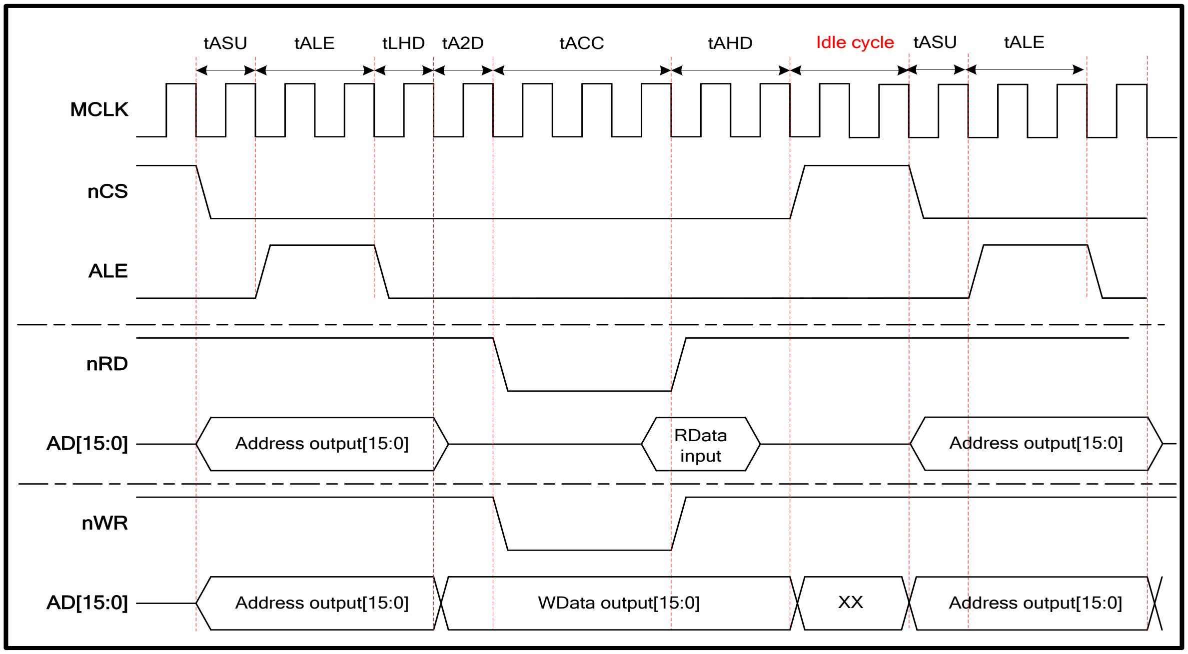
The NuMicro™ 100 series equips an external bus interface (EBI) for external device used.
To save the connections between external device and this chip, EBI support address bus and data bus multiplex mode. And, address latch enable (ALE) signal supported differentiate the address and data cycle.

Only NUC1x0xxxBx and NUC1x0xxxCx series support this function, ex:NUC140RD2BN and NUC140VE3CN. Please refer to NuMicro™ NUC100 Series Products Selection Guide of Appendix in details.

17.2. EBI Feature

- External devices with max. 64K byte (8 bit data width)/128K byte (16 bit data width) supported.
- Variable external bus base clock (MCLK) supported.
- 8 bit or 16 bit data width supported.
- Variable data access time (tACC), address latch enable time (tALE) and address hold time (tAHD) supported.
- Address bus and data bus multiplex mode supported to save the address pins.
- Configurable idle cycle supported for differene access condition: Write command finish (W2X), Read-to-Read (R2R).

- Relative timing control waveform shown as below,



17.3. Type Definition

E_DRVEBI_DATA_WIDTH

Enumeration Identifier	Value	Description
E_DRVEBI_DATA_8BIT	0x0	EBI data bus width is 8 bit
E_DRVEBI_DATA_16BIT	0x1	EBI data bus width is 16 bit

E_DRVEBI_ADDR_WIDTH

Enumeration Identifier	Value	Description
E_DRVEBI_ADDR_8BIT	0x0	EBI address bus width is 8 bit
E_DRVEBI_ADDR_16BIT	0x1	EBI address bus width is 16 bit

E_DRVEBI_MCLKDIV

Enumeration Identifier	Value	Description
E_DRVEBI_MCLKDIV_1	0x0	EBI output clock is HCLK/1
E_DRVEBI_MCLKDIV_2	0x1	EBI output clock is HCLK/2
E_DRVEBI_MCLKDIV_4	0x2	EBI output clock is HCLK/4

E_DRVEBI_MCLKDIV_8	0x3	EBI output clock is HCLK/8
E_DRVEBI_MCLKDIV_16	0x4	EBI output clock is HCLK/16
E_DRVEBI_MCLKDIV_32	0x5	EBI output clock is HCLK/32
E_DRVEBI_MCLKDIV_DEFAULT	0x6	EBI output clock is HCLK/1

17.4. API Functions

DrvEBI_Open

Prototype

int32_t DrvEBI_Open (DRVEBI_CONFIG_T sEBIConfig)

Description

Enable EBI function and configure the relative EBI Control Registers.

Parameter

sEBIConfig [in]

Input the general EBI Control Register settings

DRVEBI_CONFIG_T

eDataWidth:

E_DRVEBI_DATA_WIDTH, it could be E_DRVEBI_DATA_8BIT or E_DRVEBI_DATA_16BIT.

eAddrWidth:

E_DRVEBI_ADDR_WIDTH, it could be E_DRVEBI_ADDR_8BIT or E_DRVEBI_ADDR_16BIT.

u32BaseAddress:

If eAddrWidth is 8 bits: 0x60000000 <= u32BaseAddress <0x60010000
If eAddrWidth is 16 bits: 0x60000000 <= u32BaseAddress <0x60020000

u32Size:

If eAddrWidth is 8 bits: 0x0 < u32Size <= 0x10000
If eAddrWidth is 16 bits: 0x0 < u32Size <= 0x20000

Include

Driver/DrvEBI.h

Return Value

E_SUCCESS: Operation successful

E _DRVEBI_ERR_ARGUMENT: Invalid argument

Example

```
/* Open the EBI device with 16bit bus width. The start address of the device is at 0x60000000
and the storage size is 128KB */
DRVEBI_CONFIG_T sEBIConfig;
sEBIConfig.eDataWidth = eDRVEBI_DATA_16BIT;
sEBIConfig.eAddrWidth = eDRVEBI_ADDR_16BIT;
sEBIConfig.u32BaseAddress = 0x60000000;
sEBIConfig.u32Size = 0x20000;
DrvEBI_Open (sEBIConfig);
```

DrvEBI_Close

Prototype

void DrvEBI_Close (void)

Description

Disable EBI function and release the relative pins for GPIO used.

Parameter

None

Include

Driver/DrvEBI.h

Return Value

None

Example:

```
/* Close the EBI device */
DrvEBI_Close ();
```

DrvEBI_SetBusTiming

Prototype

void DrvEBI_SetBusTiming (DRVEBI_TIMING_T sEBITiming)

Description

Configure the relative EBI bus timing.

Parameter

sEBITiming [in]

DRVEBI_TIMING_T

eMCLKDIV:

E_DRVEBI_MCLKDIV, it could be E_DRVEBI_MCLKDIV_1, E_DRVEBI_MCLKDIV_2, E_DRVEBI_MCLKDIV_4,

E_DRVEBI_MCLKDIV_8, E_DRVEBI_MCLKDIV_16,
E_DRVEBI_MCLKDIV_32 or E_DRVEBI_MCLKDIV_DEFAULT.

u8ExttALE: Expand time of ALE 0~7, tALE = (u8ExttALE+1)*MCLK.

u8ExtIR2R: Idle cycle between Read-Read 0~15, idle cycle = u8ExtIR2R*MCLK

u8ExtIW2X: Idle cycle after Write 0~15, idle cycle = u8ExtIW2X*MCLK

u8ExttAHD: EBI address hold time 0~7, tAHD = (u8ExttAHD+1)*MCLK

u8ExttACC: EBI data access time 0~31, tAHD = (u8ExttACC+1)*MCLK

Include

Driver/DrvEBI.h

Return Value

None

Example:

```
/* Set the relative EBI bus timing */
DRVEBI_TIMING_T sEBITiming;
sEBITiming.eMCLKDIV = eDRVEBI_MCLKDIV_1;
sEBITiming.u8ExttALE = 0;
sEBITiming.u8ExtIR2R = 0;
sEBITiming.u8ExtIW2X = 0;
sEBITiming.u8ExttAHD = 0;
sEBITiming.u8ExttACC = 0;
DrvEBI_SetBusTiming (sEBITiming);
```

DrvEBI_GetBusTiming

Prototype

void DrvEBI_GetBusTiming (DRVEBI_TIMING_T *psEBITiming)

Description

Get the current bus timing of the EBI.

Parameter

psEBITiming [out]

DRVEBI_TIMING_T, refer to DrvEBI_SetBusTiming for detail information

Include

Driver/DrvEBI.h

Return Value

Data buffer pointer that stored the EBI bus timing settings

Example:

```
/* Get the current EBI bus timing */
DRVEBI_TIMING_T sEBITiming;
DrvEBI_GetBusTiming (&sEBITiming);
```

DrvEBI_GetVersion

Prototype

uint32_t DrvEBI_GetVersion (void)

Description

Get the version number of EBI driver.

Include

Driver/DrvEBI.h

Return Value

Version number:

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

Example

```
/* Get the current version of EBI Driver */
u32Version = DrvEBI_GetVersion ();
```

18. Appendix

18.1. NuMicro™ NUC100 Series Products Selection Guide

NUC100 Advance Line Selection Guide (low density)

Part number	Flash	SRAM	Connectivity			I2S	PWM	Comp.	ADC	Timer	RTC	EBI	ISP ICP	PDMA	I/O	Package
			UART	SPI	I2C											
NUC100LC1BN	32 KB	4 KB	2	1	2	1	4	1	8x12-Bit	4x32-bit	v	-	v	1	up to 35	LQFP48
NUC100LD1BN	64 KB	4 KB	2	1	2	1	4	1	8x12-Bit	4x32-bit	v	-	v	1	up to 35	LQFP48
NUC100LD2BN	64 KB	8 KB	2	1	2	1	4	1	8x12-bit	4x32-bit	v	-	v	1	up to 35	LQFP48
NUC100RC1BN	32 KB	4 KB	2	2	2	1	4	2	8x12-Bit	4x32-bit	v	v	v	1	up to 49	LQFP64
NUC100RD1BN	64 KB	4 KB	2	2	2	1	4	2	8x12-Bit	4x32-bit	v	v	v	1	up to 49	LQFP64
NUC100RD2BN	64 KB	8 KB	2	2	2	1	4	2	8x12-bit	4x32-bit	v	v	v	1	up to 49	LQFP64

NUC100 Advance Line Selection Guide (medium density)

Part number	Flash (KB)	SRAM	Connectivity			I2S	PWM	Comp.	ADC	Timer	RTC	EBI	ISP ICP	PDMA	I/O	Package
			UART	SPI	I2C											
NUC100LD3AN	64 KB	16 KB	2	1	2	1	6	1	8x12-bit	4x32-bit	v	-	v	9	up to 35	LQFP48
NUC100LE3AN	128 KB	16 KB	2	1	2	1	6	1	8x12-bit	4x32-bit	v	-	v	9	up to 35	LQFP48
NUC100RD3AN	64 KB	16 KB	3	2	2	1	6	2	8x12-bit	4x32-bit	v	-	v	9	up to 49	LQFP64
NUC100RE3AN	128 KB	16 KB	3	2	2	1	6	2	8x12-bit	4x32-bit	v	-	v	9	up to 49	LQFP64
NUC100VD2AN	64 KB	8 KB	3	4	2	1	8	2	8x12-bit	4x32-bit	v	-	v	9	up to 80	LQFP100
NUC100VD3AN	64 KB	16 KB	3	4	2	1	8	2	8x12-bit	4x32-bit	v	-	v	9	up to 80	LQFP100
NUC100VE3AN	128 KB	16 KB	3	4	2	1	8	2	8x12-bit	4x32-bit	v	-	v	9	up to 80	LQFP100

NUC120 USB Line Selection Guide (low density)

Part number	Flash	SRAM	Connectivity				I2S	PWM	Comp.	ADC	Timer	RTC	EBI	ISP ICP	PDMA	I/O	Package
			UART	SPI	I2C	USB											
NUC120LC1BN	32 KB	4 KB	2	1	2	1	1	4	1	8x12-Bit	4x32-bit	v	-	v	1	up to 31	LQFP48
NUC120LD1BN	64 KB	4 KB	2	1	2	1	1	4	1	8x12-Bit	4x32-bit	v	-	v	1	up to 31	LQFP48
NUC120LD2BN	64 KB	8 KB	2	1	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	1	up to 31	LQFP48

NUC120RC1BN	32 KB	4 KB	2	2	2	1	1	4	2	8x12-Bit	4x32-bit	v	v	v	1	up to 45	LQFP64
NUC120RD1BN	64 KB	4 KB	2	2	2	1	1	4	2	8x12-Bit	4x32-bit	v	v	v	1	up to 45	LQFP64
NUC120RD2BN	64 KB	8 KB	2	2	2	1	1	4	2	8x12-bit	4x32-bit	v	v	v	1	up to 45	LQFP64

NUC120 USB Line Selection Guide (medium density)

Part number	Flash	SRAM	Connectivity				I2S	PWM	Comp.	ADC	Timer	RTC	EBI	ISP ICP	PDMA	I/O	Package
			UART	SPI	I2C	USB											
NUC120LD3AN	64 KB	16 KB	2	1	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 31	LQFP48
NUC120LE3AN	128 KB	16 KB	2	1	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 31	LQFP48
NUC120RD3AN	64 KB	16 KB	2	2	2	1	1	6	2	8x12-bit	4x32-bit	v	-	v	9	up to 45	LQFP64
NUC120RE3AN	128 KB	16 KB	2	2	2	1	1	6	2	8x12-bit	4x32-bit	v	-	v	9	up to 45	LQFP64
NUC120VD2AN	64 KB	8 KB	3	4	2	1	1	8	2	8x12-bit	4x32-bit	v	-	v	9	up to 76	LQFP100
NUC120VD3AN	64 KB	16 KB	3	4	2	1	1	8	2	8x12-bit	4x32-bit	v	-	v	9	up to 76	LQFP100
NUC120VE3AN	128 KB	16 KB	3	4	2	1	1	8	2	8x12-bit	4x32-bit	v	-	v	9	up to 76	LQFP100

NUC130 Automotive Line Selection Guide

Part number	Flash	SRAM	Connectivity					I2S	PWM	Comp.	ADC	Timer	RTC	EBI	ISP ICP	PDMA	I/O	Package
			UART	SPI	I2C	LIN	CAN											
NUC130LC1CN	32 KB	4 KB	3	1	2	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 35	LQFP48
NUC130LD2CN	64 KB	8 KB	3	1	2	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 35	LQFP48
NUC130LE3CN	128KB	16KB	3	1	2	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 35	LQFP48
NUC130RC1CN	32 KB	4 KB	3	2	2	2	1	1	6	2	8x12-bit	4x32-bit	v	v	v	9	up to 49	LQFP64
NUC130RD2CN	64 KB	8 KB	3	2	2	2	1	1	6	2	8x12-bit	4x32-bit	v	v	v	9	up to 49	LQFP64
NUC130RE3CN	128KB	16KB	3	2	2	2	1	1	6	2	8x12-bit	4x32-bit	v	v	v	9	up to 49	LQFP64
NUC130VE3CN	128KB	16KB	3	4	2	2	1	1	8	2	8x12-bit	4x32-bit	v	v	v	9	up to 80	LQFP100

NUC140 Connectivity Line Selection Guide

Part number	Flash	SRAM	Connectivity						I2S	PWM	Comp.	ADC	Timer	RTC	EBI	ISP ICP	PDMA	I/O	Package
			UART	SPI	I2C	USB	LIN	CAN											
NUC140LC1CN	32 KB	4 KB	2	1	2	1	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 31	LQFP48
NUC140LD2CN	64 KB	8 KB	2	1	2	1	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 31	LQFP48
NUC140LE3CN	128KB	16KB	2	1	2	1	2	1	1	4	1	8x12-bit	4x32-bit	v	-	v	9	up to 31	LQFP48
NUC140RC1CN	32 KB	4 KB	3	2	2	1	2	1	1	4	2	8x12-bit	4x32-bit	v	v	v	9	up to 45	LQFP64
NUC140RD2CN	64 KB	8 KB	3	2	2	1	2	1	1	4	2	8x12-bit	4x32-bit	v	v	v	9	up to 45	LQFP64
NUC140RE3CN	128KB	16KB	3	2	2	1	2	1	1	4	2	8x12-bit	4x32-bit	v	v	v	9	up to 45	LQFP64
NUC140VE3CN	128KB	16KB	3	4	2	1	2	1	1	8	2	8x12-bit	4x32-bit	v	v	v	9	up to 76	LQFP100

NUC101 Selection Guide

Part number	Flash	SRAM	Connectivity						I2S	PWM	Comp.	ADC	Timer	RTC	ISP ICP	I/O	Package
			UART	SPI	I2C	USB	LIN	CAN									
NUC101LC1BN	32 KB	4 KB	1	2	1	1	-	-	1	4	1	-	4x32-bit	-	v	up to 31	LQFP48
NUC101LD2BN	64 KB	8 KB	1	2	1	1	-	-	1	4	1	-	4x32-bit	-	v	up to 31	LQFP48
NUC101YC1BN	32 KB	4 KB	1	2	1	1	-	-	1	1	1	-	4x32-bit	-	v	up to 31	QFP36
NUC101YD2BN	64 KB	8 KB	1	2	1	1	-	-	1	1	1	-	4x32-bit	-	v	up to 31	QFP36

18.2. PDID Table

NUC100 Advance Line PDID List (low density)

Part number	PDID
NUC100LC1BN	0x10010008
NUC100LD1BN	0x10010005
NUC100LD2BN	0x10010004
NUC100RC1BN	0x10010017
NUC100RD1BN	0x10010014
NUC100RD2BN	0x10010013

NUC100 Advance Line PDID List (medium density)

Part number	PDID
NUC100LD3AN	0x00010003
NUC100LE3AN	0x00010000
NUC100RD3AN	0x00010012
NUC100RE3AN	0x00010009
NUC100VD2AN	0x00010022
NUC100VD3AN	0x00010021
NUC100VE3AN	0x00010018

NUC120 USB Line PDID List (low density)

Part number	PDID
NUC120LC1BN	0x10012008
NUC120LD1BN	0x10012005
NUC120LD2BN	0x10012004
NUC120RC1BN	0x10012017
NUC120RD1BN	0x10012014
NUC120RD2BN	0x10012013

NUC120 USB Line PDID List (medium density)

Part number	PDID
NUC120LD3AN	0x00012003
NUC120LE3AN	0x00012000
NUC120RD3AN	0x00012012
NUC120RE3AN	0x00012009
NUC120VD2AN	0x00012022

NUC120VD3AN	0x00012021
NUC120VE3AN	0x00012018

NUC130 Automotive Line PDID List

Part number	PDID
NUC130LC1CN	0x20013008
NUC130LD2CN	0x20013004
NUC130LE3CN	0x20013000
NUC130RC1CN	0x20013017
NUC130RD2CN	0x20013013
NUC130RE3CN	0x20013009
NUC130VE3CN	0x20013018

NUC140 Connectivity Line PDID List

Part number	PDID
NUC140LC1CN	0x20014008
NUC140LD2CN	0x20014004
NUC140LE3CN	0x20014000
NUC140RC1CN	0x20014017
NUC140RD2CN	0x20014013
NUC140RE3CN	0x20014009
NUC140VE3CN	0x20014018

NUC101 PDID List

Part number	PDID
NUC101LC1BN	0x10010108
NUC101LD2BN	0x10010104
NUC101YC1BN	0x10010147
NUC101YD2BN	0x10010143

19. Revision History

Version	Date	Description
V1.00.001	Jan. 8, 2009	● Created
V1.00.002	July. 30, 2010	● Fix errors ● Add example of API
V1.03.001	Jan. 5, 2011	● Fix errors ● Fix clock diagram error ● Update API description according to NUC100 Series BSP v.1.003.001
V1.04.001	Mar. 19, 2011	● Update CAN driver ● Supports NUC130XXXCN and 140XXXCN
V1.04.002	Apr. 27, 2011	● Update clock diagram to add EBI clock tree
V1.04.003	Apr. 30, 2011	● Fix the deviation value of 10KHz and 22.1184MHz oscillator ● Fix the register name of CAN driver
V1.05.001	June. 27, 2011	● Rename API name of CAN driver ● Add some new API