

NuMicro™ M051 Series Driver Reference Guide

V1.00.001

Publication Release Date: Aug. 2010

Support Chips:

M051 Series

Support Platforms:

Nuvoton

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

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

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

Nuvoton Technology Corp. All rights reserved.

Table of Contents

1. Overview	11
1.1. Organization.....	11
1.2. Relative Documents	11
1.3. Abbreviations and Glossaries	11
1.4. Data Type Definition	12
2. SYS Driver	13
2.1. Introduction.....	13
2.2. Clock Diagram	14
2.3. Type Definition	15
E_SYS_IP_RST.....	15
E_SYS_IP_CLK	15
E_SYS_PLL_CLKSRC	16
E_SYS_IP_DIV	16
E_SYS_IP_CLKSRC	16
E_SYS_CHIP_CLKSRC	16
E_SYS_PD_TYPE	16
2.4. Functions.....	17
DrvSYS_ReadProductID	17
DrvSYS_GetResetSource	17
DrvSYS_ClearResetSource	18
DrvSYS_ResetIP	18
DrvSYS_ResetCPU	19
DrvSYS_ResetChip	19
DrvSYS_SelectBODVolt.....	20
DrvSYS_SetBODFunction	20
DrvSYS_EnableBODLowPowerMode.....	21
DrvSYS_DisableBODLowPowerMode.....	22
DrvSYS_EnableLowVoltReset.....	22
DrvSYS_DisableLowVoltReset	23
DrvSYS_GetBODState.....	23
DrvSYS_UnlockProtectedReg.....	24
DrvSYS_LockProtectedReg	24
DrvSYS_IsProtectedRegLocked	25
DrvSYS_EnablePOR	25
DrvSYS_DisablePOR	26
DrvSYS_SetRCAdjValue	26
DrvSYS_SetIPClock.....	27
DrvSYS_SelectHCLKSource	28

DrvSYS_SelectSysTickSource	28
DrvSYS_SelectIPClockSource	29
DrvSYS_SetClockDivider	30
DrvSYS_SetOscCtrl	31
DrvSYS_SetPowerDownWakeUpInt	31
DrvSYS_EnterPowerDown	32
DrvSYS_SelectPLLSource	33
DrvSYS_SetPLLMode	33
DrvSYS_GetExtClockFreq	34
DrvSYS_GetPLLContent	34
DrvSYS_SetPLLContent	35
DrvSYS_GetPLLClockFreq	36
DrvSYS_GetHCLKFreq	36
DrvSYS_Open	37
DrvSYS_SetFreqDividerOutput	37
DrvSYS_Delay	38
DrvSYS_GetChipClockSourceStatus	39
DrvSYS_GetClockSwitchStatus	39
DrvSYS_ClearClockSwitchStatus	40
DrvSYS_GetVersion	40
3. UART Driver	42
3.1. UART Introduction	42
3.2. UART Feature	42
3.3. Constant Definition	43
3.4. Type Definition	43
E_UART_PORT	43
E_INT_SOURCE	43
E_DATABITS_SETTINGS	43
E_PARITY_SETTINGS	43
E_STOPBITS_SETTINGS	44
E_FIFO_SETTINGS	44
E_UART_FUNC	44
E_MODE_RS485	44
3.5. Macros	45
_DRVUART_SENDBYTE	45
_DRVUART_RECEIVEBYTE	45
_DRVUART_SET_DIVIDER	45
_DRVUART_RECEIVEAVAILABLE	46
_DRVUART_WAIT_TX_EMPTY	46
3.6. Functions	47
DrvUART_Open	47
DrvUART_Close	49
DrvUART_EnableInt	49
DrvUART_DisableInt	50
DrvUART_ClearIntFlag	51
DrvUART_GetIntStatus	52
DrvUART_GetCTSInfo	53

DrvUART_SetRTS.....	54
DrvUART_Read.....	55
DrvUART_Write.....	56
DrvUART_EnablePDMA.....	57
DrvUART_DisablePDMA.....	57
DrvUART_SetFnIRDA.....	58
DrvUART_SetFnRS485.....	59
DrvUART_SetFnLIN.....	60
DrvUART_GetVersion.....	61

4. TIMER/WDT Driver62

4.1. TIMER/WDT Introduction.....	62
4.2. TIMER/WDT Feature.....	62
4.3. Type Definition.....	62
E_TIMER_CHANNEL.....	62
E_TIMER_OPMODE.....	63
E_WDT_CMD.....	63
E_WDT_INTERVAL.....	63
4.4. Functions.....	64
DrvTIMER_Init.....	64
DrvTIMER_Open.....	64
DrvTIMER_Close.....	65
DrvTIMER_SetTimerEvent.....	65
DrvTIMER_ClearTimerEvent.....	66
DrvTIMER_EnableInt.....	67
DrvTIMER_DisableInt.....	67
DrvTIMER_GetIntFlag.....	68
DrvTIMER_ClearIntFlag.....	68
DrvTIMER_Start.....	69
DrvTIMER_GetIntTicks.....	69
DrvTIMER_ResetIntTicks.....	70
DrvTIMER_Delay.....	70
DrvTIMER_SetEXTClockFreq.....	71
DrvTIMER_GetVersion.....	71
DrvWDT_Open.....	72
DrvWDT_Close.....	72
DrvWDT_InstallISR.....	73
DrvWDT_Iocctl.....	73

5. GPIO Driver75

5.1. GPIO introduction.....	75
5.2. GPIO Feature.....	75
5.3. Type Definition.....	75
E_DRVGPIO_PORT.....	75
E_DRVGPIO_PIN.....	75
E_DRVGPIO_EXT_INT_PIN.....	76

E_DRVGPIO_IO	76
E_DRVGPIO_INT_TYPE	76
E_DRVGPIO_INT_MODE	76
E_DRVGPIO_DBCLKSRC	76
E_DRVGPIO_FUNC	76
5.4. Macros	77
_PORT_DOUT	77
P0[n]_DOUT / P1[n]_DOUT / P2[n]_DOUT / P3[n]_DOUT / P4[n]_DOUT	78
5.5. Functions	78
DrvGPIO_Open	78
DrvGPIO_Close	79
DrvGPIO_SetBit	80
DrvGPIO_GetBit	80
DrvGPIO_ClrBit	81
DrvGPIO_SetPortBits	82
DrvGPIO_GetPortBits	82
DrvGPIO_GetDoutBit	83
DrvGPIO_GetPortDoutBits	83
DrvGPIO_SetBitMask	84
DrvGPIO_GetBitMask	84
DrvGPIO_ClrBitMask	85
DrvGPIO_SetPortMask	86
DrvGPIO_GetPortMask	86
DrvGPIO_ClrPortMask	87
DrvGPIO_EnableDebounce	87
DrvGPIO_DisableDebounce	88
DrvGPIO_SetDebounceTime	88
DrvGPIO_GetDebounceSampleCycle	89
DrvGPIO_EnableInt	90
DrvGPIO_DisableInt	91
DrvGPIO_SetIntCallback	91
DrvGPIO_EnableEINT	92
DrvGPIO_DisableEINT	93
DrvGPIO_GetIntStatus	93
DrvGPIO_InitFunction	94
DrvGPIO_GetVersion	95
6. DrvADC Introduction	96
6.1. ADC Introduction	96
6.2. ADC Feature	96
6.3. Type Definition	97
E_ADC_INPUT_MODE	97
E_ADC_OPERATION_MODE	97
E_ADC_CLK_SRC	97
E_ADC_EXT_TRI_COND	97
E_ADC_CH7_SRC	97
E_ADC_CMP_CONDITION	97
6.4. Macros	98

_DRVADC_CONV	98
_DRVADC_GET_FIFO_SIZE	98
_DRVADC_GET_ADC_INT_FLAG	99
_DRVADC_GET_CMP0_INT_FLAG	99
_DRVADC_GET_CMP1_INT_FLAG	99
_DRVADC_CLEAR_ADC_INT_FLAG	100
_DRVADC_CLEAR_CMP0_INT_FLAG	100
_DRVADC_CLEAR_CMP1_INT_FLAG	101
6.5. Functions.....	101
DrvADC_Open	101
DrvADC_Close	103
DrvADC_SetADCCChannel	103
DrvADC_ConfigADCCChannel7	104
DrvADC_SetADCInputMode	104
DrvADC_SetADCOperationMode	105
DrvADC_SetADCClkSrc	106
DrvADC_SetADCDivisor	106
DrvADC_EnableADCInt	107
DrvADC_DisableADCInt	108
DrvADC_EnableADCCmp0Int	108
DrvADC_DisableADCCmp0Int	109
DrvADC_EnableADCCmp1Int	109
DrvADC_DisableADCCmp1Int	110
DrvADC_GetConversionRate	111
DrvADC_EnableExtTrigger	111
DrvADC_DisableExtTrigger	112
DrvADC_StartConvert	113
DrvADC_StopConvert	113
DrvADC_IsConversionDone	113
DrvADC_GetConversionData	114
DrvADC_IsDataValid	115
DrvADC_IsDataOverrun	115
DrvADC_EnableADCCmp0	116
DrvADC_DisableADCCmp0	117
DrvADC_EnableADCCmp1	117
DrvADC_DisableADCCmp1	118
DrvADC_EnableSelfCalibration	119
DrvADC_IsCalibrationDone	119
DrvADC_DisableSelfCalibration	120
DrvADC_GetVersion	120
7. DrvSPI Introduction.....	122
7.1. SPI Introduction	122
7.2. General Feature	122
7.3. Constant Definition	123
E_DRVSPI_PORT	123
E_DRVSPI_MODE	123
E_DRVSPI_TRANS_TYPE	123
E_DRVSPI_ENDIAN	123
E_DRVSPI_BYTE_REORDER	123

E_DRVSPi_SSLTRIG	124
E_DRVSPi_SS_ACT_TYPE	124
7.4. Functions.....	125
DrvSPi_Open.....	125
DrvSPi_Close	126
DrvSPi_SetEndian	127
DrvSPi_SetBitLength	127
DrvSPi_SetByteReorder	128
DrvSPi_SetSuspendCycle	129
DrvSPi_SetTriggerMode	130
DrvSPi_SetSlaveSelectActiveLevel	131
DrvSPi_GetLevelTriggerStatus	132
DrvSPi_EnableAutoSS	132
DrvSPi_DisableAutoSS	133
DrvSPi_SetSS	134
DrvSPi_ClrSS	134
DrvSPi_IsBusy	135
DrvSPi_BurstTransfer	136
DrvSPi_SetClockFreq	137
DrvSPi_GetClock1Freq	138
DrvSPi_GetClock2Freq	138
DrvSPi_SetVariableClockFunction	139
DrvSPi_EnableInt	140
DrvSPi_DisableInt	141
DrvSPi_GetIntFlag	142
DrvSPi_ClrIntFlag	142
DrvSPi_SingleRead	143
DrvSPi_SingleWrite	144
DrvSPi_BurstRead	145
DrvSPi_BurstWrite	145
DrvSPi_DumpRxRegister	146
DrvSPi_SetTxRegister	147
DrvSPi_SetGo	148
DrvSPi_ClrGo	148
DrvSPi_GetVersion	149
8. I2C Driver.....	150
8.1. I2C Introduction	150
8.2. I2C Feature	150
8.3. Type Definition	150
E_I2C_CALLBACK_TYPE	150
8.4. Functions	151
DrvI2C_Open	151
DrvI2C_Close	151
DrvI2C_SetClockFreq	152
DrvI2C_GetClockFreq	152
DrvI2C_SetAddress	153
DrvI2C_SetAddressMask	153
DrvI2C_GetStatus	154

DrvI2C_WriteData	155
DrvI2C_ReadData	155
DrvI2C_Ctrl	155
DrvI2C_GetIntFlag	156
DrvI2C_ClearIntFlag	157
DrvI2C_EnableInt	157
DrvI2C_DisableInt	158
DrvI2C_InstallCallBack	158
DrvI2C_UninstallCallBack	159
DrvI2C_SetTimeoutCounter	160
DrvI2C_ClearTimeoutFlag	160
DrvI2C_GetVersion	161

9. DrvPWM Introduction.....162

9.1. PWM Introduction	162
9.2. PWM Features	162
9.3. Constant Definition	162
9.4. Functions	163
DrvPWM_IsTimerEnabled	163
DrvPWM_SetTimerCounter	164
DrvPWM_GetTimerCounter	165
DrvPWM_EnableInt	166
DrvPWM_DisableInt	167
DrvPWM_ClearInt	168
DrvPWM_GetIntFlag	169
DrvPWM_GetRisingCounter	170
DrvPWM_GetFallingCounter	171
DrvPWM_GetCaptureIntStatus	172
DrvPWM_ClearCaptureIntStatus	173
DrvPWM_Open	174
DrvPWM_Close	174
DrvPWM_EnableDeadZone	175
DrvPWM_Enable	176
DrvPWM_SetTimerClk	177
DrvPWM_SetTimerIO	180
DrvPWM_SelectClockSource	181
DrvPWM_GetVersion	182

10. FMC Driver184

10.1. FMC Introduction	184
10.2. FMC Feature	184
Memory Address Map	184
Flash Memory Structure	184
10.3. Type Definition	185
E_FMC_BOOTSELECT	185
10.4. Functions	185

DrvFMC_EnableISP	185
DrvFMC_DisableISP	186
DrvFMC_BootSelect	186
DrvFMC_GetBootSelect	187
DrvFMC_EnableLDUpdate	187
DrvFMC_DisableLDUpdate	188
DrvFMC_EnableConfigUpdate	188
DrvFMC_DisableConfigUpdate	189
DrvFMC_EnablePowerSaving	189
DrvFMC_DisablePowerSaving	190
DrvFMC_Write	190
DrvFMC_Read	191
DrvFMC_Erase	191
DrvFMC_WriteConfig	192
DrvFMC_ReadDataFlashBaseAddr	193
DrvFMC_EnableLowSpeedMode	193
DrvFMC_DisableLowSpeedMode	194
DrvFMC_GetVersion	194
11. EBI Driver	195
11.1. EBI Introduction	195
11.2. EBI Feature	195
11.3. Type Definition	196
E_DRVEBI_BUS_WIDTH	196
E_DRVEBI_MCLKDIV	196
11.4. API Functions	196
DrvEBI_Open	196
DrvEBI_Close	197
DrvEBI_SetBusTiming	197
DrvEBI_GetBusTiming	198
DrvEBI_GetVersion	199
12. Appendix	200
12.1. NuMicro™ M051 Series Selection Guide	200
12.2. PDIP Table	200
13. Revision History	201

1. Overview

1.1. Organization

This document describes the NuMicro™ M051 series driver reference manual. System-level software developers can use the NuMicro™ M051 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™ M051 series.

This document is organized into several chapters. Chapter 1 is an overview. From chapter 2 to chapter 11 are the detailed driver descriptions including the followings: System Driver, UART Driver, Timer Driver, GPIO Driver, ADC Driver, SPI Driver, I2C Driver, PWM Driver, FMC Driver and EBI Driver.

Finally, for the NuMicro™ M051 series selection guide and product identity list are described in Appendix.

1.2. Relative Documents

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

- NuMicro™ M051 series Technical Reference Manual (TRM)

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
EBI	External Bus Interface
FIFO	First-In-First-Out
FMC	Flash Memory Controller
GPIO	General Purpose Input/Output
I2C	Inter Integrated Circuit
LVR	Low Voltage Reset

PDID	Product Device Identify
PLL	Phase-Locked Loop
POR	Power On Reset
PWM	Pulse-Width Modulation
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter

1.4. Data Type Definition

The definition of all basic data types used in our drivers follow 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

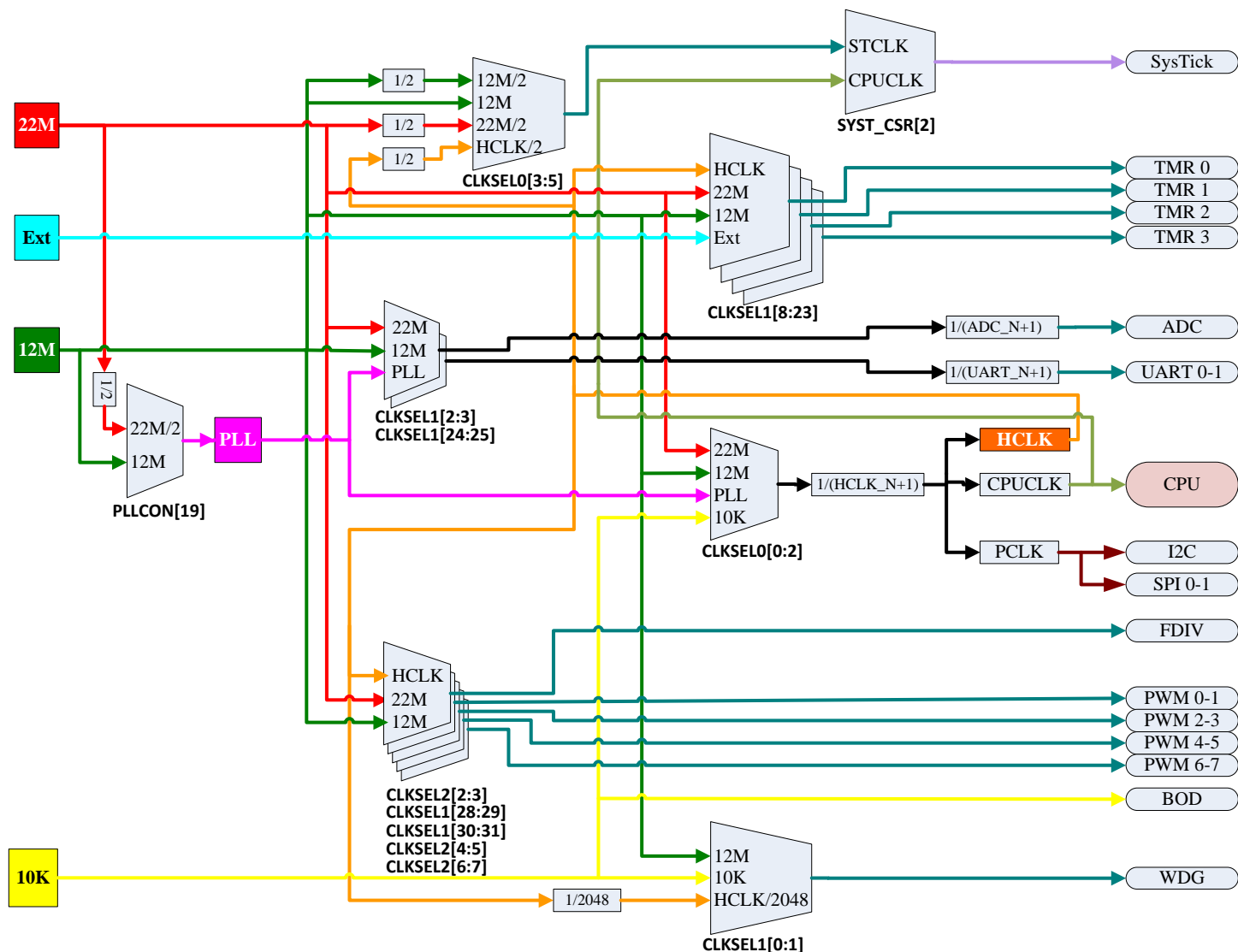
2.1. Introduction

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 22.1184 MHz with 1% deviation. 10K means internal 10 KHz RC clock source with 30% deviation.



2.3. Type Definition

E_SYS_IP_RST

Enumeration identifier	Value	Description
E_SYS_GPIO_RST	1	GPIO reset
E_SYS_TMR0_RST	2	Timer0 reset
E_SYS_TMR1_RST	3	Timer1 reset
E_SYS_TMR2_RST	4	Timer2 reset
E_SYS_TMR3_RST	5	Timer3 reset
E_SYS_I2C_RST	8	I2C reset
E_SYS_SPI0_RST	12	SPI0 reset
E_SYS_SPI1_RST	13	SPI1 reset
E_SYS_UART0_RST	16	UART0 reset
E_SYS_UART1_RST	17	UART1 reset
E_SYS_PWM03_RST	20	PWM0~3 reset
E_SYS_PWM47_RST	21	PWM4~7 reset
E_SYS_ADC_RST	28	ADC reset
E_SYS_EBI_RST	32	EBI reset

E_SYS_IP_CLK

Enumeration identifier	Value	Description
E_SYS_WDT_CLK	0	Watch Dog Timer 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_I2C_CLK	8	I2C clock enable control
E_SYS_SPI0_CLK	12	SPI0 clock enable control
E_SYS_SPI1_CLK	13	SPI1 clock enable control
E_SYS_UART0_CLK	16	UART0 clock enable control
E_SYS_UART1_CLK	17	UART1 clock enable control
E_SYS_PWM01_CLK	20	PWM01 clock enable control
E_SYS_PWM23_CLK	21	PWM23 clock enable control
E_SYS_PWM45_CLK	22	PWM45 clock enable control
E_SYS_PWM67_CLK	23	PWM67 clock enable control
E_SYS_ADC_CLK	28	ADC 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_HCLK_DIV	2	HCLK source clock divider setting

E_SYS_IP_CLKSRC

Enumeration identifier	Value	Description
E_SYS_WDT_CLKSRC	0	Watch Dog Timer clock source setting
E_SYS_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_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_OSC22M	1	Select Internal 22M Oscillator
E_SYS_OSC10K	2	Select Internal 10K Oscillator
E_SYS_PLL	3	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 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, I2C, SPI0, SPI1, UART0, UART1, PWM03, PWM47, ADC, and EBI.

Parameter

eIpRst [in]

Enumeration for IP reset, reference the [E_SYS_IP_RST](#) of Section 2.3.

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_ResetCPU

Prototype

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

Description

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

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.

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.

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 i32Enalbe, 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.

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.

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.

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.

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.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_GetBODState

Prototype

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

Description

To get Brown-out Detector state.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

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

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

Example

```
uint32_t u32flag;
```

```
/* Get Brown-out state if Brown-out detector function is enabled */
```

```
u32flag = DrvSYS_GetBODState ( );
```

DrvSYS_UnlockProtectedReg

Prototype

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

Description

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

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed

<0 Failed

Example

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

DrvSYS_LockProtectedReg

Prototype

```
int32_t DrvSYS_LockProtectedReg (void);
```

Description

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

Parameters

None

Include

Driver/DrvSYS.h

Return Value

0 Succeed
 <0 Failed

Example

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

DrvSYS_IsProtectedRegLocked
Prototype

```
int32_t    DrvSYS_IsProtectedRegLocked (void);
```

Description

To check the protected registers are locked or not.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

1: The Protected Registers are unlocked.
 0: The Protected Registers are locked.

Example

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

DrvSYS_EnablePOR
Prototype

```
void    DrvSYS_EnablePOR (void);
```

Description

To re-enable power-on-reset control.

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.

Parameters

None

Include

Driver/DrvSYS.h

Return Value

None

Example

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

DrvSYS_SetRCAdjValue

Prototype

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

Description

To set RC adjustment value to trim RC oscillator frequency. The greater RC adjustment value, the less RC output frequency. The adjustment step is about 1% and middle number is 0x20, i.e. 0%. Following table shows the relationship between adjustment steps and frequency deviation amount.

Adjustment Value	Deviation
0x0	~ -32%
...	
0x19	~ -1%
0x20	0%
0x21	~ 1%
...	...
0x3F	~ 31%

Parameter

u32Adj [in]

The RC adjustment value. The value is 0x00~0x3F.

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Set RC adjustment value 0x20 */
DrvSYS_SetRCAdjValue (0x20);
```

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, Timer0, Timer1, Timer2, Timer3, I2C, SPI0, SPI1, UART0, UART1, PWM01, PWM23, PWM45, PWM67, ADC, EBI, Flash ISP controller and Frequency Divider Output.

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, PLL clock, internal 10K oscillator clock, or internal 22M oscillator clock. Please refer to the [Clock Diagram](#) for HCLK usage in details.

Parameter

u8ClkSrcSel [in]

0: External 12M 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 Sys Tick clock source from external 12M crystal clock, external 12M crystal clock/2, HCLK/2, or internal 22M oscillator clock/2.

Parameter

u8ClkSrcSel [in]

- 0: External 12M clock
- 2: External 12M clock / 2
- 3: HCLK / 2
- 4~7: 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 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.

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

u8ClkSrcSel [in]

IP's corresponding clock source.

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 external trigger */
DrvSYS_SelectIPClockSource (E_SYS_TMR0_CLKSRC, 0x03);
/* Select I2S clock source from HLCK */
```

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_HCLK_DIV

i32value [in]

Divide number.

HCLK, 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 12M crystal.

Parameter

eOscCtrl [in]

E_SYS_XTL12M / 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 4096 clock cycle delay to wait the 12M crystal or 22M oscillator clock stable. The power down wake up interrupt will occur when GPIO, UART, WDT, or BOD wakeup.

Parameter

i32Enable [in]

1: enable, 0: disable

pdwucallbackFn [in]

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

i32enWUDelay [in]

1: enable the 4096 clock cycle delay, 0: disable the 4096 clock cycle delay

Include

Driver/DrvSYS.h

Return Value

None

Example

```
/* Enable Power down Wake up Interrupt function, install callback function
'PWRWU_CallbackFn', and enable the 4096 clock cycle 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.

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 ePlISrc, uint32_t u32PlIClk);

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_GetPLLClock \(\)](#) could be used to get actual PLL clock.

The [DrvSYS_GetHCLK \(\)](#) could be used to get actual HCLK clock.

The [DrvSYS_SetClockDivider \(\)](#) could be used to get lower HCLK clock.

Parameter

u32Hclk [in]

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

Include

Driver/DrvSYS.h

Return Value

0 Succeed
< 0 The clock setting is out of range

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

M051 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 M051 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_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, 12M crystal, or PLL clock.

Parameter

eOscCtrl [in]

E_SYS_XTL12M / 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.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

0 Clock switch success
1 Clock switch fail

Example

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

DrvSYS_ClearClockSwitchStatus

Prototype

void DrvSYS_ClearClockSwitchStatus (void);

Description

To clear the Clock Switch Fail Flag.

Parameter

None

Include

Driver/DrvSYS.h

Return Value

None

Example

```
uint32_t u32flag;
DrvSYS_SelectHCLKSource (0);   /* Change HCLK clock source to be external 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 Receive 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

Rx FIFO pointer value.

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).

u8RxTriggerLevel: 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 disable the THRE interrupt enable flag. */

DrvUART_DisableInt (UART_PORT0, DRVUART_THREINT);

DrvUART_GetIntStatus

Prototype

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

Description

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

Parameter

u32Port [in]

Specify UART_PORT0/UART_PORT1/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. Return 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.

Sample Code

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

Sample Code

```
/* 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.

Sample Code

```
/* 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.

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

- Independent clock source for each channel,
TMR0_CLK, TMR1_CLK, TMR2_CLK and TMR3_CLK.
- Internal 8-bit pre-scale counter.
- Internal 24-bit up counter is readable through TDR (Timer Data Register).
- Time out period =
(Period of timer clock input) * (8-bit pre-scale counter + 1) * (24-bit TCMP).
- Maximum counting cycle time = $(1 / 25 \text{ MHz}) * (2^8) * (2^{24} - 1)$, if TMR_CLK = 25 MHz.
- 18-bit free running counter to avoid CPU from WDT reset before the delay time expires.
- Selectable time-out interval ($2^4 \sim 2^{18}$) and the time out interval is 86.67us ~ 21.93ms,
if WDT_CLK = 12MHz).
- Reset period = $(1/12\text{MHz}) * 63$, if WDT_CLK = 12MHz.

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_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

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: No interrupt 1: 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

uint32_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_SetEXTClockFreq

Prototype

void DrvTIMER_SetEXTClockFreq (uint32_t u32ClockValue)

Description

Set the external clock frequency, it's used for timer clock source.
User can change the timer clock source from the external clock source by calling [DrvSYS_SelectIPClockSource \(...\)](#).

Parameter

u32ClockFreq [in]

Set the clock frequency (Hz) for external clock source

Include

Driver/DrvTIMER.h

Return Value

None

Example:

```
/* Set external clock value is 32 KHz */
DrvTIMER_SetEXTClockFreq (32000);
```

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

```
void DrvWDT_Open (E_WDT_INTERVAL WDTlevel)
```

Description

Enable WDT engine clock and set WDT time-out interval.

Parameter

WDTlevel [in]

E_WDT_INTERVAL, enumerate the WDT time-out interval.
Refer to [WDT_INTERVAL enumeration](#) for detail time-out value.

Include

Driver/DrvTIMER.h

Return Value

None

Example:

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

DrvWDT_Close

Prototype

```
void DrvWDT_Close (void)
```

Description

The function is used to stop/disable WDT relative functions.

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.

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.

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

There are up to 40 General Purpose I/O pins shared with special feature functions in M051 series. The 40 pins are arranged in 5 ports named with P0, P1, P2, P3 and P4. Each port equips maximum 8 pins.

5.2. GPIO Feature

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

5.3. Type Definition

E_DRVGPIO_PORT

Enumeration Identifier	Value	Description
E_PORT0	0	Define GPIO Port 0
E_PORT1	1	Define GPIO Port 1
E_PORT2	2	Define GPIO Port 2
E_PORT3	3	Define GPIO Port 3
E_PORT4	4	Define GPIO Port 4

E_DRVGPIO_PIN

Enumeration Identifier	Value	Description
E_PIN0	0	Define the GPIO pin 0
E_PIN1	1	Define the GPIO pin 1
E_PIN2	2	Define the GPIO pin 2
E_PIN3	3	Define the GPIO pin 3
E_PIN4	4	Define the GPIO pin 4
E_PIN5	5	Define the GPIO pin 5
E_PIN6	6	Define the GPIO pin 6

E_PIN7	7	Define the GPIO pin 7
--------	---	-----------------------

E_DRVGPIO_EXT_INT_PIN

Enumeration Identifier	Value	Description
E_EINT0_PIN	2	Define the external interrupt pin 2 for GPIO Port 3
E_EINT1_PIN	3	Define the external interrupt pin 3 for GPIO Port 3

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	P3.6	Enable frequency output function
E_FUNC_I2C	P3.4	Enable I2C function
E_FUNC_SPI0 / E_FUNC_SPI1	P1.4 / P0.4	Enable SPI0/SPI1 function
E_FUNC_ADC0 / E_FUNC_ADC1 / E_FUNC_ADC2 / E_FUNC_ADC3 / E_FUNC_ADC4 / E_FUNC_ADC5 / E_FUNC_ADC6 / E_FUNC_ADC7	P1.0 / P1.1 / P1.2 / P1.3 / P1.4 / P1.5 / P1.6 / P1.7	Enable ADC0/ADC1/ADC2/ADC3/ ADC4/ADC5/ADC6/ADC7 function

E_FUNC_EXTINT0 / E_FUNC_EXTINT1	P3.2 / P3.3	Enable External INT0/INT1 function
E_FUNC_TMR0 / E_FUNC_TMR1 / E_FUNC_TMR2 / E_FUNC_TMR3	P3.4 / P3.5 / P1.0 / P1.1	Enable TIMER0/TIMER1/TIMER2/ TIMER3 as Toggle Out mode
E_FUNC_UART0 / E_FUNC_UART1	P3.0&P3.1&P0.2&P0.3 / P1.2&P1.3&P0.0&P0.1	Enable UART0/UART function
E_FUNC_PWM01 / E_FUNC_PWM23 / E_FUNC_PWM45 / E_FUNC_PWM67	P2.0&P2.1&P4.0&P4.1 / P2.2&P2.3&P4.2&P4.3 / P2.4&P2.5 / P2.6&P2.7 /	Enable PWM01/PWM23/PWM45/ PWM67 function
E_FUNC_EBI_8B	P0.0 ~ P0.7 / P3.3 & P3.6 & P3.7 / P4.4 & P4.5	Enable EBI with 8 bit bus width
E_FUNC_EBI_16B	P2.0~P2.7 / P0.0 ~ P0.7 / P3.3 & P3.6 & P3.7 / P4.4 & P4.5	Enable EBI with 16 bit bus width
E_FUNC_ICE	P4.6 & P4.7	Configure pins for ICE CLK and DAT

5.4. Macros

_PORT_DOUT

Prototype

_PORT_DOUT (PortNum, PinNum)

Description

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

Parameter

PortNum [in]

Specify the GPIO port. It could be 0~4 to correspond to the Port0/1/2/3/4.

PinNum [in]

Specify pin of the GPIO port. It could be 0~7.

Include

Driver/DrvGPIO.h

Example:

```
/* Configure Port0-1 to output mode */
DrvGPIO_Open (E_PORT0, E_PIN1, E_IO_OUTPUT);
/* Set Port0-1 to high */
_PORT_DOUT (0, 1) = 1;
/* ..... */
```

```

/* Configure Port1-3 to input mode */
uint8_t u8PinValue;
DrvGPIO_Open (E_PORT1, E_PIN3, E_IO_INPUT);
/* Get Port1-3 pin value */
u8PinValue = _PORT_DOUT (1, 3);

```

P0[n]_DOUT / P1[n]_DOUT / P2[n]_DOUT / P3[n]_DOUT / P4[n]_DOUT

Prototype

```

P00_DOUT~P07_DOUT / P10_DOUT~P17_DOUT / P20_DOUT~P27_DOUT /
P30_DOUT~P37_DOUT / P40_DOUT~P47_DOUT

```

Description

These macros are the same as _PORT_DOUT macro but without any parameters. User can use the macro define directly like P00_DOUT to output data to the specified pin, or get pin value from this specified pin.

Parameter

None

Include

Driver/DrvGPIO.h

Example:

```

/* Configure Port0-1 to output mode */
DrvGPIO_Open (E_PORT0, E_PIN1, E_IO_OUTPUT);
/* Set Port0-1 to high */
_PORT_DOUT (0, 1) = 1;
/* ..... */
/* Configure Port1-3 to input mode */
uint8_t u8PinValue;
DrvGPIO_Open (E_PORT1, E_PIN3, E_IO_INPUT);
/* Get Port1-3 pin value */
u8PinValue = _PORT_DOUT (1, 3);

```

5.5. Functions

DrvGPIO_Open

Prototype

```

void DrvGPIO_Open (
    E_DRVGPIOPORT    port,
    E_DRVGPIOPIN     pin,
    E_DRVGPIOMODE     IOMode
)

```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

IOMode [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

None

Example:

```
/* Configure Port0-0 to GPIO output mode and Port0-1 to GPIO input mode*/
DrvGPIO_Open (E_PORT0, E_PIN0, E_IO_OUTPUT);
DrvGPIO_Open (E_PORT0, E_PIN1, E_IO_INPUT);
```

DrvGPIO_Close

Prototype

```
void DrvGPIO_Close (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)
```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

None

```
/* Close Port0-0 function and set to default quasi-bidirectional mode */
DrvGPIO_Close (E_PORT0, E_PIN0);
```

```
/* Configure Port0-0 as GPIO output mode*/
DrvGPIO_Open (E_PORT0, E_PIN0, E_IO_OUTPUT);
/* Set Port0-0 to 1(high) */
DrvGPIO_SetBit (E_PORT0, E_PIN0);
```

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

The specified input pin value: 0 / 1

Example:

```
int32_t i32BitValue;
/* Configure Port0-1 as GPIO input mode*/
DrvGPIO_Open (E_PORT0, E_PIN1, E_IO_INPUT);
i32BitValue = DrvGPIO_GetBit (E_PORT0, E_PIN1);
if (u32BitValue == 1)
{
    printf("Port0-1 pin status is high.\n");
}
else
{
    printf("Port0-1 pin status is low.\n");
}
```

DrvGPIO_ClrBit

Prototype

int32_t DrvGPIO_ClrBit (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)

Description

Set the specified GPIO pin to 0.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port.

It could be E_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example:

```
/* Configure Port0-0 as GPIO output mode*/
DrvGPIO_Open (E_PORT0, E_PIN0, E_IO_OUTPUT);
/* Set Port0-0 to 0(low) */
DrvGPIO_ClrBit (E_PORT0, E_PIN0);
```

DrvGPIO SetPortBits

Prototype

```
int32_tDrvGPIO_SetPortBits (E_DRVGPIO_PORT port, int32_t i32PortValue)
```

Description

Set the output port value to the specified GPIO port.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port.

It could be E_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

i32PortValue [in]

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

Include

Driver/DrvGPIO.h

Return Value

```
E_SUCCESS:           Operation successful
```

Example:

```
/* Set the output value of GPIO Port0 to 0x12 */
```

```
DrvGPIO_SetPortBits (E_PORT0, 0x12);
```

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 PORT0, E PORT1, E PORT2, E PORT3 and E PORT4.

Include

Driver/DrvGPIO.h

Return Value

The specified input port value: 0 ~ 0xFF

Example:

```
/* Get the GPIO Port0 input data value */
```

```
int32_t i32PortValue;
i32PortValue = DrvGPIO_GetPortBits (E_PORT0);
```

DrvGPIO_GetDoutBit

Prototype

```
int32_t DrvGPIO_GetDoutBit (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)
```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified register: 0 / 1

Example:

```
/* Get the Port0-1 data output value */
int32_t i32BitValue;
i32BitValue = DrvGPIO_GetDoutBit (E_PORT0, E_PIN1);
```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

Include

Driver/DrvGPIO.h

Return Value

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

Example:

```
/* Get the GPIO Port0 data output value */
int32_t i32PortValue;
i32PortValue = DrvGPIO_GetPortDoutBits (E_PORT0);
```

DrvGPIO_SetBitMask

Prototype

int32_t DrvGPIO_SetBitMask (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

Example:

```
/* Protect Port0-0 write data function */
DrvGPIO_SetBitMask (E_PORT0, E_PIN0);
```

DrvGPIO_GetBitMask

Prototype

int32_t DrvGPIO_GetBitMask (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)

Description

Get the bit value from the specified Data Output Write Mask Register.
If the bit value is 1, it's meaning 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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

The bit value of the specified register: 0 / 1

Example:

```
/* Get the bit value from GPIO Port0 Data Output Write Mask Resister */
int32_t i32MaskValue;
i32MaskValue = DrvGPIO_GetBittMask (E_PORT0, E_PIN0);
/* If (i32MaskValue = 1), its meaning Port0-0 is write protected */
```

DrvGPIO_ClrBitMask

Prototype

int32_t DrvGPIO_ClrBitMask (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)

Description

This function is used to remove the write protect function of the the corresponding GPIO pin. 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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS: Operation successful

```
/* Remove the Port0-0 write protect function */
DrvGPIO_ClrBitMask (E_PORT0, E_PORT0);
```

```
/* Protect Port0-0 and Port0-4 write data function */
DrvGPIO_SetPortMask (E_PORT0, 0x11);
```

Include

Driver/DrvGPIO.h

Return Value

The port value of the specified register: 0 ~ 0xFF

Example:

```

/* Get the port value from GPIO Port0 Data Output Write Mask Register */
int32_t i32MaskValue;
i32MaskValue = DrvGPIO_GetPortMask (E_PORT0);
/* If (i32MaskValue = 0x11), its meaning Port0-0 and Port0-4 are protected */

```

DrvGPIO **ClrPortMask**

Prototype

```
int32_t DrvGPIO_ClrPortMask (E_DRVGPIO_PORT port, int32_t i32PortValue)
```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

i32PortValue [in]

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

Include

Driver/DrvGPIO.h

Return Value

```
E_SUCCESS:                               Operation successful
```

Example:

```
/* Remove the Port0-0 and Port0-4 write protect function */
DrvGPIO_ClrPortMask (E_PORT0, 0x11);
```

DrvGPIO EnableDebounce

Prototype

```
int32_t DrvGPIO_EnableDebounce (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)
```

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 PORT0, E PORT1, E PORT2, E PORT3 and E PORT4.

pin [**in**]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS:

Operation successful

Example:

```
/* Enable Port0-0 interrupt de-bounce function */
```

DrvGPIO_EnableDebounce (E_PORT0, 0);

DrvGPIO DisableDebounce

Prototype

```
int32_t DrvGPIO_DisableDebounce (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)
```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.

Include

Driver/DrvGPIO.h

Return Value

E SUCCESS:

Operation successful

Example:

```
/* Disable Port0-0 interrupt de-bounce function */
```

```
DrvGPIO_DisableDebounce (E_PORT0, 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 $(2^4) * (1 / (10 * 1000)) \text{ s} = 16 * 0.0001 \text{ s} = 1600 \text{ us}$, 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 $(2^{(u32CycleSelection)}) * (\text{ClockSource}) \text{ second}$.

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,
    E_DRVGPIO_PIN        pin,
    E_DRVGPIO_INT_TYPE   Type,
    E_DRVGPIO_INT_MODE   Mode
)
```

Description

Enable the interrupt function of the specified GPIO pin.
Except for Port3-2 and Port3-3 pins are for external interrupt used.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port.
It could be E_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.
But the Port3-2 and Port3-3 are only used for external interrupt 0/1.

Type [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 Port1-3 interrupt function and its rising and edge trigger. */
DrvGPIO_EnableInt (E_PORT1, E_PIN3, E_IO_RISING, E_MODE_EDGE);
```

DrvGPIO_DisableInt

Prototype

```
int32_t DrvGPIO_DisableInt (E_DRVGPIO_PORT port, E_DRVGPIO_PIN pin)
```

Description

Disable the interrupt function of the specified GPIO pin.
Except for Port3-2 and Port3-3 pins are for external interrupt used.

Parameter

port [in]

E_DRVGPIO_PORT, specify GPIO port.
It could be E_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4.

pin [in]

Specify pin of the GPIO port. It could be E_PIN0, E_PIN2 ... ~ E_PIN7.
But the Port3-2 and Port3-3 are only used for external interrupt 0/1.

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS:	Operation successful
------------	----------------------

Example:

```
/* Disable Port1-3 interrupt function. */
DrvGPIO_DisableInt (E_PORT1, E_PIN3);
```

DrvGPIO_SetIntCallback

Prototype

```
void DrvGPIO_SetIntCallback (
    P0P1_CALLBACK      pfP0P1Callback,
    P2P3P4_CALLBACK    pfP2P3P4Callback
)
```

Description

Install the interrupt callback function for GPIO Port0/1 and Port2/3/4.

Parameter

pfP0P1Callback [in], the function pointer of GPIO Port0/1 callback function.
pfP2P3P4Callback [in], the function pointer of GPIO Port2/3/4 callback function.

Include

Driver/DrvGPIO.h

Return Value

None

Example:

```
/* Set GPIO Port0/1 and Port2/3/4 interrupt callback functions */
DrvGPIO_SetIntCallback (P0P1Callback, P2P3P4Callback);
```

DrvGPIO_EnableEINT

Prototype

```
Int32_t DrvGPIO_EnableEINT (
    E_DRVGPIO_EXT_INT_PIN    pin,
    E_DRVGPIO_INT_TYPE       Type,
    E_DRVGPIO_INT_MODE       Mode,
    EINT_CALLBACK             pfEINTCallback
)
```

Description

Enable the interrupt function for external GPIO interrupt from Port3-2 or Port3-3.

Parameter

pin [in]

Specify the external INT pin of GPIO Port3.
 It could be E_EINT0_PIN (Port3-2) or E_EINT1_PIN (Port3-3).

Type [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

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

Driver/DrvGPIO.h

E SUCCESS: Operation successful

```
/* Enable external INT0 interrupt as both-edge trigger. */
DrvGPIO_EnableEINT (E_EINT0_PIN, E_IO_BOTH_EDGE,
                    E_MODE_EDGE, EINT0Callback);
```

Int32_t DrvGPIO_DisableEINT (E_DRVGPIO_EXT_INT_PIN pin)

Disable the interrupt function for external GPIO interrupt from Port3-2 or Port3-3.

Specify the external INT pin of GPIO Port3.
It could be E_EINT0_PIN (Port3-2) or E_EINT1_PIN (Port3-3).

Driver/DrvGPIO.h

```
E_SUCCESS:                                Operation successful
```

```
/* Disable external INT0 interrupt function. */
DrvGPIO_DisableEINT (E_EINT0_PIN);
```

```
int32_t DrvGPIO_GetIntStatus (E_DRVGPIO_PORT port)
```

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_PORT0, E_PORT1, E_PORT2, E_PORT3 and E_PORT4..

Include

Driver/DrvGPIO.h

Return Value

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

Example:

```
/* Get GPIO Port0 interrupt status. */
int32_t i32INTStatus;
i32INTStatus = DrvGPIO_GetIntStatus (E_PORT0);
```

DrvGPIO_InitFunction
Prototype

int32_t DrvGPIO_InitFunction (E_DRVGPIO_FUNC function)

Description

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

Parameter
function [in]

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

E_FUNC_GPIO,
E_FUNC_CLKO,
E_FUNC_I2C,
E_FUNC_SPI0 / E_FUNC_SPI1,
E_FUNC_ADC0 / E_FUNC_ADC1 / E_FUNC_ADC2 /
E_FUNC_ADC3 / E_FUNC_ADC4 / E_FUNC_ADC5 /
E_FUNC_ADC6 / E_FUNC_ADC7,
E_FUNC_EXTINT0 / E_FUNC_EXTINT1,
E_FUNC_TMR0 / E_FUNC_TMR1 / E_FUNC_TMR2 /
E_FUNC_TMR3,
E_FUNC_UART0 / E_FUNC_UART1,
E_FUNC_PWM01 / E_FUNC_PWM23 / E_FUNC_PWM45 /
E_FUNC_PWM67,
E_FUNC_EBI_8B / E_FUNC_EBI_16B,
E_FUNC_ICE

Include

Driver/DrvGPIO.h

Return Value

E_SUCCESS:	Operation successful
E_DRVGPIO_ARGUMENT:	Incorrect argument

Example:

```
/* Init UART0 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. DrvADC Introduction

6.1. ADC Introduction

M051 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 four operation modes: single, burst, single-cycle scan and continuous scan mode. The A/D converters can be started by software and external STADC/P3.2 pin. In this document, we will introduce how to use the ADC driver.

Note that the used analog input pins must be configured as input type before starting the A/D conversion.

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
 1. Single mode
 2. Burst mode
 3. Single-cycle scan mode
 4. Continuous scan mode
- An A/D conversion can be started by
 1. Software write 1 to ADST bit
 2. External pin STADC
- Conversion result can be compared with specify value and 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 support 2 input sources: external analog voltage and internal fixed bandgap voltage.
- 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_BURST_OP	1	Burst mode
ADC_SINGLE_CYCLE_OP	2	Single cycle scan mode
ADC_CONTINUOUS_OP	3	Continuous scan mode

E_ADC_CLK_SRC

Enumeration Identifier	Value	Description
EXTERNAL_12MHZ	0	External 12MHz clock
INTERNAL_PLL	1	Internal PLL clock
INTERNAL_RC22MHZ	2	Internal 22MHz 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

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

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_FIFO_SIZE`

Prototype

```
void _DRVADC_GET_FIFO_SIZE (void);
```

Description

Get the number of available conversion results that stored in FIFO (First-In-First-Out) buffers of burst mode. User can get the conversion results by reading the data register of channel 0, no matter what channel was selected.

Include

Driver/DrvADC.h

Return Value

The number of available conversion results.

Example

```
/* Get the available conversion results that stored in FIFO buffers */
u8DataCount = _DRVADC_GET_FIFO_SIZE();
for(i=0; i< u8DataCount; i++)
    printf("%d\n", DrvADC_GetConversionData(0));
```

_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 complete 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, burst, single cycle scan or continuous scan mode.

ADC_SINGLE_OP : single mode

ADC_BURST_OP : burst mode

ADC_SINGLE_CYCLE_OP : single cycle scan mode

ADC_CONTINUOUS_OP : continuous scan mode

u8ChannelSelBitwise [in]

Specify the input channels. The channel 0 will be enabled automatically if this setting value is 0. If software enables more than one channel in single mode, only the lowest channel 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 one of the two corresponding channels needs to be selected. The conversion result will be placed to the corresponding data register of the selected channel. For example, in single-end input mode, 0x8 means the channel 3 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_RC22MHZ : internal 22MHz 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 analog input channels. The channel 0 will be enabled automatically if this setting value is 0. If software enables more than one channel in single mode, only the lowest channel will be converted and the other enabled channels will be ignored. In differential input mode, only one of the two corresponding channels needs to be selected. The conversion result will be placed to the corresponding data register of the selected channel. For example, in single-end input mode, 0x8 means the channel 3 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 (ADC_CH7_SRC Ch7Src);

Description

Select the input signal source of channel 7.

Parameters

Ch7Src [in]

Specify the analog input source.

EXTERNAL_INPUT_SIGNAL : external analog input

INTERNAL_BANDGAP : internal bandgap voltage

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_RC22MHZ : internal 22MHz 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.

1. The A/D conversion of the specified channel is completed in single mode.
2. 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.

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 */
u32AdcData = DrvADC_GetConversionData(3);
```

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 ~ 0xFFFF.

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 ~ 0xFFFF.

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_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. DrvSPI Introduction

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. M051 series contain two sets of SPI controller performing a serial-to-parallel conversion on data received from a peripheral device, and a parallel-to-serial conversion on data transmitted to a peripheral device. 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. Configure the DIVIDER register can program the frequency of serial clock output when it is as the master. If the VARCLK_EN bit in SPI_CNTRL[23] is enabled, the serial clock can be set as two programmable frequencies which are defined in DIVIDER and DIVIDER2. The format of the variable frequency is defined in VARCLK.

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

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

7.2. General Feature

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

7.3. Constant Definition

E_DRVSPI_PORT

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

E_DRVSPI_MODE

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

E_DRVSPI_TRANS_TYPE

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

E_DRVSPI_ENDIAN

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

E_DRVSPI_BYTE_REORDER

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

E_DRVSPI_SSLTRIG

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

E_DRVSPI_SS_ACT_TYPE

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

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

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.

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.

eDRVSPI_TYPE4: the clock idle state is high; drive data at rising-edge of serial clock; latch data at rising-edge of serial clock.

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.

i32BitLength [in]

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

Include

Driver/DrvSPI.h

Return Value

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

Example

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

DrvSPI_Close

Prototype

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

Description

Close the specified SPI module and disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Close SPI0 */
```

```
DrvSPI_Close(eDRVSPI_PORT0);
```

DrvSPI_SetEndian

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eEndian [in]

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

Include

Driver/DrvSPI.h

Return Value

None

Example

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

```
DrvSPI_SetEndian(eDRVSPI_PORT0, eDRVSPI_LSB_FIRST);
```

DrvSPI_SetBitLength

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

i32BitLength [in]

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

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.

E_DRVSPI_ERR_BIT_LENGTH : The bit length is out of range.

Example

```
/* The transfer bit length of SPI0 is 8-bit */
```

```
DrvSPI_SetBitLength(eDRVSPI_PORT0, 8);
```

DrvSPI_SetByteReorder

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

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 functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eDRVSPI_PORT2 : SPI2

eDRVSPI_PORT3 : SPI3

i32Interval [in]

In 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. It could be 2~17.

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

eSSTriggerMode [in]

Specify the trigger mode.

eDRVSPI_EDGE_TRIGGER: edge trigger.

eDRVSPI_LEVEL_TRIGGER: level trigger.

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_SetSlaveSelectActiveLevel

Prototype

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

Description

Set the active level of slave select.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

eSSActType [in]

Select the active type of slave select pin.

eDRVSPI_ACTIVE_LOW_FALLING:

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

eDRVSPI_ACTIVE_HIGH_RISING:

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

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_GetLevelTriggerStatus

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

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

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

Example

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

DrvSPI_EnableAutoSS

Prototype

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

Description

This function is used to enable the automatic slave select function and select the slave select pin. 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

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable the automatic slave select function */
DrvSPI_EnableAutoSS(eDRVSPI_PORT0);
```

DrvSPI_DisableAutoSS

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_SetSS
Prototype

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

Description

Configure the slave select pin. When the automatic slave select function is enabled, call this function to select the slave select pin. The state of slave select pin will be controlled by hardware. When the automatic slave select function is disabled, the slave select pin will be set to active state. In slave mode, executing this function is functionless.

Parameters
eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_ClrSS
Prototype

```
void DrvSPI_ClrSS(
    E_DRVSPI_PORT eSpiPort,
```

);

Description

When the automatic slave select function is enabled, call this function to deselect the slave select pin. The slave select pin will be set to inactive state. When the automatic slave select function is disabled, the slave select pin will be set to inactive state. In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

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

/* Set the slave select pin to inactive state */
DrvSPI_ClrSS(eDRVSPI_PORT0);
```

DrvSPI_IsBusy

Prototype

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

Description

Check the busy status of the specified SPI port.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

TURE: The SPI port is in busy.

FALSE: The SPI port is not in busy.

Example

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

DrvSPI_BurstTransfer

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

i32BurstCnt [in]

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

i32Interval [in]

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

Include

Driver/DrvSPI.h

Return Value

E_SUCCESS : Success.
 E_DRVSPi_ERR_BURST_CNT : The burst count is out of range.
 E_DRVSPi_ERR_SUSPEND_INTERVAL : The interval is out of range.

Example

/* Configure the SPi0 burst transfer mode; two transactions in one transfer; 10 delay clocks between the transactions. */

DrvSPi_BurstTransfer(eDRVSPi_PORT0, 2, 10);

DrvSPi_SetClockFreq

Prototype

```
uint32_t
DrvSPi_SetClockFreq(
    E_DRVSPi_PORT eSpiPort,
    uint32_t u32Clock1,
    uint32_t u32Clock2
);
```

Description

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

Parameters

eSpiPort [in]

Specify the SPi port.

eDRVSPi_PORT0 : SPi0

eDRVSPi_PORT1 : SPi1

u32Clock1 [in]

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

u32Clock2 [in]

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

Include

Driver/DrvSPi.h

Driver/DrvSYS.h

Return Value

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

Example

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

DrvSPI_GetClock1Freq

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

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

Example

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

DrvSPI_GetClock2Freq

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Driver/DrvSYS.h

Return Value

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

Example

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

DrvSPI_SetVariableClockFunction

Prototype

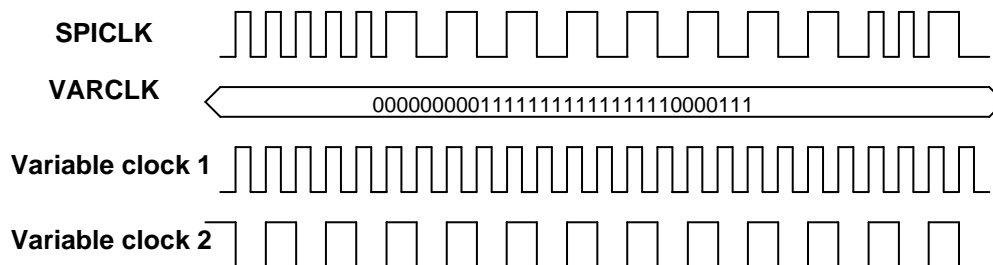
```
void
DrvSPI_SetVariableClockFunction (
    E_DRVSPI_PORT eSpiPort,
    uint8_t bEnable,
    uint32_t u32Pattern
);
```

Description

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

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

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



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

In slave mode, executing this function is functionless.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

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

pfnCallback [in]

The callback function of the corresponding SPI interrupt.

u32UserData [in]

The parameter which will be passed to the callback function.

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Enable the SPI0 interrupt and install the callback function. The parameter 0 will be passed
to the callback function. */
```

```
DrvSPI_EnableInt(eDRVSPI_PORT0, SPI0_Callback, 0);
```

DrvSPI_DisableInt

Prototype

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

Description

Disable the SPI interrupt.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

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

DrvSPI_GetIntFlag

Prototype

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

Description

Get the SPI interrupt flag.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

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
```

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(eDRV_SPI_PORT0, &u32DestinationData);
```

DrvSPI_SingleWrite
Prototype

```
uint8_t DrvSPI_SingleWrite (
    E_DRV_SPI_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.

eDRV_SPI_PORT0 : SPI0

eDRV_SPI_PORT1 : SPI1

pu32Data [in]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Data has been transferred.

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

Example

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



```
uint32_t u32SourceData;
DrvSPI_SingleWrite(eDRVSPI_PORT0, &u32SourceData);
```

DrvSPI_BurstRead

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [out]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf is valid.

FALSE: The data stored in pu32Buf is invalid.

Example

```
/* Read two words of data from SPI0 RX registers to au32DestinationData[u32DataCount]
and au32DestinationData[u32DataCount+1]. And then trigger SPI for next transfer. */
DrvSPI_BurstRead(eDRVSPI_PORT0, &au32DestinationData[u32DataCount]);
```

DrvSPI_BurstWrite

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [in]

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

Include

Driver/DrvSPI.h

Return Value

TRUE: The data stored in pu32Buf has been transferred.

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

Example

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

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

DrvSPI_DumpRxRegister

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPI_PORT0 : SPI0

eDRVSPI_PORT1 : SPI1

pu32Buf [out]

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

u32DataCount [in]

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

Include

Driver/DrvSPI.h

Return Value

The count of data actually read from RX registers.

Example

```
/* Read one word of data from SPI0 RX buffer and store to
au32DestinationData[u32DataCount] */
DrvSPI_DumpRxRegister(eDRVSPi_PORT0, &au32DestinationData[u32DataCount], 1);
```

DrvSPI_SetTxRegister
Prototype

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

Description

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

Parameters
eSpiPort [in]

Specify the SPI port.

eDRVSPi_PORT0 : SPI0

eDRVSPi_PORT1 : SPI1

pu32Buf [in]

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

u32DataCount [in]

The count of data written to TX registers.

Include

Driver/DrvSPI.h

Return Value

The count of data actually written to TX registers.

Example

```
/* Write one word of data stored in u32Buffer to SPI0 TX register. */
DrvSPI_SetTxRegister(eDRVSPi_PORT0, &u32Buffer, 1);
```

DrvSPI_SetGo

Prototype

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

Description

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

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPi_PORT0 : SPI0

eDRVSPi_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Trigger a SPI data transfer */
DrvSPI_SetGo(eDRVSPi_PORT0);
```

DrvSPI_ClrGo

Prototype

```
void DrvSPI_ClrGo (
    E_DRVSPi_PORT eSpiPort
);
```

Description

Stop a SPI data transfer.

Parameters

eSpiPort [in]

Specify the SPI port.

eDRVSPi_PORT0 : SPI0

eDRVSPi_PORT1 : SPI1

Include

Driver/DrvSPI.h

Return Value

None

Example

```
/* Stop a SPI data transfer */
DrvSPI_ClrGo(eDRV_SPI_PORT0);
```

DrvSPI_GetVersion

Prototype

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

Description

Get the version number of M051 series 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™ M051 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_CALLBACK_TYPE

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

8.4. Functions

DrvI2C_Open

Prototype

```
int32_t DrvI2C_Open (uint32_t u32BusClock);
```

Description

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

Parameter

u32BusClock [in]

To configure I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

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

DrvI2C_Close

Prototype

```
int32_t DrvI2C_Close (void);
```

Description

To close the I2C hardware.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

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

DrvI2C_SetClockFreq

Prototype

```
int32_t   DrvI2C_SetClockFreq (uint32_t u32BusClock);
```

Description

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

Parameter

u32BusClock [in]

To configure I2C bus clock. The unit is Hz.

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

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

DrvI2C_GetClockFreq

Prototype

```
uint32_t   DrvI2C_GetClockFreq (void);
```

Description

To get the I2C bus clock. $I2C\ bus\ clock = I2C\ source\ clock / (4 \times (I2CCLK_DIV+1))$

Parameter

None

Include

Driver/DrvI2C.h

Return Value

I2C bus clock

Example

```
uint32_t   u32clock;
```



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

DrvI2C_SetAddress

Prototype

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

Description

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

Parameter

slaveNo [in]

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

slave_addr [in]

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

GC_Flag [in]

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed

<0 Failed

Example

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

DrvI2C_SetAddressMask

Prototype

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

Description

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

Parameter

slaveNo [in]

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

slaveAddrMask [in]

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed
<0 Failed

Example

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

DrvI2C_GetStatus

Prototype

uint32_t DrvI2C_GetStatus (void);

Description

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

Parameter

None

Include

Driver/DrvI2C.h

Return Value

I2C status code

Example

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

DrvI2C_WriteData

Prototype

```
void DrvI2C_WriteData (uint8_t u8data);
```

Description

To set a byte of data to be sent.

Parameter

u8data [in]

Byte data.

Include

Driver/DrvI2C.h

Return Value

None

Example

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

DrvI2C_ReadData

Prototype

```
uint8_t DrvI2C_ReadData (void);
```

Description

To read the last data from I2C bus.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Last byte data

Example

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

DrvI2C_Ctrl

Prototype

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

Description

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

Parameter

start [in]

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

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

stop [in]

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

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

intFlag [in]

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

ack [in]

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

Include

Driver/DrvI2C.h

Return Value

None

Example

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

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

DrvI2C_GetIntFlag

Prototype

```
uint8_t DrvI2C_GetIntFlag (void);
```

Description

To get I2C interrupt flag status.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Interrupt status (1 or 0)

Example

```
uint8_t  u8flagStatus;

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

DrvI2C_ClearIntFlag

Prototype

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

Description

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

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

Example

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

DrvI2C_EnableInt

Prototype

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

Description

To enable I2C interrupt function.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
DrvI2C_EnableInt ( ); /* Enable I2C interrupt */
```

DrvI2C_DisableInt

Prototype

```
int32_t DrvI2C_DisableInt (void);
```

Description

To disable I2C interrupt function.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

```
DrvI2C_DisableInt ( ); /* Disable I2C interrupt */
```

DrvI2C_InstallCallBack

Prototype

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

Description

To install I2C call back function in I2C interrupt handler.

Parameter

Type [in]

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

I2CFUNC: For normal I2C condition

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

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

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

callbackfn [in]

Call back function name for specified interrupt event.

Include

Driver/DrvI2C.h

Return Value

0 Succeed
<0 Failed

Example

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

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

DrvI2C_UninstallCallBack

Prototype

```
int32_t    DrvI2C_UninstallCallBack (E_I2C_CALLBACK_TYPE Type);
```

Description

To uninstall I2C call back function in I2C interrupt handler.

Parameter

Type [in]

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

I2CFUNC: For normal I2C condition

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

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

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed
<0 Failed

Example

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

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

DrvI2C_SetTimeoutCounter

Prototype

```
int32_t   DrvI2C_SetTimeoutCounter (int32_t i32enable, uint8_t u8div4);
```

Description

To configure 14-bit time-out counter.

Parameter

i32enable [in]

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

u8div4 [in]

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

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

Include

Driver/DrvI2C.h

Return Value

0 Succeed

Example

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

DrvI2C_ClearTimeoutFlag

Prototype

```
void   DrvI2C_ClearTimeoutFlag (void);
```

Description

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

Parameter

None

Include

Driver/DrvI2C.h

Return Value

None

Example

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

DrvI2C_GetVersion

Prototype

uint32_t DrvI2C_GetVersion (void);

Description

Get this module's version.

Parameter

None

Include

Driver/DrvI2C.h

Return Value

Version number:

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

9. DrvPWM Introduction

9.1. PWM Introduction

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

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

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

9.2. PWM Features

The PWM controller includes following features:

- Up to two PWM group (PWMA/PWMB). Please refer to [NuMicro™ M051 Series 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 pre-scaler, 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.

9.3. Constant Definition

Name	Value	Description
------	-------	-------------

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_AUTO_RELOAD_MODE	1	PWM Timer auto-reload mode
DRVPWM_ONE_SHOT_MODE	0	PWM Timer One-shot mode

9.4. Functions

DrvPWM_IsTimerEnabled

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer.

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

DRV_PWM_TIMER4: PWM timer 4.

DRV_PWM_TIMER5: PWM timer 5.

DRV_PWM_TIMER6: PWM timer 6.

DRV_PWM_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 (DRV_PWM_TIMER3)==1)
printf("PWM timer 3 is enabled!\n");
else if(DrvPWM_IsTimerEnabled (DRV_PWM_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.

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

DRV_PWM_TIMER4: PWM timer 4.

DRV_PWM_TIMER5: PWM timer 5.

DRV_PWM_TIMER6: PWM timer 6.

DRV_PWM_TIMER7: PWM timer 7.

u16Counter [in]

Specify the timer value. (0~65535)

Include

Driver/DrvPWM.h

Return Value

None

Note

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

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 (DRV_PWM_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.

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

DRV_PWM_TIMER4: PWM timer 4.

DRV_PWM_TIMER5: PWM timer 5.

DRV_PWM_TIMER6: PWM timer 6.

DRV_PWM_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 (DRV_PWM_TIMER5);
```

DrvPWM_EnableInt

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV_PWM_TIMER0: PWM timer 0.

DRV_PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

DRV_PWM_TIMER4: PWM timer 4.

DRV_PWM_TIMER5: PWM timer 5.

DRV_PWM_TIMER6: PWM timer 6.

DRV_PWM_TIMER7: PWM timer 7.

or the capture.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

DRV PWM_CAP4: PWM capture 4.

DRV PWM_CAP5: PWM capture 5.

DRV PWM_CAP6: PWM capture 6.

DRV PWM_CAP7: PWM capture 7.

u8Int [in]

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

DRV PWM_CAP_RISING_INT: The capture rising interrupt.

DRV PWM_CAP_FALLING_INT: The capture falling interrupt.

DRV PWM_CAP_ALL_INT: All capture interrupt.

pfncallback [in]

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

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Enable PWM capture 5 falling edge interrupt and install DRV PWM_CapIRQHandler() as
it's interrupt callback function.*/
```

```
DrvPWM_EnableInt (DRV PWM_CAP5, DRV PWM_CAP_FALLING_INT,
DRV PWM_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

DRV PWM_TIMER0: PWM timer 0.

DRV PWM_TIMER1: PWM timer 1.

DRV PWM_TIMER2: PWM timer 2.

DRV PWM_TIMER3: PWM timer 3.

DRV PWM_TIMER4: PWM timer 4.

DRV PWM_TIMER5: PWM timer 5.
 DRV PWM_TIMER6: PWM timer 6.
 DRV PWM_TIMER7: PWM timer 7.

or the capture.

DRV PWM_CAP0: PWM capture 0.
 DRV PWM_CAP1: PWM capture 1.
 DRV PWM_CAP2: PWM capture 2.
 DRV PWM_CAP3: PWM capture 3.
 DRV PWM_CAP4: PWM capture 4.
 DRV PWM_CAP5: PWM capture 5.
 DRV PWM_CAP6: PWM capture 6.
 DRV PWM_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 (DRV PWM_CAP5);

/* Disable PWM timer 5 interrupt and uninstall PWM timer 5 callback function.*/
DrvPWM_DisableInt (DRV PWM_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

DRV PWM_TIMER0: PWM timer 0.
 DRV PWM_TIMER1: PWM timer 1.
 DRV PWM_TIMER2: PWM timer 2.

DRV PWM_TIMER3: PWM timer 3.

DRV PWM_TIMER4: PWM timer 4.

DRV PWM_TIMER5: PWM timer 5.

DRV PWM_TIMER6: PWM timer 6.

DRV PWM_TIMER7: PWM timer 7.

or the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

DRV PWM_CAP4: PWM capture 4.

DRV PWM_CAP5: PWM capture 5.

DRV PWM_CAP6: PWM capture 6.

DRV PWM_CAP7: PWM capture 7.

Include

Driver/DrvPWM.h

Return Value

None

Example

```
/* Clear PWM timer 1 interrupt flag.*/
DrvPWM_ClearInt (DRV PWM_TIMER1);
/* Clear PWM capture 0 interrupt flag. */
DrvPWM_ClearInt (DRV PWM_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

DRV PWM_TIMER0: PWM timer 0.

DRV PWM_TIMER1: PWM timer 1.

DRV_PWM_TIMER2: PWM timer 2.

DRV_PWM_TIMER3: PWM timer 3.

DRV_PWM_TIMER4: PWM timer 4.

DRV_PWM_TIMER5: PWM timer 5.

DRV_PWM_TIMER6: PWM timer 6.

DRV_PWM_TIMER7: PWM timer 7.

or the capture.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

DRV_PWM_CAP4: PWM capture 4.

DRV_PWM_CAP5: PWM capture 5.

DRV_PWM_CAP6: PWM capture 6.

DRV_PWM_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(DRV_PWM_TIMER6)==1)
printf("PWM timer 6 interrupt occurs!\n");
else if(DrvPWM_GetIntFlag(DRV_PWM_TIMER6)==0)
printf("PWM timer 6 interrupt doesn'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.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

DRV_PWM_CAP4: PWM capture 4.

DRV_PWM_CAP5: PWM capture 5.

DRV_PWM_CAP6: PWM capture 6.

DRV_PWM_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 (DRV_PWM_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.

DRV_PWM_CAP0: PWM capture 0.

DRV_PWM_CAP1: PWM capture 1.

DRV_PWM_CAP2: PWM capture 2.

DRV_PWM_CAP3: PWM capture 3.

DRV_PWM_CAP4: PWM capture 4.

DRV_PWM_CAP5: PWM capture 5.

DRV_PWM_CAP6: PWM capture 6.

DRV PWM_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 (DRV PWM_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.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

DRV PWM_CAP4: PWM capture 4.

DRV PWM_CAP5: PWM capture 5.

DRV PWM_CAP6: PWM capture 6.

DRV PWM_CAP7: PWM capture 7.

u8IntType [in]

Specify the Capture Latched Indicator.

DRV PWM_CAP_RISING_FLAG: The capture rising indicator flag.

DRV PWM_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 timer0~7 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) / Disable 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 peroid: Total Pulse peroid = 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.

DRV PWM_TIMER5: PWM timer 5.

DRV PWM_TIMER6: PWM timer 6.

DRV PWM_TIMER7: PWM timer 7.

or the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

DRV PWM_CAP4: PWM capture 4.

DRV PWM_CAP5: PWM capture 5.

DRV PWM_CAP6: PWM capture 6.

DRV PWM_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 (DRV PWM_TIMER0, 1);
/* Enable PWM capture 1 function.*/
DrvPWM_Enable (DRV PWM_CAP1, 1);
```

DrvPWM_SetTimerClk

Prototype

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

Description

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

Parameter

u8Timer [in]

Specify the timer

DRV PWM_TIMER0: PWM timer 0.

DRV PWM_TIMER1: PWM timer 1.

DRV PWM_TIMER2: PWM timer 2.

DRV PWM_TIMER3: PWM timer 3.

DRV PWM_TIMER4: PWM timer 4.

DRV PWM_TIMER5: PWM timer 5.

DRV PWM_TIMER6: PWM timer 6.

DRV PWM_TIMER7: PWM timer 7.

or the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

DRV PWM_CAP4: PWM capture 4.

DRV PWM_CAP5: PWM capture 5.

DRV PWM_CAP6: PWM capture 6.

DRV PWM_CAP7: PWM capture 7.

***sPt [in]**

It includes the following parameter

u32Frequency: The timer/capture frequency (Hz)

u8HighPulseRatio: High pulse ratio (1~100)

u8Mode: DRV PWM_ONE_SHOT_MODE / DRV PWM_AUTO_RELOAD_MODE

bInverter: Inverter Enable (1) / Inverter Disable (0)

u8ClockSelector: Clock Selector

DRV PWM_CLOCK_DIV_1: PWM input clock is divided by 1

DRV PWM_CLOCK_DIV_2: PWM input clock is divided by 2

DRV PWM_CLOCK_DIV_4: PWM input clock is divided by 4

DRV PWM_CLOCK_DIV_8: PWM input clock is divided by 8

DRV PWM_CLOCK_DIV_16: PWM input clock is divided by 16

(The parameter takes effect when u8Frequency = 0)

u8PreScale: Prescale (1 ~ 255).

The PWM input clock = PWM source clock / (u8PreScale + 1)

(The parameter takes effect when u8Frequency = 0)

Note: If the u8PreScale is set to 0, the timer will stop.

u32Duty: Pulse duty (0x1 ~ 0x10000)

(The parameter takes effect when u8Frequency = 0 or u8Timer = DRV PWM_CAP0/DRV PWM_CAP1/DRV PWM_CAP2/DRV PWM_CAP3/DRV PWM_CAP4/DRV PWM_CAP5/DRV PWM_CAP6/DRV PWM_CAP7)

Include

Driver/DrvPWM.h

Return Value

The actual specified PWM frequency (Hz).

Note

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

Sample code

```
/* PWM timer 0 output 1 KHz 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 1 KHz 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.

$$\begin{aligned}\text{Output frequency} &= \text{HCLK freq} / \text{sPt.u8ClockSelector} / \text{sPt.u8PreScale} / \text{sPt.u32Duty} \\ &= 22\text{MHz} / 1 / 22 / 1000 = 1\text{KHz}\end{aligned}$$

```
/* PWM Timer property */
sPt.u8Mode = DRVPWM_AUTO_RELOAD_MODE;
sPt.u8HighPulseRatio = 20; /* High Pulse period : Total Pulse period = 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.

DRV PWM_TIMER1: PWM timer 1.

DRV PWM_TIMER2: PWM timer 2.

DRV PWM_TIMER3: PWM timer 3.

DRV PWM_TIMER4: PWM timer 4.

DRV PWM_TIMER5: PWM timer 5.

DRV PWM_TIMER6: PWM timer 6.

DRV PWM_TIMER7: PWM timer 7.

or the capture.

DRV PWM_CAP0: PWM capture 0.

DRV PWM_CAP1: PWM capture 1.

DRV PWM_CAP2: PWM capture 2.

DRV PWM_CAP3: PWM capture 3.

DRV PWM_CAP4: PWM capture 4.

DRV PWM_CAP5: PWM capture 5.

DRV PWM_CAP6: PWM capture 6.

DRV PWM_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 (DRV PWM_TIMER0, 1);
/* Disable PWM timer 0 output. */
DrvPWM_SetTimerIO (DRV PWM_TIMER0, 0);
/* Enable PWM capture 3 input. */
DrvPWM_SetTimerIO (DRV PWM_CAP3, 1);
/* Disable PWM capture timer 3 input
DrvPWM_SetTimerIO (DRV PWM_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.

Parameter

u8Timer [in]

Specify the timer

DRV PWM_TIMER0 or DRV PWM_TIMER1: PWM timer 0 & PWM timer 1.

DRV PWM_TIMER2 or DRV PWM_TIMER3: PWM timer 2 & PWM timer 3.

DRV PWM_TIMER4 or DRV PWM_TIMER5: PWM timer 4 & PWM timer 5.

DRV PWM_TIMER6 or DRV PWM_TIMER7: PWM timer 6 & PWM timer 7.

u8ClockSourceSelector [in]

DRV PWM_EXT_12M / DRV PWM_HCLK / DRV PWM_INTERNAL_22M

DRV PWM_EXT_12M: external 12 MHz crystal clock

DRV PWM_HCLK: HCLK

DRV PWM_INTERNAL_22M: internal 22 MHz crystal clock

Include

Driver/DrvPWM.h

Return Value

None

Note

1. PWM timer 0 and PWM timer 1 use the same engine clock source. If user changed 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.
2. PWM timer 2 and PWM timer 3 use the same engine clock source.
3. PWM timer 4 and PWM timer 5 use the same engine clock source.
4. PWM timer 6 and PWM timer 7 use the same engine clock source.

Example

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

```
DrvPWM_SelectClockSource (DRV PWM_TIMER0, DRV PWM_HCLK);
```

Select PWM timer 6 and PWM timer 7 engine clock source from external 12MHz.

```
DrvPWM_SelectClockSource (DRV PWM_TIMER7, DRV PWM_EXT_12M);
```

DrvPWM_GetVersion

Prototype

```
uint32_t DrvPWM_GetVersion (void);
```

Description

Get this module's version.

Parameter

None

Include

Driver/DrvPWM.h

Return Value

Version number:

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

Example

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

int32_t i32PWMVersionNum ;

i32PWMVersionNum = DrvPWM_GetVersion();
```

10. FMC Driver

10.1. FMC Introduction

NuMicro™ M051 series equips with 64/32/16/8k bytes on chip embedded flash for application program memory (APROM), 4k bytes for ISP loader program memory (LDROM), and user configuration (Config0). User configuration block provides several bytes to control system logic, like flash security lock, boot select, brown out voltage level, and so on. NuMicro™ M051 series also provide additional 4k bytes data flash for user to store some application depended data before chip power off.

10.2. FMC Feature

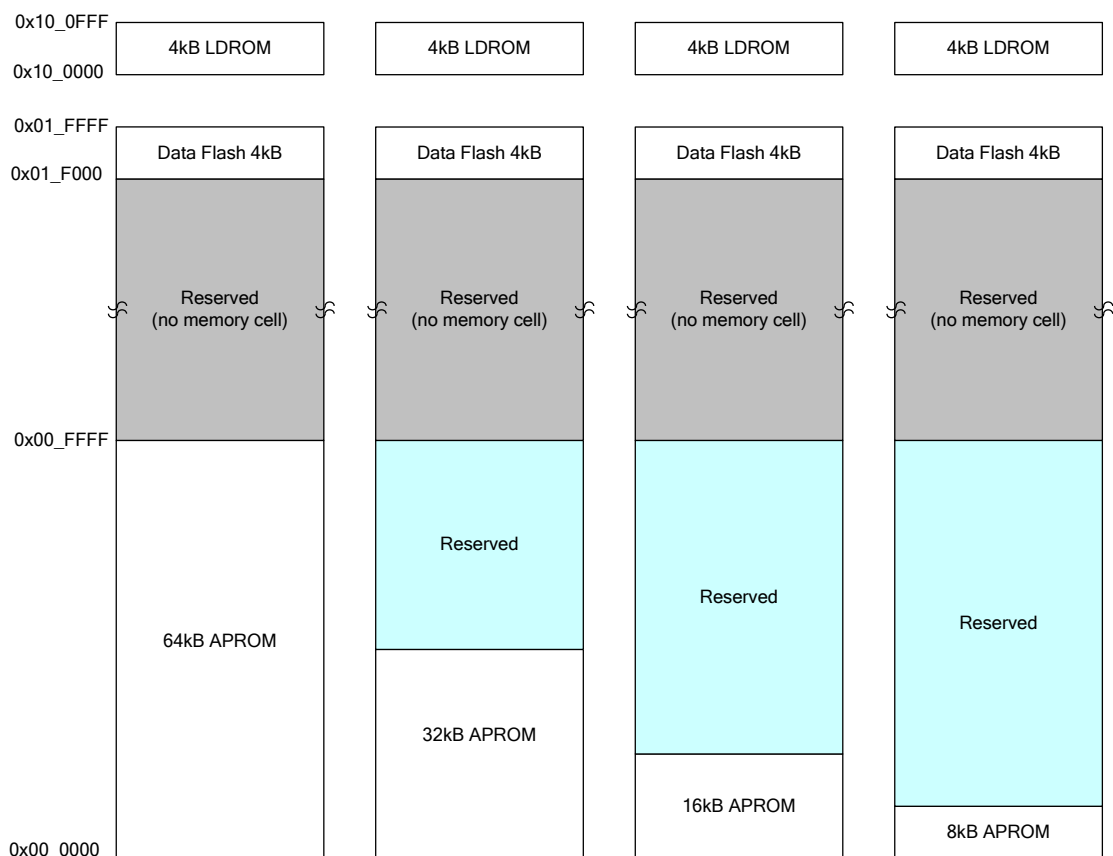
The FMC includes following features:

- 64/32/16/8kB application program memory (APROM).
- 4kB in system programming loader program memory (LDROM).
- 4kB data flash with 512 bytes page erase unit for user to store data
- Provide user configuration to control system logic.
- APROM cannot be updated when the MCU is running in APROM; LDROM can not be updated when the MCU is running in LDROM

Memory Address Map

Block Name	Size	Start Address	End Address
AP ROM	64 KB 32 KB 16 KB 8 KB	0x00000000	0x0000FFFF 0x00007FFF 0x00003FFF 0x00001FFF
Data Flash	4 KB	0x0001F000	0x0001FFFF
LD ROM	4KB	0x00100000	0x00100FFF
User Configuration	1 words	0x00300000	0x00300000

Flash Memory Structure



8/16/32/64kB Flash Memory Structure

10.3. Type Definition

E_FMC_BOOTSELECT

Enumeration identifier	Value	Description
E_FMC_APROM	0	Boot from APROM
E_FMC_LDROM	1	Boot from LDROMI

10.4. Functions

DrvFMC_EnableISP

Prototype

```
void DrvFMC_EnableISP (void);
```

Description

To enable ISP function.

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.

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.

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.

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.

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 Config update function is enabled, the user configuration can be update regardless of MCU is running in APROM or LDROM.

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.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

```
DrvFMC_DisableConfigUpdate ( );          /* Disable Config 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.

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.

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 is depended on the product of NuMicro™ M051 series. Please refer to [NuMicro™ M051 Series Selection Guide of Appendix](#) for APROM size. The corresponding functions in Config0 are described in FMC Section of TRM in details.

Parameter

u32addr [in]

Word address of APROM, LDROM, Data Flash or Config0.

u32data [in]

Word data to be programmed into APROM, LDROM, Data Flash or Config0.

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

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

Description

To read data from APROM, LDROM, Data Flash or Config0. The Memory Map of APROM is depended on the product of NuMicro™ M051 series. Please refer to [NuMicro™ M051 Series Selection Guide of Appendix](#) for APROM size.

Parameter

u32addr [in]

Word address of APROM, LDROM, Data Flash or Config0.

u32data [in]

The word data to store data from APROM, LDROM, Data Flash or Config0.

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

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

Description

To page erase APROM, LDROM, Data Flash or Config0. The flash page erase unit is 512 bytes. The Memory Map of APROM is depended on the product of NuMicro™ M051 series. Please refer to [NuMicro™ M051 Series Selection Guide of Appendix](#) for APROM size.

Parameter

u32addr [in]

Flash page base address of APROM, LDROM and Data Flash, or Config0 address.

Include

Driver/DrvFMC.h

Return Value

0 Succeed
<0 Failed

Example

```
/* Page Erase from 0x1F000 to 0x1F1FF */
DrvFMC_Erase (0x1F000);
```

DrvFMC_WriteConfig

Prototype

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

Description

To erase Config0 and write data into Config0. The corresponding functions in Config0 are described in FMC Section of TRM in details.

Parameter

u32data0 [in]

Word data to be programmed into Config0.

Include

Driver/DrvFMC.h

Return Value

0 Succeed
<0 Failed

Example

```
/* Program word data 0xFFFFFFFF into Config0 */
DrvFMC_Config (0xFFFFFFFF);
```


DrvFMC_ReadDataFlashBaseAddr

Prototype

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

Description

To read data flash base address. The base address is fixed at 0x1F000 for M051 series.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

Data Flash base address

Example

```
uint32_t u32Data;
/* Read Data Flash base address */
u32Data = DrvFMC_ReadDataFlashBaseAddr ( );
```

DrvFMC_EnableLowSpeedMode

Prototype

```
void DrvFMC_EnableLowSpeedMode (void);
```

Description

To enable flash access low speed mode. It can improve flash access performance when CPU runs at low speed.

Note

Set this bit only when $HCLK \leq 25\text{MHz}$. **If $HCLK > 25\text{MHz}$, CPU will fetch wrong code and cause fail result.**

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_EnableLowSpeedMode (); /* Enable flash access low speed mode */

DrvFMC_DisableLowSpeedMode

Prototype

void DrvFMC_DisableLowSpeedMode (void);

Description

To disable flash access low speed mode.

Parameter

None

Include

Driver/DrvFMC.h

Return Value

None

Example

DrvFMC_DisableLowSpeedMode (); /* Disable flash access low speed mode */

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

11. EBI Driver

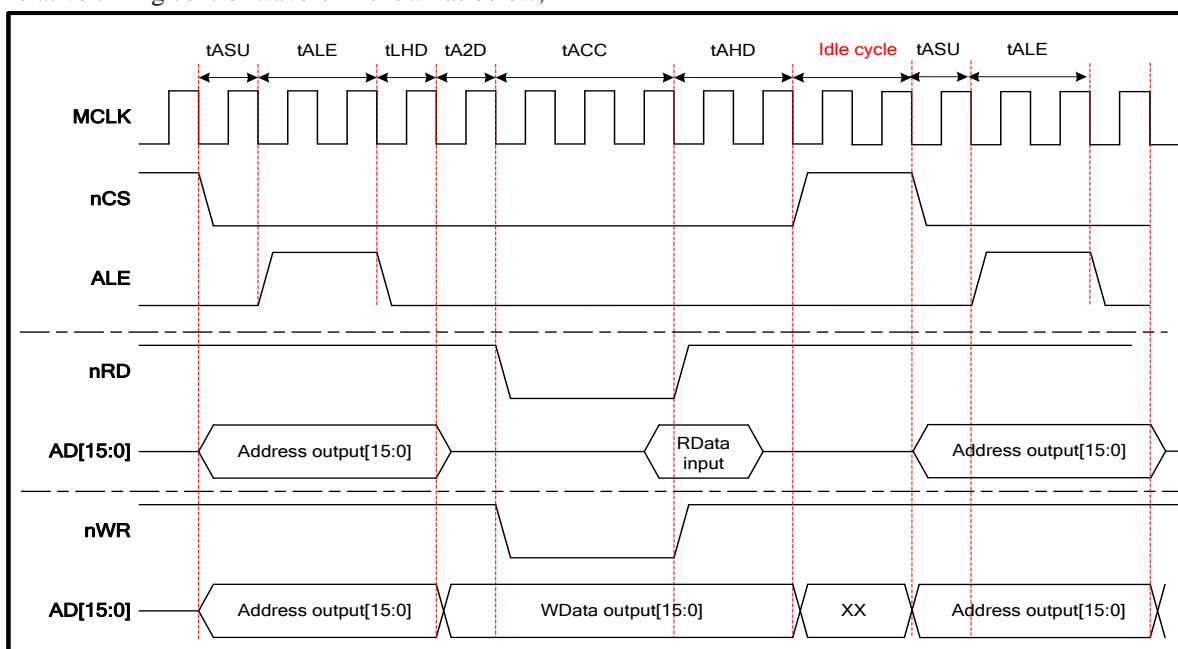
11.1. EBI Introduction

NuMicro™ M051 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.

11.2. EBI Feature

- Support external devices with max. 64K byte (8 bit data width)/128K byte (16 bit data width).
- Variable external bus base clock (MCLK) supported.
- Support 8 bit or 16 bit data width.
- Variable data access time (t_{ACC}), address latch enable time (t_{ALE}) and address hold time (t_{AHD}) supported.
- Address bus and data bus multiplex mode supported to save the address pins.
- Configurable idle cycle supported for difference access condition: Write command finish (W2X), Read-to-Read (R2R), Read-to-Write (R2W).
- Relative timing control waveform shown as below,



11.3. Type Definition

E_DRIVEBI_BUS_WIDTH

Enumeration Identifier	Value	Description
E_DRIVEBI_BUS_8BIT	0x0	EBI bus width is 8 bit
E_DRIVEBI_BUS_16BIT	0x1	EBI bus width is 16 bit

E_DRIVEBI_MCLKDIV

Enumeration Identifier	Value	Description
E_DRIVEBI_MCLKDIV_1	0x0	EBI output clock is HCLK/1
E_DRIVEBI_MCLKDIV_2	0x1	EBI output clock is HCLK/2
E_DRIVEBI_MCLKDIV_4	0x2	EBI output clock is HCLK/4
E_DRIVEBI_MCLKDIV_8	0x3	EBI output clock is HCLK/8
E_DRIVEBI_MCLKDIV_16	0x4	EBI output clock is HCLK/16
E_DRIVEBI_MCLKDIV_32	0x5	EBI output clock is HCLK/32
E_DRIVEBI_MCLKDIV_DEFAULT	0x6	EBI output clock is HCLK/1

11.4. 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, the DRVEBI_CONFIG_T structure.

DRVEBI_CONFIG_T

eBusWidth:

E_DRIVEBI_BUS_WIDTH, it could be E_DRIVEBI_BUS_8BIT or E_DRIVEBI_BUS_16BIT.

u32BaseAddress:

If eBusWidth is 8 bits: 0x60000000 <= u32BaseAddress <0x60010000
If eBusWidth is 16 bits: 0x60000000 <= u32BaseAddress <0x60020000

u32Size:

If eBusWidth is 8 bits: 0x0 < u32Size <= 0x10000
If eBusWidth 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.eBusWidth      = eDRVEBI_BUS_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]

It is meaning the DRVEBI_TIMING_T structure and included eMCLKDIV, u8ExttALE, u8ExtIR2R, u8ExtIR2W, u8ExtIW2X, u8ExttAHD and u8ExttACC.

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, $t_{ALE} = (u8ExttALE + 1) * MCLK$.

u8ExtIR2R: Idle cycle between Read-Read

0~15, idle cycle = $u8ExtIR2R * MCLK$

u8ExtIR2W: Idle cycle between Read-Write

0~15, idle cycle = $u8ExtIR2W * MCLK$

u8ExtIW2X: Idle cycle after Write

0~15, idle cycle = $u8ExtIW2X * MCLK$

u8ExttAHD: EBI address hold time

0~7, $t_{AHD} = (u8ExttAHD + 1) * MCLK$

u8ExttACC: EBI data access time

0~31, $t_{AHD} = (u8ExttACC + 1) * MCLK$

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.u8ExtIR2W = 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]

It is meaning the DRVEBI_TIMING_T structure and included eMCLKDIV, u8ExttALE, u8ExtIR2R, u8ExtIR2W, u8ExtIW2X, u8ExttAHD and u8ExttACC.

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 ();
```

12. Appendix

12.1. NuMicro™ M051 Series Selection Guide

Part number	Flash	SRAM	Data Flash	Connectivity			PWM	ADC	Timer	EBI	ISP ICP	I/O	Package
				UART	SPI	I2C							
M052LAN	8 KB	4 KB	4 KB	2	2	1	8	8x12-bit	4	v	v	up to 38	LQFP48
M052ZAN	8 KB	4 KB	4 KB	2	1	1	5	5x12-bit	4	-	v	up to 22	QFN32
M054LAN	16 KB	4 KB	4 KB	2	2	1	8	8x12-bit	4	v	v	up to 38	LQFP48
M054ZAN	16 KB	4 KB	4 KB	2	1	1	5	5x12-bit	4	-	v	up to 22	QFN32
M058LAN	32 KB	4 KB	4 KB	2	2	1	8	8x12-bit	4	v	v	up to 38	LQFP48
M058ZAN	32 KB	4 KB	4 KB	2	1	1	5	5x12-bit	4	-	v	up to 22	QFN32
M0516LAN	64 KB	4 KB	4 KB	2	2	1	8	8x12-bit	4	v	v	up to 38	LQFP48
M0516ZAN	64 KB	4 KB	4 KB	2	1	1	5	5x12-bit	4	-	v	up to 22	QFN32

12.2. Production Identity (PDID) Table

Part number	PDID
M052LAN	0x00005200
M052ZAN	0x00005203
M054LAN	0x00005400
M054ZAN	0x00005403
M058LAN	0x00005800
M058ZAN	0x00005803
M0516LAN	0x00005A00
M0516ZAN	0x00005A03

13. Revision History

Version	Date	Description
V1.00.001	Jan. 8, 2009	<ul style="list-style-type: none"> • Created
V1.00.002	July. 30, 2010	<ul style="list-style-type: none"> • Fix errors • Add example of API

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.