TOSHIBA

TOSHIBA TX03 Peripheral Driver User Guide (TMPM365)

Ver 1
Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M365UG-01E

RESTRICTIONS ON PRODUCT USE

- DO NOT USE THIS SOFTWARE WITHOUT THE SOFTWARE LISENCE AGREEMENT.

Index

1. Introduction

TOSHIBA TX03 Peripheral Driver is a set of drivers for all peripherals found on the TOSHIBA TX03 series microcontrollers. TMPM365 Peripheral Driver is an important part of TOSHIBA TX03 Peripheral Driver, which are designed for TMPM365 series MCUs.

TOSHIBA TX03 Peripheral Driver contains a collection of macros, data types, and structures for each peripheral.

The design goals of TOSHIBA TMPM365 Peripheral Driver:
- Completely written in C except the start-up routine and where not possible
- Cover all the peripherals on MCU

2. Organization of TOSHIBA TX03 Peripheral Driver

/Libraries
This folder contains all CMSIS files and TMPM365 Peripheral Drivers.

/Libraries/ TX03_CMSIS
This folder contains the TMPM365 CMSIS files: device peripheral access layer and core peripheral access layer.

/Libraries/TX03_Periph_Driver
This folder contains all the source code of the drivers, the core of TOSHIBA TMPM365 Peripheral Driver.

/Libraries/TX03_Periph_Driver/inc
This folder contains all the header files of TMPM365 Peripheral Drivers for each peripheral.

/Libraries/TX03_Periph_Driver/src
This folder contains all the source files of TMPM365 Peripheral Drivers for each peripheral.

/Project
This folder contains template project and examples for using TMPM365 Peripheral Driver.

/Project/Template
This folder contains template project of TOSHIBA TMPM365 Peripheral Driver.

/Project/Examples
This folder contains a set of examples for using TMPM365 Peripheral Driver

/Utilities/TMPM365-EVAL
This folder contains the configuration and driver files for hardware resources (e.g. led, key) on Toshiba TMPM365 -EVAL board.

3. ADC

3.1 Overview

This device contains a 12-bit, sequential-conversion analog/digital converter (ADC) with 12 analog input channels.

The 12-bit AD converter has the following features:
1. Start normal AD conversion and top-priority AD conversion by software activation, 16-bit timer(TMRB) activation or hardware activation with an external trigger input($\overline{\text{ADTRG}}$ pin).
2. AD conversion in 4 different modes:
   Fixed-channel single conversion mode
   Channel scan single conversion mode
   Fixed-channel repeat conversion mode
   Channel scan repeat conversion mode
3. Normal / Top-priority AD conversion completion interrupt
4. Normal / Top-priority AD conversion completion / busy flag
5. AD monitor function
   When the AD monitor function is enabled, an interrupt is generated if any comparison result is matched.
6. When AD conversion is completed, two types of DMA requests are supported.
7. Standby mode is supported.
8. Output switching monitor function.

The ADC API provides a set of functions for using the TMPM365 ADC modules. It includes ADC channel set, mode set, monitor function set, interrupt set, ADC status read, ADC result value read and so on.

This driver is contained in TX03_Periph_Driver\src\tmpm365_adc.c, with TX03_Periph_Driver\inc\tmpm365_adc.h containing the API definitions for use by applications.

3.2 API Functions

3.2.1 Function List

- void ADC_SWReset(void)
- void ADC_SetClk(uint32_t ***Sample_HoldTime***, uint32_t ***Prescaler_Output***)
- void ADC_Start(void)
- void ADC_SetScanMode(FunctionalState ***NewState***)
- void ADC_SetRepeatMode(FunctionalState ***NewState***)
- void ADC_SetINTMode(uint8_t ***INTMode***)
- void ADC_SetInputChannel(uint8_t ***InputChannel***)
- void ADC_SetScanChannel(uint8_t ***StartChannel***, uint8_t ***Range***)
- void ADC_SetVrefCut(uint8_t ***VrefCtrl***)
- void ADC_SetIdleMode(FunctionalState ***NewState***)
- void ADC_SetVref(FunctionalState ***NewState***)
- void ADC_SetInputChannelTop(uint8_t ***TopInputChannel***)
- void ADC_StartTopConvert(void)
- void ADC_SetMonitor(ADC_CMPCRx ***ADCMPx***, FunctionalState ***NewState***)
- void ADC_ConfigMonitor(ADC_CMPCRx ***ADCMPx***, ADC_MonitorTypeDef* ***Monitor***)
- void ADC_SetHWTrg(uint8_t ***HwSource***, FunctionalState ***NewState***)
- void ADC_SetHWTrgTop(uint8_t ***HwSource***, FunctionalState ***NewState***)
- ADC_State ADC_GetConvertState(void)

- ADC_Result ADC_GetConvertResult (uint8_t ***ADREGx***)
- void ADC_SetClkSupply(FunctionalState ***NewState***)
- void ADC_SetDMAReq(uint8_t ***DMAReq***, FunctionalState ***NewState***)

3.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) ADC setting by ADC_SetClk(), ADC_SetScanMode(), ADC_SetRepeatMode(), ADC_SetINTMode(), ADC_SetInputChannel(), ADC_SetScanChannel(), ADC_SetVref(), ADC_SetInputChannelTop(), ADC_SetMonitor(), ADC_ConfigMonitor(), ADC_SetHWTrg(), ADC_SetHWTrgTop().
2) ADC function start by ADC_Start(), ADC_StartTopConvert().
3) ADC state or data read functions by ADC_GetConvertState(), ADC_GetConvertResult().
4) ADC_SWReset(), ADC_SetVrefCut(), ADC_SetIdleMode(), ADC_SetClkSupply() and ADC_SetDMAReq() handle other specified functions.

3.2.3 Function Documentation

3.2.3.1 ADC_SWReset

Software reset ADC.

Prototype:
void
ADC_SWReset(void)

Parameters:
None

Description:
This function will software reset ADC.

Notes:
A software reset initializes all the registers except for ADCLK<ADCLK>.
Initialization takes 3μs in case of the software reset.

Return:
None

3.2.3.2 ADC_SetClk

Set ADC sample hold time and prescaler output.

Prototype:
void
ADC_SetClk(uint32_t ***Sample_HoldTime***,
uint32_t ***Prescaler_Output***)

Parameters:
Sample_HoldTime: Select ADC sample hold time.
This parameter can be one of the following values:

- **ADC_CONVERSION_CLK_10:** 10x <ADCLK>
- **ADC_CONVERSION_CLK_20:** 20x <ADCLK>
- **ADC_CONVERSION_CLK_30:** 30x <ADCLK>
- **ADC_CONVERSION_CLK_40:** 40x <ADCLK>
- **ADC_CONVERSION_CLK_80:** 80x <ADCLK>

Prescaler_Output: Select ADC prescaler output(ADCLK).
This parameter can be one of the following values:

- **ADC_FC_DIVIDE_LEVEL_1:** fc
- **ADC_FC_DIVIDE_LEVEL_2:** fc / 2
- **ADC_FC_DIVIDE_LEVEL_4:** fc / 4
- **ADC_FC_DIVIDE_LEVEL_8:** fc / 8

Description:
This function will set ADC sample hold time by ***Sample_HoldTime*** and prescaler output by ***Prescaler_Output***.

Notes:
Please do not use this function to change the analog to digital conversion clock setting during the analog to digital conversion. And **ADC_GetConvertState()** to check AD conversion state is not **BUSY**, then call this function.

Examples of sample hold time and conversion time as shown as below.

Prescaler_Output	***Sample_HoldTime***	**Conversion time**		
		fc=32MHz	fc=40MHz	fc=54MHz
ADC_FC_DIVIDE_LEVEL_1 (fc)	ADC_CONVERSION_CLK_10	1.25 µs	1.00 µs	-
	ADC_CONVERSION_CLK_20	1.56 µs	1.25 µs	-
	ADC_CONVERSION_CLK_30	1.88 µs	1.50 µs	-
	ADC_CONVERSION_CLK_40	2.19 µs	1.75 µs	-
	ADC_CONVERSION_CLK_80	3.44 µs	2.75 µs	-
ADC_FC_DIVIDE_LEVEL_2 (fc / 2)	ADC_CONVERSION_CLK_10	2.50 µs	2.00 µs	1.48 µs
	ADC_CONVERSION_CLK_20	3.13 µs	2.50 µs	1.85 µs
	ADC_CONVERSION_CLK_30	3.75 µs	3.00 µs	2.22 µs
	ADC_CONVERSION_CLK_40	4.38 µs	3.50 µs	2.59 µs
	ADC_CONVERSION_CLK_80	6.88 µs	5.50 µs	4.07 µs
ADC_FC_DIVIDE_LEVEL_4 (fc / 4)	ADC_CONVERSION_CLK_10	5.00 µs	4.00 µs	2.96 µs
	ADC_CONVERSION_CLK_20	6.25 µs	5.00 µs	3.70 µs
	ADC_CONVERSION_CLK_30	7.50 µs	6.00 µs	4.44 µs
	ADC_CONVERSION_CLK_40	8.75 µs	7.00 µs	5.19 µs
	ADC_CONVERSION_CLK_80	-	-	8.15 µs
ADC_FC_DIVIDE_LEVEL_8 (fc / 8)	ADC_CONVERSION_CLK_10	10.0 µs	8.00 µs	5.93 µs
	ADC_CONVERSION_CLK_20	-	10.0 µs	7.41 µs
	ADC_CONVERSION_CLK_30	-	-	8.89 µs
	ADC_CONVERSION_CLK_40	-	-	-
	ADC_CONVERSION_CLK_80	-	-	-

Setting the element indicated by "-" in the above table is prohibited. Specify ADCLK setting in the 1µs to 10µs range.

Return:
None

3.2.3.3 ADC_Start

Start AD conversion.

Prototype:
void
ADC_Start(void)

Parameters:
None

Description:
This function will start normal AD conversion.

Notes:
This function should be called after specifying the mode, which is one of the followings:

Fixed-channel single conversion mode
Channel scan single conversion mode
Fixed-channel repeat conversion mode
Channel scan repeat conversion mode

Please refer to the description of **ADC_SetScanMode(), ADC_SetRepeatMode(), ADC_SetInputChannel(), ADC_SetScanChannel()** for the details.

Before starting AD conversion, Vref should be enabled by calling **ADC_SetVref (ENABLE)**, wait for 3 μs during which time the internal reference voltage is stable, and then **ADC_Start()**.

Return:
None

3.2.3.4 ADC_SetScanMode

Enable or disable ADC scan mode.

Prototype:
void
ADC_SetScanMode(FunctionalState ***NewState***)

Parameters:
NewState: Specify ADC scan mode state
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable or disable ADC scan mode.

Return:
None

3.2.3.5 ADC_SetRepeatMode

Enable or disable ADC repeat mode.

Prototype:
void
ADC_SetRepeatMode(FunctionalState ***NewState***)

Parameters:
NewState: Specify ADC repeat mode state
This parameter can be one of the following values:
ENABLE or **DISABLE**.

Description:
This function will enable or disable ADC repeat mode.

Return:
None

3.2.3.6 ADC_SetINTMode

Set ADC interrupt mode in fixed channel repeat conversion mode.

Prototype:
void
ADC_SetINTMode(uint8_t ***INTMode***)

Parameters:
INTMode: Specify AD conversion interrupt mode.
The parameter can be one of the following values:

- **ADC_INT_SINGLE:** Generate in interrupt once every single conversion.
- **ADC_INT_CONVERSION_2:** Generate interrupt once every 2 conversions.
- **ADC_INT_CONVERSION_3:** Generate interrupt once every 3 conversions.
- **ADC_INT_CONVERSION_4:** Generate interrupt once every 4 conversions.
- **ADC_INT_CONVERSION_5:** Generate interrupt once every 5 conversions.
- **ADC_INT_CONVERSION_6:** Generate interrupt once every 6 conversions.
- **ADC_INT_CONVERSION_7:** Generate interrupt once every 7 conversions.
- **ADC_INT_CONVERSION_8:** Generate interrupt once every 8 conversions.

Description:
This function will specify ADC interrupt mode by ***INTMode*** setting.

Notes:
This function is valid only in fixed channel repeat conversion mode.
Examples for setting fixed channel repeat conversion mode:
1. **ADC_SetScanMode(DISABLE)**.
2. **ADC_SetRepeatMode(ENABLE)**.

Return:
None

3.2.3.7 ADC_SetInputChannel

Set ADC input channel.

Prototype:
void
ADC_SetInputChannel(uint8_t ***InputChannel***)

Parameters:

InputChannel: Analog input channel.
This parameter can be one of the following values:
ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07,
ADC_AN_08, ADC_AN_09, ADC_AN_10, ADC_AN_11.

Description:
This function will specify ADC input channel by ***InputChannel*** setting.

Notes:
Only one channel of **ADC_AN_00**~**ADC_AN_11** can be selected as normal conversion input each time.

Return:
None

3.2.3.8 ADC_SetScanChannel

Set ADC scan channel.

Prototype:
void
ADC_SetScanChannel (uint8_t ***StartChannel***, uint8_t ***Range***)

Parameters:
StartChannel: Specify the start channel to be scanned.
The parameter can be one of the following values:
ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07,
ADC_AN_08, ADC_AN_09, ADC_AN_10, ADC_AN_11.

Range: Specify the range of assignable channel scan value.
The parameter can be **1** to **12**.

Description:
This function will specify ADC start channels by ***StartChannel*** setting and channel scan range by ***Range*** setting.

Notes:
Valid channel scan setting values are shown as follows:

StartChannel	***Range***
ADC_AN_00	1 ~ 12
ADC_AN_01	1 ~ 11
ADC_AN_02	1 ~ 10
ADC_AN_03	1 ~ 9
ADC_AN_04	1 ~ 8
ADC_AN_05	1 ~ 7
ADC_AN_06	1 ~ 6
ADC_AN_07	1 ~ 5
ADC_AN_08	1 ~ 4
ADC_AN_09	1 ~ 3
ADC_AN_10	1 ~ 2
ADC_AN_11	1

In case of a setting other than listed above, AD conversion is not activated even if **ADC_Start()** is called.

Return:
None

3.2.3.9 ADC_SetVrefCut

Control AVREFH-AVREFL current.

Prototype:
void
ADC_SetVrefCut(uint8_t ***VrefCtrl***)

Parameters:
VrefCtrl: Specify how to apply AVREFH-AVREFL current.
This parameter can be one of the following values:

- **ADC_APPLY_VREF_IN_CONVERSION:** Apply the current only in conversion.
- **ADC_APPLY_VREF_AT_ANY_TIME:** Apply the current at any time except in RESET.

Description:
This function will control AVREFH-AVREFL current by ***VrefCtrl*** setting.

Return:
None

3.2.3.10 ADC_SetIdleMode

Set ADC operation in IDLE mode.

Prototype:
void
ADC_SetIdleMode(FunctionalState ***NewState***)

Parameters:
NewState: Specify ADC operation state in IDLE mode.
This parameter can be one of the following values:
ENABLE or **DISABLE**.

Description:
This function will enable or disable ADC operation state in system IDLE mode.
This function is necessary to be called before system enter IDLE mode.

Return:
None

3.2.3.11 ADC_SetVref

Set ADC Vref application control on or off.

Prototype:
void

ADC_SetVref(FunctionalState ***NewState***)

Parameters:
NewState: Specify AD conversion Vref application control.
This parameter can be one of the following values:
ENABLE or **DISABLE**.

Description:
This function will specify reference voltage on or off by ***NewState***.

Notes:
ADC_SetVref(DISABLE) should be called before system enter standby mode.

Return:
None

3.2.3.12 ADC_SetInputChannelTop

Select ADC top-priority conversion analog input channel.

Prototype:
void
ADC_SetInputChannelTop(uint8_t ***TopInputChannel***)

Parameters:
TopInputChannel: Analog input channel for top-priority conversion.
This parameter can be one of the following values:
ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07,
ADC_AN_08, ADC_AN_09, ADC_AN_10, ADC_AN_11.

Description:
This function will specify top-priority conversion analog input channel by ***TopInputChannel***.

Notes:
Only one channel of **ADC_AN_00**~**ADC_AN_11** can be selected as Top-priority conversion input each time.

Return:
None

3.2.3.13 ADC_StartTopConvert

Start top-priority AD conversion.

Prototype:
void
ADC_StartTopConvert(void)

Parameters:
None

Description:
This function will start top-priority AD conversion.

Notes:
This function should be called after **ADC_SetInputChannelTop()**.

Return:
None

3.2.3.14 ADC_SetMonitor

Enable or disable the specified ADC monitor module.

Prototype:
void
ADC_SetMonitor(ADC_CMPCRx ***ADCMPx***,
FunctionalState ***NewState***)

Parameters:
ADCMPx: Select which compare control register will be used.
The parameter can be one of the following values:
- **ADC_CMPCR_0:** ADCMPCR0
- **ADC_CMPCR_1:** ADCMPCR1

NewState: Specify ADC monitor function state.
This parameter can be one of the following values:
ENABLE or **DISABLE**.

Description:
This device has 2 AD monitor modules which are controlled by 2 compare control registers.
This function will specify compare control register by ***ADCMPx*** setting and specify ADC monitor function enable or disable by ***NewState*** setting.

Return:
None

3.2.3.15 ADC_ConfigMonitor

Configure the specified ADC monitor module.

Prototype:
void
ADC_ConfigMonitor(ADC_CMPCRx ***ADCMPx***,
ADC_MonitorTypeDef * ***Monitor***)

Parameters:
ADCMPx: Select which compare control register will be used.
The parameter can be one of the following values:
- **ADC_CMPCR_0:** ADCMPCR0
- **ADC_CMPCR_1:** ADCMPCR1

Monitor: A structure contains ADC monitor configuration including compare count, compare condition, compare mode, compare channel and compare value. Please refer to the comment for members of ADC_MonitorTypeDef for more detail usage.

Description:
This device has 2 AD monitor modules which are controlled by 2 compare control registers.
This function will specify compare control register by ***ADCMPx*** setting and specify ADC monitor configuration ***Monitor*** setting.

Notes: Please make sure to disable ADC monitor module before calling this function.

Return:
None

3.2.3.16 ADC_SetHWTrg

Set hardware trigger for normal AD conversion.

Prototype:
void
ADC_SetHWTrg(uint8_t ***HwSource***,
FunctionalState ***NewState***)

Parameters:
HwSource: Hardware source for activating normal AD conversion.
This parameter can be one of the following values:

- **ADC_EXT_TRG:** External trigger
- **ADC_MATCH_TB5RG0:** Match with timer register 0 (TB5RG0)

NewState: Specify state of hardware source for activating normal AD conversion.
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will specify hardware trigger source for activating normal AD conversion by ***HwSource*** setting and specify hardware trigger for normal AD conversion enable or disable by ***NewState*** setting.
This function also has relation with TB5 setting.

Notes:
The external trigger cannot be used for H/W activation of normal AD conversion when it is used for H/W activation of top-priority AD conversion.

Return:
None

3.2.3.17 ADC_SetHWTrgTop

Set hardware trigger for top-priority AD conversion.

Prototype:
void
ADC_SetHWTrgTop(uint8_t ***HwSource***,
FunctionalState ***NewState***)

Parameters:

HwSource: Hardware source for activating top-priority AD conversion.
This parameter can be one of the following values:

- **ADC_EXT_TRG:** External trigger
- **ADC_MATCH_TB4RG0:** Match with timer register 0 (TB4RG0)

NewState: Specify state of hardware source for activating top-priority AD conversion.
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will specify hardware trigger source for activating top-priority AD conversion by ***HwSource*** setting and specify hardware trigger for top-priority AD conversion enable or disable by ***NewState*** setting.
This function also has relation with TB4 setting.

Notes:
The external trigger cannot be used for H/W activation of normal AD conversion when it is used for H/W activation of top-priority AD conversion.

Return:
None

3.2.3.18 ADC_GetConvertState

Read AD conversion completion flag (normal and top-priority).

Prototype:
WorkState
ADC_GetConvertState(void)

Parameters:
None

Description:
This function will read AD conversion completion flag (both normal and top-priority). This function is used to check whether AD conversion has completed or not.

Return:
AD conversion state:
NormalComplete (Bit 1) means normal AD conversion is complete.
TopComplete (Bit 3) means top-priority AD conversion is complete.

3.2.3.19 ADC_GetConvertResult

Read AD conversion result.

Prototype:
ADC_Result
ADC_GetConvertResult(uint8_t ***ADREGx***)

Parameters:
ADREGx: Select ADC result register.
The parameter can be one of the following values:

ADC_REG_00, **ADC_REG_01**, **ADC_REG_02**, **ADC_REG_03**, **ADC_REG_04**, **ADC_REG_05**, **ADC_REG_06**, **ADC_REG_07**, **ADC_REG_08**, **ADC_REG_09**, **ADC_REG_10**, **ADC_REG_11**, **ADC_REG_SP**.

Description:
This function will read ADC register's result storage flag state, overrun state, output switching state and result value which specified by ***ADREGx*** setting.

Notes:
The ***ADREGx*** result stored state will set to **DONE** if a conversion result is stored. The result stored state will be cleared after ***ADREGx*** is read by this function.

The ***ADREGx*** overrun state will set to **ADC_OVERRUN** if a conversion result is overwritten before the conversion result storage register (ADREGx) is read. The overrun state will be cleared after overrun state is read by this function.

Relations between analog channel inputs and AD conversion result registers are shown in below tables.

Fixed-channel single mode	
Channel	Storage register
ADC_AN_00	ADC_REG_00
ADC_AN_01	ADC_REG_01
ADC_AN_02	ADC_REG_02
ADC_AN_03	ADC_REG_03
ADC_AN_04	ADC_REG_04
ADC_AN_05	ADC_REG_05
ADC_AN_06	ADC_REG_06
ADC_AN_07	ADC_REG_07
ADC_AN_08	ADC_REG_08
ADC_AN_09	ADC_REG_09
ADC_AN_10	ADC_REG_10
ADC_AN_11	ADC_REG_11

Fixed-channel repeat mode	
Interrupt mode	Storage register
Interrupt by each time AD/C	ADC_REG_00
Interrupt by each time 2 AD/C	ADC_REG_00 to ADC_REG_01
Interrupt by each time 3 AD/C	ADC_REG_00 to ADC_REG_02
Interrupt by each time 4 AD/C	ADC_REG_00 to ADC_REG_03
Interrupt by each time 5 AD/C	ADC_REG_00 to ADC_REG_04
Interrupt by each time 6 AD/C	ADC_REG_00 to ADC_REG_05
Interrupt by each time 7 AD/C	ADC_REG_00 to ADC_REG_06
Interrupt by each time 8 AD/C	ADC_REG_00 to ADC_REG_07

Channel scan single mode / repeat mode		
Start channel	Scan channel range	Storage register
ADC_AN_00	12 channels	ADC_REG_00 to ADC_REG_11
ADC_AN_01	11 channels	ADC_REG_01 to ADC_REG_11
ADC_AN_02	10 channels	ADC_REG_02 to ADC_REG_11
ADC_AN_03	9 channels	ADC_REG_03 to ADC_REG_11
ADC_AN_04	8 channels	ADC_REG_04 to ADC_REG_11
ADC_AN_05	7 channels	ADC_REG_05 to ADC_REG_11
ADC_AN_06	6 channels	ADC_REG_06 to ADC_REG_11
ADC_AN_07	5 channels	ADC_REG_07 to ADC_REG_11

ADC_AN_08	4 channels	ADC_REG_08 to ADC_REG_11
ADC_AN_09	3 channels	ADC_REG_09 to ADC_REG_11
ADC_AN_10	2 channels	ADC_REG_10 to ADC_REG_11
ADC_AN_11	1 channels	ADC_REG_11 to ADC_REG_11

The ADC mode setting, please refer to relate APIs.
For top-priority AD conversion, the result is stored in ADC_REG_SP.

Return:
AD conversion result:
ADResult (Bit 0 to Bit 11) means AD result value.
Stored (Bit 12) means AD result has been stored.
OverRun (Bit 13) means new AD result is stored before the old one is read.
OutputSwitching (Bit 14) means the output switching flag of AIN port.

3.2.3.20 ADC_SetClkSupply

Enable or disable ADC clock.

Prototype:
void
ADC_SetClkSupply(FunctionalState ***NewState***)

Parameters:
NewState: Specify ADC clock supply state.
The parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable or disable ADC clock supply.

Return:
None

3.2.3.21 ADC_SetDMAReq

Enable or disable DMA activation factor for normal or top-priority AD conversion.

Prototype:
void
ADC_SetDMAReq(uint8_t ***DMAReq***,
FunctionalState ***NewState***)

Parameters:
DMAReq: Specify AD conversion DMA request type.
The parameter can be one of the following values:
- **ADC_DMA_REQ_NORMAL:** normal AD conversion.
- **ADC_DMA_REQ_TOP:** top-priority AD conversion.

NewState: Specify AD conversion DMA activation factor.
This parameter can be one of the following values:
ENABLE or **DISABLE**.

Description:

This function will specify AD conversion DMA request type by ***DMAReq*** setting and specify AD conversion DMA activation factor by ***NewState*** setting.

Return:
None

3.2.4 Data Structure Description

3.2.4.1 ADC_MonitorTypeDef

Data Fields for this structure:

uint8_t
CmpChannel Select which ADC Result Register to be used,
which can be:
ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07,
ADC_AN_08, ADC_AN_09, ADC_AN_10, ADC_AN_11.

uint32_t
CmpCnt Define how many valid comparison times will be counted,
which can be **1** to **16**.

ADC_CmpCondition
Condition Conditon to compare AINx with ADCMPn (x= 0 to 11, n = 0 to 1),
which can be:

- **ADC_LARGER_THAN_CMP_REG:** If the value of the conversion result register is bigger than the comparison register 0, an interrupt is generated.
- **ADC_SMALLER_THAN_CMP_REG:** If the value of the conversion result register is smaller than the comparison register 0, an interrupt is generated.

ADC_CmpCntMode
CntMode Mode to compare AINx with ADCMPn (x = 0 to 11, n = 0 to 1),
which can be:

- **ADC_SEQUENCE_CMP_MODE:** Sequence mode.
- **ADC_CUMULATION_CMP_MODE:** Cumulation mode.

uint32_t
CmpValue Comparison value to be set in ADCMP0 or ADCMP1,
which can be **0 to 4095**

3.2.4.2 ADC_State

Data Fields for this structure:
uint32_t
All specifies AD conversion state.

Bit Fields:

uint32_t ***Reserved0*** (Bit 0)	reserved.
uint32_t ***NormalComplete*** (Bit 1)	means normal AD conversion is complete.
uint32_t ***Reserved1*** (Bit 2)	reserved.

uint32_t
TopComplete (Bit 3) means top-priority AD conversion is complete.
uint32_t
Reserved2 (Bit 4 to Bit 31) reserved.

3.2.4.3 ADC_Result

Data Fields for this structure:
uint32_t
All specifies AD conversion result.

Bit Fields:
uint32_t
ADResult (Bit 0 to Bit 11) means AD result value.
uint32_t
Stored (Bit 12) means AD result has been stored.
uint32_t
OverRun (Bit 13) means new AD result is stored before the old one is read.
uint32_t
OutputSwitching (Bit 14) means the output switching flag of AIN port.
uint32_t
Reserved (Bit 15 to Bit 31) reserved.

4. CG

4.1 Overview

The CG API provides a set of functions for using the TMPM365 CG modules as the following:

- Set up high-speed oscillators and input clock, set up the PLL.
- Select clock gear, prescaler clock, the PLL and oscillator.
- Set warm up timer and read the warm up result.
- Set up Low Power Consumption Modes.
- Switch among Normal Mode and Low Power Consumption Modes.
- Configure the interrupts for releasing standby modes, clear interrupt request.

This driver is contained in TX03_Periph_Driver\src\tmpm365_cg.c , with TX03_Periph_Driver\inc\tmpm365_cg.h containing the API definitions for use by applications.

The following symbols fosc, fpll, fc, fgear, fsys, fperiph, ΦT0 are used for kinds of clock in CG. Please refer to the clock system diagram in section “Clock System Block Diagram” of the datasheet for their meaning.

fosc : Clock generated by internal oscillator. Clock input from the X1 and X2 pins..
fpll : Clock multiplied by 8 by PLL.
fc : Clock specified by CGPLLSEL<PLLSEL> (high-speed clock).
fgear : Clock specified by CGSYSCR<GEAR[2:0]>.
fsys : Clock specified by same as fgear clock (system clock).
fperiph : Clock specified by CGSYSCR<FPSEL>.
ΦT0 : Clock specified by CGSYSCR<PRCK[2:0]> (prescaler clock).

4.2 API Functions

4.2.1 Function List

- void CG_SetFgearLevel(CG_DivideLevel ***DivideFgearFromFc***)
- CG_DivideLevel CG_GetFgearLevel(void)
- void CG_SetPhiT0Src(CG_PhiT0Src ***PhiT0Src***)
- CG_PhiT0Src CG_GetPhiT0Src(void)
- Result CG_SetPhiT0Level(CG_DivideLevel ***DividePhiT0FromFc***)
- CG_DivideLevel CG_GetPhiT0Level(void)
- void CG_SetSCOUTSrc(CG_SCOUTSrc ***Source***)
- CG_SCOUTSrc CG_GetSCOUTSrc(void)
- void CG_SetWarmUpTime(CG_WarmUpSrc ***Source***, uint16_t ***Time***)
- void CG_StartWarmUp(void)
- WorkState CG_GetWarmUpState(void)
- Result CG_SetFPLLValue(CG_FpllValue ***NewValue***)
- CG_FpllValue CG_GetFPLLValue(void)
- Result CG_SetPLL(FunctionalState ***NewState***)
- FunctionalState CG_GetPLLState(void)
- Result CG_SetFosc(CG_FoscSrc Source, FunctionalState ***NewState***)

- void CG_SetFoscSrc(CG_FoscSrc ***Source***)
- CG_FoscSrc CG_GetFoscSrc(void)
- FunctionalState CG_GetFoscState(CG_FoscSrc ***Source***)
- void CG_SetSTBYMode(CG_STBYMode ***Mode***)
- CG_STBYMode CG_GetSTBYMode(void)
- void CG_SetPinStateInStop1Mode(FunctionalState ***NewState***)
- FunctionalState CG_GetPinStateInStop1Mode(void)
- Result CG_SetFcSrc(CG_FcSrc ***Source***)
- CG_FcSrc CG_GetFcSrc(void)
- void CG_SetUSBSrcClk(CG_USBSrc ***Source***)
- CG_USBSrc CG_GetUSBSrcClk(void)
- void CG_SetUSBClkState(FunctionalState ***NewState***)
- FunctionalState CG_GetUSBClkState(void)
- void CG_SetProtectCtrl(FunctionalState ***NewState***)
- void CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,
  CG_INTActiveState ***ActiveState***,
  FunctionalState ***NewState***)
- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***)
- void CG_ClearINTReq(CG_INTSrc ***INTSource***)
- CG_NMIFactor CG_GetNMIFlag(void)
- CG_ResetFlag CG_GetResetFlag(void)

4.2.2 Detailed Description

The CG APIs can be broken into three groups by function:

1) One group of APIs are in charge of clock selection, such as:
   CG_SetFgearLevel(), CG_GetFgearLevel(),CG_SetPhiT0Src(),CG_GetPhiT0Src(), CG_SetPhiT0Level(), CG_GetPhiT0Level(), CG_SetSCOUTSrc(), CG_GetSCOUTSrc(), CG_SetWarmUpTime(), CG_StartWarmUp(), CG_GetWarmUpState(),CG_SetFPLLValue(), CG_GetFPLLValue(),CG_SetPLL(), CG_GetPLLState(), CG_SetFosc(),CG_SetFoscSrc(), CG_GetFoscSrc(), CG_GetFoscState(),CG_SetFcSrc(),CG_GetFcSrc(),CG_SetUSBSrcClk(),CG_GetUSBSrcClk(),CG_SetUSBClkState(),CG_GetUSBClkState(),CG_SetProtectCtrl().

2) The 2nd group of APIs handle settings of standby modes:
   CG_SetSTBYMode(), CG_GetSTBYMode( ), CG_SetPinStateInStop1Mode(),CG_GetPinStateInStop1Mode(),

3) The other APIs handle settings of interrupts:
   CG_SetSTBYReleaseINTSrc(), CG_GetSTBYReleaseINTState(), CG_ClearINTReq(), CG_GetNMIFlag(), CG_GetResetFlag(),

4.2.3 Function Documentation

4.2.3.1 CG_SetFgearLevel

Set the dividing level between clock fgear and fc.

Prototype:
void
CG_SetFgearLevel(CG_DivideLevel ***DivideFgearFromFc***)

Parameters:
DivideFgearFromFc: the divide level between fgear and fc
The value could be the following values:

- **CG_DIVIDE_1**: fgear = fc
- **CG_DIVIDE_2**: fgear = fc/2

- **CG_DIVIDE_4**: fgear = fc/4
- **CG_DIVIDE_8**: fgear = fc/8
- **CG_DIVIDE_16**: fgear = fc/16

Description :
This function will set the dividing level between clock fgear and fc.

Return:
None

4.2.3.2 CG_GetFgearLevel

Get the dividing level between fgear and fc.

Prototype:
CG_DivideLevel
CG_GetFgearLevel (void)

Parameters:
None

Description:
This function will get the dividing level between fgear and fc.
If the value "Reserved" is read from the register, the API will return **CG_DIVIDE_UNKNOWN**.

Return:
The dividing level between clock fgear and fc.
The value returned can be one of the following values:
CG_DIVIDE_1: fgear = fc
CG_DIVIDE_2: fgear = fc/2
CG_DIVIDE_4: fgear = fc/4
CG_DIVIDE_8: fgear = fc/8
CG_DIVIDE_16: fgear = fc/16
CG_DIVIDE_UNKNOWN: invalid data is read

4.2.3.3 CG_SetPhiT0Src

Set fperiph for PhiT0.

Prototype:
void
CG_SetPhiT0Src(CG_PhiT0Src ***PhiT0Src***)

Parameters:
PhiT0Src: Select PhiT0 source.
This parameter can be one of the following values:
- **CG_PHIT0_SRC_FGEAR** means PhiT0 source is fgear.
- **CG_PHIT0_SRC_FC** means PhiT0 source is fc.

Description:
This function selects the source for PhiT0.

Return:
None

4.2.3.4 CG_GetPhiT0Src

Get the PhiT0 source.

Prototype:
CG_PhiT0Src
CG_GetPhiT0Src (void)

Parameters:
None

Description:
This function will get the PhiT0 source.

Return:
CG_PHIT0_SRC_FGEAR means PhiT0 source is fgear.
CG_PHIT0_SRC_FC means PhiT0 source is fc.

4.2.3.5 CG_SetPhiT0Level

Set the dividing level between PhiT0 (ΦT0) and fc.

Prototype:
Result
CG_SetPhiT0Level (CG_DivideLevel ***DividePhiT0FromFc***)

Parameters:
DividePhiT0FromFc: divide level between PhiT0(ΦT0) and fc.
This parameter can be one of the following values:

- **CG_DIVIDE_1**: ΦT0 **=** fc
- **CG_DIVIDE_2**: ΦT0 = fc/2
- **CG_DIVIDE_4**: ΦT0 = fc/4
- **CG_DIVIDE_8**: ΦT0 = fc/8
- **CG_DIVIDE_16**: ΦT0 = fc/16
- **CG_DIVIDE_32**: ΦT0 = fc/32
- **CG_DIVIDE_64**: ΦT0 = fc/64
- **CG_DIVIDE_128**: ΦT0 = fc/128
- **CG_DIVIDE_256**: ΦT0 = fc/256
- **CG_DIVIDE_512**: ΦT0 = fc/512

Description:
This function will set the dividing level of prescaler clock.

Return:
SUCCESS means the setting has been written to registers successfully.
ERROR means the setting has not been written to registers**.**

4.2.3.6 CG_GetPhiT0Level

Get the dividing level between clock ΦT0 and fc.

Prototype:
CG_DivideLevel
CG_GetPhiT0Level(void)

Parameters:
None

Description:
This function will get the dividing level of prescaler clock.
If the value “Reserved” is read from the register, the API will return **CG_DIVIDE_UNKNOWN**.

Return:
Dividing level between clock ΦT0 and fc, the value will be one of the following:
CG_DIVIDE_1: ΦT0 = fc
CG_DIVIDE_2: ΦT0 = fc/2
CG_DIVIDE_4: ΦT0 = fc/4
CG_DIVIDE_8: ΦT0 = fc/8
CG_DIVIDE_16: ΦT0 = fc/16
CG_DIVIDE_32: ΦT0 = fc/32
CG_DIVIDE_64: ΦT0 = fc/64
CG_DIVIDE_128 : ΦT0 = fc/128
CG_DIVIDE_256 : ΦT0 = fc/256
CG_DIVIDE_512 : ΦT0 = fc/512
CG_DIVIDE_UNKNOWN : invalid data is read.

4.2.3.7 CG_SetSCOUTSrc

Set the clock source of SCOUT output.

Prototype:
void
CG_SetSCOUTSrc(CG_SCOUTSrc ***Source***)

Parameters:
Source: select clock source of SCOUT.
This parameter can be one of the following values:
- **CG_SCOUT_SRC_HALF_FSYS**: SCOUT source is set to fsys/2.
- **CG_SCOUT_SRC_FSYS**: SCOUT source is set to fsys.
- **CG_SCOUT_SRC_PHIT0**: SCOUT source is set to ΦT0.

Description:
This function will set the clock source of SCOUT output.

Return:
None

4.2.3.8 CG_GetSCOUTSrc

Get the clock source of SCOUT output.

Prototype:
SCOUTSrc
CG_GetSCOUTSrc(void)

Parameters:
None

Description:
This function will get the clock source of SCOUT output.

Return:
The clock source of SCOUT output:

CG_SCOUT_SRC_HALF_FSYS: SCOUT source is set to fsys/2
CG_SCOUT_SRC_FSYS: SCOUT source is fsys
CG_SCOUT_SRC_PHIT0: SCOUT source is ΦT0
CG_SCOUT_SRC_ UNKNOWN: Invalid data is read.

4.2.3.9 CG_SetWarmUpTime

Set the warm up time.

Prototype:
void
CG_SetWarmUpTime (CG_WarmUpSrc ***Source***,
uint16_t ***Time***)

Parameters:
Source: select source of warm-up counter.
- **CG_WARM_UP_SRC_OSC_EXT**: external clock is selected as timer source,
- **CG_WARM_UP_SRC_OSC_INT**: internal clock is selected as timer source,

Time: Time value range is 0U to 0xFFFFU.

Description:
This function will set the warm-up time and warm-up counter. And the formula is as the following:

Number of warm-up cycle = (warm-up time to set) / (input frequency cycle(s)).

Example of calculating register value for warm-up time:

/* When using high-speed oscillator 8MHz, and set warm-up time 5ms. */
So value = (warm-up time to set) / (input frequency cycle(s)) = 5ms / (1/8MHz) = 4000cycle = 0x9C40.
Round lower 4 bit off, set 0x9C4 to CGOSCCR<WUODR[11:0]>

Return:
None.

4.2.3.10 CG_StartWarmUp

Start operation of warm up timer for oscillator.

Prototype:
void
CG_StartWarmUp (void)

Parameters:
None

Description:
This function will start the warm up timer.

Return:
None

4.2.3.11 CG_GetWarmUpState

Check whether warm up is completed or not.

Prototype:
WorkState
CG_GetWarmUpState (void)

Parameters:
None

Description:
This function will check that warm-up operation is in progress or finished.

Example of using warm-up timer:

```
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT, 0x32);
/* start warm up */
CG_StartWarmUp();
/* check warm up is finished or not*/
While( CG_GetWarmUpState() == BUSY);
```

Return:
Warm up state:
DONE: means warm-up operation is finished.
BUSY: means warm-up operation is in progress.

4.2.3.12 CG_SetFPLLValue

Set PLL multiplying value

Prototype:
Result
CG_SetFPLLValue(CG_FpllValue ***NewValue***)

Parameters:
NewValue:
- **CG_FPLL_MULTIPLY_8**: 8 multiplying value is selected.

Description:
This function sets PLL multiplying value.

Return:
SUCCESS: operation is finished successfully.
ERROR: operation is not done.

4.2.3.13 CG_GetFPLLValue

Get the value of PLL setting.

Prototype:
CG_FpllValue
CG_GetFPLLValue(void)

Parameters:
None

Description:

This function will get the PLL multiplying value.
If the value “Reserved” is read from the register, the API will return **CG_FPLL_MULTIPLY_UNKNOWN**.

Return:
The source of PLL multiplying value
CG_FPLL_MULTIPLY_8: 8 multiplying is selected.
CG_FPLL_MULTIPLY_UNKNOWN: invalid data is read.

4.2.3.14 CG_SetPLL

Enable or disable the PLL circuit.

Prototype:
Result
CG_SetPLL(FunctionalState ***NewState***)

Parameters:
NewState:
- **ENABLE**: to enable the PLL circuit.
- **DISABLE**: to disable the PLL circuit.

Description:
This function will enable or disable the PLL circuit as the input parameter.

Return:
SUCCESS: operation is finished successfully.
ERROR: operation is not done.

4.2.3.15 CG_GetPLLState

Get the state of PLL circuit.

Prototype:
FunctionalState
CG_GetPLLState(void)

Parameters:
None

Description:
This function will get the state of PLL circuit.

Return:
The state of PLL
ENABLE: PLL is enabled.
DISABLE: PLL is disabled.

4.2.3.16 CG_SetFosc

Enable or disable high-speed oscillator (fosc).

Prototype:
Result
CG_SetFosc(CG_FoscSrc ***Source***,
FunctionalState ***NewState***)

Parameters:
Source: select clock source of fosc.
This parameter can be one of the following values:
- **CG_FOSC_OSC_EXT**: external high-speed oscillator is selected,
- **CG_FOSC_OSC_INT**: internal high-speed oscillator is selected.

NewState
- **ENABLE**: to enable the high-speed oscillator.
- **DISABLE**: to disable the high-speed oscillator.

Description:
This function will enable or disable the high-speed oscillator as the input parameter.

Return:
SUCCESS: operation is finished successfully.
ERROR: operation is not done.

4.2.3.17 CG_SetFoscSrc

Set the source of high-speed oscillation (fosc).

Prototype:
void
CG_SetFoscSrc(CG_FoscSrc Source)

Parameters:
Source: select source for fosc.
This parameter can be one of the following values:
- **CG_FOSC_OSC_EXT**: external high-speed oscillator is selected,
- **CG_FOSC_CLKIN_EXT**: external clock input is selected.
- **CG_FOSC_OSC_INT**: internal high-speed oscillator is selected.

Description:
This function will set the source for high-speed oscillation (fosc).

Return:
None

4.2.3.18 CG_GetFoscSrc

Get the source of the high-speed oscillator.

Prototype:
CG_FoscSrc
CG_GetFoscSrc(void)

Parameters:
None

Description:
This function will get the source of the high-speed oscillator.

Return:
The source of fosc
CG_FOSC_OSC_EXT: external high-speed oscillator is selected,
CG_FOSC_CLKIN_EXT: external clock input is selected.

CG_FOSC_OSC_INT: internal high-speed oscillator is selected.

4.2.3.19 CG_GetFoscState

Get the state of the high-speed oscillator.

Prototype:
FunctionalState
CG_GetFoscState(CG_FoscSrc Source)

Parameters:
Source: select source for fosc.
- **CG_FOSC_OSC_EXT**: external high-speed oscillator is selected,
- **CG_FOSC_OSC_INT**: internal high-speed oscillator is selected.

Description:
This function will get the state of the high-speed oscillator.

Return:
The state of fosc
ENABLE: fosc is enabled.
DISABLE: fosc is disabled.

4.2.3.20 CG_SetSTBYMode

Set the standby mode.

Prototype:
void
CG_SetSTBYMode(CG_STBYMode ***Mode***)

Parameters:
Mode: the low power consumption mode, the description of each value is as the following:
- **CG_STBY_MODE_STOP1**: STOP1 mode. All the internal circuits including the internal oscillator are brought to a stop.
- **CG_STBY_MODE_IDLE**: IDLE mode. Only CPU stop in this mode.

Description:
This function will change the setting of the standby mode to enter when using standby instruction.

Return:
None

4.2.3.21 CG_GetSTBYMode

Get the standby mode.

Prototype:
CG_STBYMode
CG_GetSTBYMode (void)

Parameters:
None

Description:

This function will get the setting of standby mode.
If the value “Reserved” is read, “**CG_STBY_MODE_UNKNOWN**” will be returned.

Return:
The low power mode:
CG_STBY_MODE_STOP1: STOP1 mode.
CG_STBY_MODE_IDLE: IDLE mode
CG_STBY_MODE_UNKNOWN: Invalid data is read.

4.2.3.22 CG_SetPinStateInStop1Mode

Set pin status in stop1 mode

Prototype:
void
CG_SetPinStateInStop1Mode(FunctionalState ***NewState***)

Parameters:
NewState:
- **DISABLE:** <DRVE>=0
- **ENABLE:** <DRVE>=1

For the detailed state of port corresponding to “<DRVE>=0” or “<DRVE>=1”, please refer to the table “Pin Status in the STOP1/STOP2 Mode” in the datasheet.

Description:
This function sets pin status in stop1 mode.

Return:
None

4.2.3.23 CG_GetPinStateInStop1Mode

Get pin status in stop1 mode

Prototype:
FunctionalState
CG_GetPinStateInStop1Mode(void)

Parameters:
None

Description:
This function gets the state of pin status in stop1 mode.

Return:
The pin state in stop1 mode
DISABLE: <DRVE>=0
ENABLE: <DRVE>=1

4.2.3.24 CG_SetFcSrc

Set the clock source of fc

Prototype:

Result
CG_SetFcSrc(CG_FcSrc ***Source***)

Parameters:
Source: the source for fc
This parameter can be one of the following values:
- **CG_FC_SRC_FOSC** : fc source will be set to fosc
- **CG_FC_SRC_ QUARTER_FPLL**: fc source will be set to fpll/4

Description:
This function will set the clock source of fc.

Return:
SUCCESS: set clock souce for fc successfully
ERROR: clock source of fc is not changed.

4.2.3.25 CG_GetFcSrc

Get the clock source of fc.

Prototype:
CG_FcSrc
CG_GetFosc (void)

Parameters:
None

Description:
This function will get the clock source of fc.

Return:
The clock source of fc
The value returned can be one of the following values:
CG_FC_SRC_FOSC: fc source is set to fosc.
CG_FC_SRC_ QUARTER_FPLL: fc source is set to fpll/4.

4.2.3.26 CG_SetUSBSrcClk

Set the source of USB device block

Prototype:
void
CG_SetUSBSrcClk(CG_USBSrc ***Source***)

Parameters:
Source: the source for USB device block
This parameter can be one of the following values:
- **CG_USB_SRC_CLK_PLL** USB source clock will be set to fpll
- **CG_USB_SRC_CLK_EXT** USB source clock will be set to external input clock

Description:
This function selects source clock to USB device block.

Return:
None

4.2.3.27 CG_GetUSBSrcClk

Get the source of USB device block

Prototype:
CG_USBSrc
CG_GetUSBSrcClk(void)

Parameters:
None

Description:
This function will get the source of USB device block.

Return:
The source for USB device block.
The value returned can be one of the following values:
CG_USB_SRC_CLK_PLL USB source clock will be set to fpll
CG_USB_SRC_CLK_EXT USB source clock will be set to external input clock

4.2.3.28 CG_SetUSBClkState

Enable or disable USB source clock

Prototype:
void
CG_SetUSBClkState(FunctionalState ***NewState***)

Parameters:
NewState
- **DISABLE:** < USBCLKEN >=0 disable USB source clock
- **ENABLE:** < USBCLKEN >=1 enable USB source clock

Description:
This function enables or disables USB source clock.

Return:
None

4.2.3.29 CG_GetUSBClkState

Get the state of USB source clock

Prototype:
FunctionalState
CG_GetUSBClkState(void)

Parameters:
None

Description:
This function will get the status of USB source clock..

Return:
The status of USB source clock..
The value returned can be one of the following values:
DISABLE: < USBCLKEN>=0 clock disable

ENABLE: < USBCLKEN>=1 clock enable

4.2.3.30 CG_SetProtectCtrl

Enable or disable to protect CG registers.

Prototype:
void
CG_SetProtectCtrl(FunctionalState ***NewState***)

Parameters:
NewState
- **DISABLE:** < CGPROTECT>= Except 0xC1 Register write disable
- **ENABLE:** < CGPROTECT>=0xC1 Register write enable

Description:
This function enables or disables CG registers to be written.

Return:
None

4.2.3.31 CG_SetSTBYReleaseINTSrc

Set the INT source for releasing low power mode.

Prototype:
void
CG_SetSTBYReleaseINTSrc (CG_INTSrc ***INTSource***,
CG_INTActiveState ***ActiveState***,
FunctionalState ***NewState***)

Parameters:
INTSource: select the INT source for releasing standby mode
This parameter can be one of the following values:
- **CG_INT_SRC_0** : INT0
- **CG_INT_SRC_1** : INT1
- **CG_INT_SRC_2** : INT2
- **CG_INT_SRC_3** : INT3
- **CG_INT_SRC_4** : INT4
- **CG_INT_SRC_5** : INT5
- **CG_INT_SRC_6** : INT6
- **CG_INT_SRC_7** : INT7
- **CG_INT_SRC_8** : INT8
- **CG_INT_SRC_9** : INT9
- **CG_INT_SRC_USB_PON** : USB Power On connection detection interrupt
- **CG_INT_SRC_USB_WKUP** : USB Wake-up interrupt

ActiveState: select the active state for release trigger.
This parameter can be one of the following values:
- **CG_INT_ACTIVE_STATE_L**: active on low level
- **CG_INT_ACTIVE_STATE_H**: active on high level
- **CG_INT_ACTIVE_STATE_FALLING**: active on falling edge
- **CG_INT_ACTIVE_STATE_RISGING**: active on rising edge
- **CG_INT_ACTIVE_STATE_BOTH_EDGES**: active on both edges

NewState: enable or disable this release trigger
This parameter can be one of the following values:

- **ENABLE**: clear standby mode when the interrupt occurs and the condition of active state is matched.
- **DISABLE:** do not clear standby mode even though the interrupt occurs and the condition of active state is matched.

Description:
This function will set the INT source for releasing standby mode.

Return:
None

4.2.3.32 CG_GetSTBYReleaseINTState

Get the active state of INT source for standby clear request.

Prototype:
CG_INT_ActiveState
CG_GetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***)

Parameters:
INTSource: select the release INT source
This parameter can be one of the following values:
CG_INT_SRC_0, CG_INT_SRC_1, CG_INT_SRC_2, CG_INT_SRC_3,
CG_INT_SRC_4, CG_INT_SRC_5, CG_INT_SRC_6, CG_INT_SRC_7,
CG_INT_SRC_8, CG_INT_SRC_9,
CG_INT_SRC_USB_PON, CG_INT_SRC_USB_WKUP.

Description:
This function will get the active state of INT source for standby clear request.

Return:
Active state of the input INT
The value returned can be one of the following values:
CG_INT_ACTIVE_STATE_FALLING: active on falling edge
CG_INT_ACTIVE_STATE_RISING: active on rising edge
CG_INT_ACTIVE_STATE_BOTH_EDGES: active on both edges
CG_INT_ACTIVE_STATE_INVALID: invalid

4.2.3.33 CG_ClearINTReq

Clears the input INT request.

Prototype:
void
CG_ClearINTReq(CG_INTSrc ***INTSource***)

Parameters:
INTSource: select the release INT source.
This parameter can be one of the following values:
CG_INT_SRC_0, CG_INT_SRC_1, CG_INT_SRC_2, CG_INT_SRC_3,
CG_INT_SRC_4, CG_INT_SRC_5, CG_INT_SRC_6, CG_INT_SRC_7,
CG_INT_SRC_8, CG_INT_SRC_9,
CG_INT_SRC_USB_PON, CG_INT_SRC_USB_WKUP.
Description:
This function will clear the INT request for releasing standby mode.

Return:

None

4.2.3.34 CG_GetNMIFlag

Get the NMI flag that shows who triggered NMI

Prototype:
CG_NMI_Factor
CG_ GetNMIFlag (void)

Parameters:
None

Description:
This function gets the NMI flag showing what triggered Non-maskable interrupt.

Return:
NMI value:
WDT (Bit 0) means generated from WDT.
NMIPin(Bit 1) means generated from NMI pin.

4.2.3.35 CG_GetResetFlag

Get the reset flag that shows the trigger of reset and clear the reset flag

Prototype:
CG_ResetFlag
CG_GetResetFlag(void)

Parameters:
None

Description:
This function gets the reset flag showing what triggered reset.

Return:
Reset flag:
ResetPin (Bit 0) means reset from Reset pin.
Reserved(Bit1) means reserved.
WDTReset (Bit 2) means reset from WDT.
DebugReset (Bit 4) means reset from SYSRESETREQ.

4.2.4 Data Structure Description

4.2.4.1 CG_NMIFactor

Data Fields:
uint32_t
All specifies CGNMI source generation state.

Bit Fields:
uint32_t
WDT(Bit 0) means generated from WDT.
uint32_t
NMIPin(Bit 1) means generated from NMI pin..

4.2.4.2 CG_ResetFlag

Data Fields:
uint32_t
All specifies CG reset source.

Bit Fields:
uint32_t
ResetPin(Bit 0) means reset from Reset pin.
uint32_t
Reserved(Bit 1) means reserved.
uint32_t
WDTReset(Bit 2) means reset from WDT.
uint32_t
DebugReset(Bit 4) means reset from SYSRESETREQ..

5. DMAC

5.1 Overview

TOSHIBA TMPM365 has one DMA controller(UNITA) controlled by DMA request select registers, and DMA controller has two channels. Each channel can operate in one of four transferring types (memory to memory, memory to peripheral, peripheral to memory, peripheral to peripheral). The priority of UNITA channel 0 is higher than UNITA channel 1.

The DMA driver APIs provide a set of functions to configure DMAC, including such parameters as source address, source address incremented state, transfer source bit width, transfer source burst size, destination address, destination address incremented state, transfer destination bit width, transfer destination burst size, transfer size, transfer direction, transfer peripheral and transfer interrupt state and so on.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm365_dmac.c, with \Libraries/TX03_Periph_Driver\inc\tmpm365_dmac.h containing the API definitions for use by applications.

5.2 API Functions

5.2.1 Function List

- void DMAC_Enable(TSB_DMAC_TypeDef * ***DMACx***);
- void DMAC_Disable(TSB_DMAC_TypeDef * ***DMACx***);
- DMAC_INTReq DMAC_GetINTReq(TSB_DMAC_TypeDef * ***DMACx***);
- DMAC_TxINTReq DMAC_GetTxINTReq(TSB_DMAC_TypeDef * ***DMACx***, DMAC_Channel ***Chx***);
- void DMAC_ClearTxINTReq(TSB_DMAC_TypeDef * ***DMACx***, DMAC_Channel ***Chx***, DMAC_INTSrc ***INTSource***);
- DMAC_TxINTReq DMAC_GetRawTxINTReq(TSB_DMAC_TypeDef * ***DMACx***, DMAC_Channel ***Chx***);
- WorkState DMAC_GetChannelTxState(TSB_DMAC_TypeDef * ***DMACx***, DMAC_Channel ***Chx***);
- void DMACA_SetSWBurstReq(DMACA_ReqNum ***BurstReq***);

- DMAC_BurstReqState DMAC_GetSWBurstReqState(TSB_DMAC_TypeDef * ***DMACx***);
- void DMAC_SetLinkedList(TSB_DMAC_TypeDef * ***DMACx,*** DMAC_Channel ***Chx***, uint32_t ***LinkedAddr***);
- WorkState DMAC_GetFIFOState(TSB_DMAC_TypeDef * ***DMACx,*** DMAC_Channel ***Chx***);
- void DMAC_SetDMAHalt(TSB_DMAC_TypeDef * ***DMACx,*** DMAC_Channel ***Chx***, FunctionalState ***NewState***);
- void DMAC_SetLockedTx(TSB_DMAC_TypeDef * ***DMACx,*** DMAC_Channel ***Chx***, FunctionalState ***NewState***)
- void DMAC_SetTxINTConfig(TSB_DMAC_TypeDef * ***DMACx,*** DMAC_Channel ***Chx***, DMAC_INTSrc ***INTSource***, FunctionalState ***NewState***);
- void DMAC_SetDMAChannel(TSB_DMAC_TypeDef * ***DMACx,*** DMAC_Channel ***Chx***, FunctionalState ***NewState***);
- void DMAC_Init(TSB_DMAC_TypeDef * ***DMACx*** , DMAC_Channel ***Chx***, DMAC_InitTypeDef * ***InitStruct***);

5.2.2 Detailed Description

Functions listed above can be divided into five parts:

1) The DMAC basic configure are handled by the DMAC_Enable(),DMAC_Disable(),DMAC_SetDMAChannel() and DMAC_Init() functions.
2) To get DMA transfer interrupt state, FIFO or DMA channel state are handled by DMAC_GetINTReq(),DMAC_GetTxINTReq(),DMAC_GetRawTxINTReq(), DMAC_GetChannelTxState() and DMAC_GetFIFOState().
3) To set DMA interrupt and clear DMA interrupt request are handled by DMAC_ClearTxINTReq() and DMAC_SetTxINTConfig().
4) To set DMA software request and get DMA software request are handled by DMACA_SetSWBurstReq(),DMAC_GetSWBurstReqState(),DMAC_SetLinkedList and DMAC_GetSWSingleReqState().
5) DMAC_SetDMAHalt () and DMAC_SetLockedTx() handle other specified functions.

5.2.3 Function Documentation

5.2.3.1 DMAC_Enable

Enable the DMA circuit.

Prototype:
void
DMAC_Enable(TSB_DMAC_TypeDef * ***DMACx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:

- **DMAC_UNIT_A** for DMA UNIT A,

Description:
This function will Enable UNIT A circuit when ***DMACx*** is **DMAC_UNIT_A**.

Notes:
If use the DMAC module, this function should be called firstly to keeps the DMA circuit operating. Since the registers for the DMA circuit cannot be written or read unless the DMA circuit operates.

Return:
None

5.2.3.2 DMAC_Disable

Disable the DMA circuit.

Prototype:
void
DMAC_Disable(TSB_DMAC_TypeDef * ***DMACx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Description:
This function will disable UNIT A circuit when ***DMACx*** is **DMAC_UNIT_A**.
Return:
None

5.2.3.3 DMAC_GetINTReq

Get DMA Channel interrupt request state.

Prototype:
DMAC_INTReq
DMAC_GetINTReq(TSB_DMAC_TypeDef * ***DMACx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Description:
This function will get UNIT A interrupt request state when ***DMACx*** is **DMAC_UNIT_A** .

Return:
The state of interrupt request.

5.2.3.4 DMAC_GetTxINTReq

Get the specified DMA Channel transfer interrupt request state.

Prototype:
DMAC_TxINTReq
DMAC_GetTxINTReq(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:

This function will get specified UNIT channel 0 transfer interrupt state when ***Chx*** is **DMAC_CHANNEL_0** and get specified UNIT channel 1 transfer interrupt state when ***Chx*** is **DMAC_CHANNEL_1**.

Return:
The request states of DMA transfer interrupt.
The value returned can be one of the followings:
DMAC_TX_NO_REQ means there is no transfer interrupt request,
DMAC_TX_END_REQ means there is a transfer end interrupt request,
DMAC_TX_ERR_REQ means there is a transfer error interrupt request,
DMAC_TX_REQS means there is more than one interrupt request.

5.2.3.5 DMAC_ClearTxINTReq

Clear the transfer interrupt request.

Prototype:
void
DMAC_ClearTxINTReq(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***,
DMAC_INTSrc ***INTSource***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

INTSource: select the release INT source, which can be one of:
- **DMAC_INT_TX_END** for DMA transfer end interrupt,
- **DMAC_INT_TX_ERR** for DMA transfer error interrupt.

Description:
This function will clear the transfer interrupt request. When ***INTSource*** is **DMAC_INT_TX_END**, this function will clear DMA transfer end interrupt request. When ***INTSource*** is **DMAC_INT_TX_ERR**, this function will clear DMA transfer error interrupt request.

Return:
None

5.2.3.6 DMAC_GetRawTxINTReq

Get the specified DMA Channel transfer raw interrupt request state.

Prototype:
DMAC_TxINTReq
DMAC_GetRawTxINTReq(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:
This function will get specified UNIT channel 0 transfer raw interrupt state when ***Chx*** is **DMAC_CHANNEL_0** and get specified UNIT channel 1 transfer raw interrupt state when ***Chx*** is **DMAC_CHANNEL_1**.

Return:
The request states of DMA transfer raw interrupt.
The value returned can be one of the followings:
DMAC_TX_NO_REQ means there is no transfer raw interrupt request,
DMAC_TX_END_REQ means there is a transfer end interrupt request,
DMAC_TX_ERR_REQ means there is a transfer error interrupt request,
DMAC_TX_REQS means there is more than one interrupt request.

5.2.3.7 DMAC_GetChannelTxState

Get the specified DMA Channel transfer state.

Prototype:
WorkState
DMAC_GetChannelTxState(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:
This function will get specified UNIT channel 0 transfer state when ***Chx*** is **DMAC_CHANNEL_0**, and get specified UNIT channel 1 transfer state when ***Chx*** is **DMAC_CHANNEL_1**. If return value is **BUSY**, meaning the DMA channel is enabled and data transmission is in progress. If return value is **DONE**, meaning DMA channel is disabled and data transmission is complete.

Return:
The DMA transfer status.
The value returned can be one of the followings:
BUSY or **DONE**

5.2.3.8 DMACA_SetSWBurstReq

Set DMACA burst transfer requests by software.

Prototype:
void
DMACA_SetSWBurstReq(DMACA_ReqNum ***BurstReq***);

Parameters:
BurstReq: Select burst request number, which can be one of:
- **DMACA_SIO0_UART0_RX** for SIO0/UART0 Reception,
- **DMACA_SIO0_UART0_TX** for SIO0/UART0 Transmission,
- **DMACA_SIO1_UART1_RX** for SIO1/UART1 Reception,
- **DMACA_SIO1_UART1_TX** for SIO1/UART1 Transmission,
- **DMACA_TMRB8_CMP_MATCH** for TMRB8 compare match,
- **DMACA_TMRB9_CMP_MATCH** for TMRB9 compare match,
- **DMACA_TMRB0_CAPTURE0** for TMRB0 input capture 0,
- **DMACA_TMRB4_CAPTURE0** for TMRB4 input capture0,
- **DMACA_TMRB4_CAPTURE1** for TMRB4 input capture1,
- **DMACA_TMRB5_CAPTURE0** for TMRB5 input capture 0,
- **DMACA_TMRB5_CAPTURE1** for TMRB5 input capture 1,
- **DMACA_NORMAL _ADC** for normal A/D Conversion End,
- **DMACA_I2C0_SIO0_RX** for I2C0 Reception,
- **DMACA_I2C0_SIO0_TX** for I2C0 Transmission,
- **DMACA_I2C1_SIO1_RX** for I2C1 Reception
- **DMACA_I2C1_SIO1_TX** for I2C1 Transmission,

Description:
This function will set DMACA burst transfer requests by software. Execute DMACA requests by software and hardware peripheral at the same time is prohibitive.

Return:
None

5.2.3.9 DMAC_GetSWBurstReqState

Get DMA software burst request state.

Prototype:
DMAC_BurstReqState
DMAC_GetSWBurstReqState(TSB_DMAC_TypeDef * ***DMACx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Description:
This function will get specified UNIT software burst request state.

Return:
The DMA burst request status.

5.2.3.10 DMAC_SetLinkedList

Set specified DMA Channel Linked List Item Register.

Prototype:
void
DMAC_SetLinkedList(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***,
uint32_t ***LinkedAddr***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

LinkedAddr: The start address of the next transfer information.
Max 0xFFFFFFF0.

Description:
This function will set specified specified UNIT Channel Linked List Item Register. If scatter/gather function is not required, please call this function with ***LinkedAddr*** set to 0.

Note:
To operate the scatter/gather function, a transfer source and destination data areas need to be defined by creating a set of Linked Lists first.

Each setting is called LLI (LinkedList). Each LLI controls the transfer of one block of data. Each LLI indicates normal DMA setting and controls transfer of successive data. Each time each DMA transfer is complete, the next LLI setting will be loaded to continue the DMA operation (Daisy Chain).

The items that can be set with Linked List are configured with the following 4 words:
1) DMACCxSrcAddr
2) DMACCxDestAddr
3) DMACCxLLI
4) DMACCxControl

Return:
None

5.2.3.11 DMAC_GetFIFOState

Indicates whether data is present in the channel FIFO

Prototype:
WorkState
DMAC_GetFIFOState(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:
This function will get specified UNIT channel FIFO state. If return value is **BUSY**, meaning data exists in the FIFO. If return value is **DONE**, meaning no data exists in the FIFO.

Return:
The FIFO status
The value returned can be one of the followings:
BUSY or **DONE**

5.2.3.12 DMAC_SetDMAHalt

Set whether ignore DMA request.

Prototype:
void
DMAC_SetDMAHalt(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***,
FunctionalState ***NewState***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

NewState: New state of DMA halt.
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will set specified UNIT Channel ignore DMA request.

Return:
None

5.2.3.13 DMAC_SetLockedTx

Set whether locked transfer.

Prototype:
void
DMAC_SetLockedTx(TSB_DMAC_TypeDef * ***DMACx,***
DMAC_Channel ***Chx***,
FunctionalState ***NewState***);

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

NewState: New state of DMA transfer.
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable specified UNIT Channel locked transfer when ***NewState*** is **ENABLE** and disable specified UNIT Channel locked transfer when ***NewState*** is **DISABLE**.

Return:
None

5.2.3.14 DMAC_SetTxINTConfig

Enable or disable the specified DMA Channel transfer interrupt.

Prototype:
```
void
DMAC_SetTxINTConfig(TSB_DMAC_TypeDef * DMACx,
                    DMAC_Channel Chx,
                    DMAC_INTSrc INTSource,
                    FunctionalState NewState);
```

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

INTSource: select the release INT source, which can be one of:
- **DMAC_INT_TX_END** for DMA transfer end interrupt,
- **DMAC_INT_TX_ERR** for DMA transfer error interrupt.

NewState: New states of DMA transfer interrupt.
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable or disable specified UNIT Channel transfer end interrupt when ***INTSource*** is **DMAC_INT_TX_END** and enable or disable specified UNIT Channel transfer error interrupt when ***INTSource*** is **DMAC_INT_TX_ERR**.

Return:
None

5.2.3.15 DMAC_SetDMAChannel

Enable or disable the specified DMA Channel.

Prototype:
```
void
DMAC_SetDMAChannel(TSB_DMAC_TypeDef * DMACx,
                   DMAC_Channel Chx,
                   FunctionalState NewState);
```

Parameters:

DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

NewState: New state of DMA channel.
This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable specified UNIT Channel when ***NewState*** is **ENABLE** and disable specified UNIT Channel when ***NewState*** is **DISABLE**. Please initialize and configure for specified UNIT channel before call this function to enable DMA channel. If use this function directly to disable specified UNIT channel, the data in FIFO will be lost. In order to avoid losing data in FIFO, **DMAC_SetDMAHalt()** should be called to ignore specified UNIT request, then call **DMAC_GetFIFOState()** to get the state of FIFO, last call this function to disable specified UNIT channel.

Return:
None

5.2.3.16 DMAC_Init

Initialize and configure the specified DMA channel.

Prototype:
```
void
DMAC_Init(TSB_DMAC_TypeDef * DMACx,
          DMAC_Channel Chx,
          DMAC_InitTypeDef * InitStruct);
```

Parameters:
DMACx is the specified DMA Unit, which can be one of:
- **DMAC_UNIT_A** for DMA UNIT A,

Chx is the specified DMA channel, which can be one of:
- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

InitStruct is the structure containing basic DMA configuration including source address, source address incremented state, transfer source bit width, transfer source burst size, destination address, destination address incremented state, transfer destination bit width, transfer destination burst size, transfer size, transfer direction, transfer peripheral and transfer interrupt state. (Refer to “Data Structure Description” for details).

Description:
This function will initialize and configure the source address, source address incremented state, transfer source bit width, transfer source burst size, destination address, destination address incremented state, transfer destination bit width, transfer destination burst size, transfer size, and transfer direction, transfer peripheral and transfer interrupt state for the specified UNIT channel.

Note:
Please use this API to initialize DMAC transmission before calling **DMAC_SetDMAChannel()**.

Return:
None

5.2.4 Data Structure Description

5.2.4.1 DMAC_InitTypeDef

Data Fields:

uint32_t
TxDirection Set transfer direction, which can be set as:
- **DMAC_MEMORY_TO_MEMORY**: transfer method is memory to memory,
- **DMAC_MEMORY_TO_PERIPH**: transfer method is memory to peripheral,
- **DMAC_PERIPH_TO_MEMORY**: transfer method is peripheral to memory.
- **DMAC_PERIPH_TO_ PERIPH**: transfer method is peripheral to peripheral.

uint32_t
SrcAddr Set source address.

uint32_t
DstAddr Set destination address.

FunctionalState
SrcIncrementState Specifies whether the source address is incremented or not, which can be set as:
ENABLE or **DISABLE**.

FunctionalState
DstIncrementState Specifies whether the destination address is incremented or not, which can be set as:
ENABLE or **DISABLE**.

DMAC_BitWidth
SrcBitWidth Set transfer source bit width, which can be set as:
- **DMAC_BYTE** means transfer source bit width set as byte,
- **DMAC_HALF_WORD** means transfer source bit width set as half word,
- **DMAC_WORD** means transfer source bit width set as word.

DMAC_BitWidth
DstBitWidth Set transfer destination bit width, which can be set as:
- **DMAC_BYTE** means transfer destination bit width set as byte,
- **DMAC_HALF_WORD** means transfer destination bit width set as half word,
- **DMAC_WORD** means transfer destination bit width set as word.

DMAC_BurstSize
SrcBurstSize Set transfer source burst size, which can be set as:
- **DMAC_1_BEAT** means transfer source burst size set as 1 beat,
- **DMAC_4_BEATS** means transfer source burst size set as 4 beats,
- **DMAC_8_BEATS** means transfer source burst size set as 8 beats,
- **DMAC_16_BEATS** means transfer source burst size set as 16 beats,
- **DMAC_32_BEATS** means transfer source burst size set as 32 beats,

- **DMAC_64_BEATS** means transfer source burst size set as 64 beats,
- **DMAC_128_BEATS** means transfer source burst size set as 128 beats,
- **DMAC_256_BEATS** means transfer source burst size set as 256 beats.

DMAC_BurstSize
DstBurstSize Set transfer destination burst size, which can be set as:

- **DMAC_1_BEAT** means transfer destination burst size set as 1 beat,
- **DMAC_4_BEATS** means transfer destination burst size set as 4 beats,
- **DMAC_8_BEATS** means transfer destination burst size set as 8 beats,
- **DMAC_16_BEATS** means transfer destination burst size set as 16 beats,
- **DMAC_32_BEATS** means transfer destination burst size set as 32 beats,
- **DMAC_64_BEATS** means transfer destination burst size set as 64 beats,
- **DMAC_128_BEATS** means transfer destination burst size set as 128 beats,
- **DMAC_256_BEATS** means transfer destination burst size set as 256 beats.

uint32_t
TxSize Set the total number of transfer, MAX is 0x0FFF.

DMACA_ReqNum
A_TxDstPeriph Set transfer destination peripheral, which can be set as:

- **DMACA_SIO0_UART0_RX** for SIO0/UART0 Reception,
- **DMACA_SIO0_UART0_TX** for SIO0/UART0 Transmission,
- **DMACA_SIO1_UART1_RX** for SIO1/UART1 Reception,
- **DMACA_SIO1_UART1_TX** for SIO1/UART1 Transmission,
- **DMACA_TMRB8_CMP_MATCH** for TMRB8 compare match,
- **DMACA_TMRB9_CMP_MATCH** for TMRB9 compare match,
- **DMACA_TMRB0_CAPTURE0** for TMRB0 input capture 0,
- **DMACA_TMRB4_CAPTURE0** for TMRB4 input capture0,
- **DMACA_TMRB4_CAPTURE1** for TMRB4 input capture1,
- **DMACA_TMRB5_CAPTURE0** for TMRB5 input capture 0,
- **DMACA_TMRB5_CAPTURE1** for TMRB5 input capture 1,
- **DMACA_NORMAL _ADC** for normal A/D Conversion End,
- **DMACA_I2C0_SIO0_RX** for I2C0 Reception,
- **DMACA_I2C0_SIO0_TX** for I2C0 Transmission,
- **DMACA_I2C1_SIO1_RX** for I2C1 Reception,
- **DMACA_I2C1_SIO1_TX** for I2C1Transmission,

DMACA_ReqNum
A_TxSrcPeriph Set transfer source peripheral, which can be set as:

- **DMACA_SIO0_UART0_RX** for SIO0/UART0 Reception,
- **DMACA_SIO0_UART0_TX** for SIO0/UART0 Transmission,
- **DMACA_SIO1_UART1_RX** for SIO1/UART1 Reception,
- **DMACA_SIO1_UART1_TX** for SIO1/UART1 Transmission,
- **DMACA_TMRB8_CMP_MATCH** for TMRB8 compare match,
- **DMACA_TMRB9_CMP_MATCH** for TMRB9 compare match,
- **DMACA_TMRB0_CAPTURE0** for TMRB0 input capture 0,
- **DMACA_TMRB4_CAPTURE0** for TMRB4 input capture0,
- **DMACA_TMRB4_CAPTURE1** for TMRB4 input capture1,
- **DMACA_TMRB5_CAPTURE0** for TMRB5 input capture 0,
- **DMACA_TMRB5_CAPTURE1** for TMRB5 input capture 1,
- **DMACA_NORMAL _ADC** for normal A/D Conversion End,
- **DMACA_I2C0_SIO0_RX** for I2C0 Reception,
- **DMACA_I2C0_SIO0_TX** for I2C0 Transmission,
- **DMACA_I2C1_SIO1_RX** for I2C1 Reception,
- **DMACA_I2C1_SIO1_TX** for I2C1Transmission,

FunctionalState
TxINT Set transfer interrupt state, which can be set as:

- **EANBLE** means enable transfer interrupt.
- **DISABLE** means disable transfer interrupt.

6. FC

6.1 Overview

TMPM365 device contains flash memory; the size of flash is 256Kbyte.

In on-board programming, the CPU is to execute software commands for rewriting or erasing the flash memory. Writing and erasing flash memory data are in accordance with the standard JEDEC commands. Besides it also provides the registers that are used to monitor the status of the flash memory and to indicate the protection status of each block, and activate security function.

The Block Configuration of Flash Memory, please refer to the MCU data sheet.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm365_fc.c with \Libraries\TX03_Periph_Driver \inc\tmpm365_fc.h containing the API definitions for use by applications.

6.2 API Functions

6.2.1 Function List

- void FC_SetSecurityBit(FunctionalState ***NewState***)
- FunctionalState FC_GetSecurityBit(void)
- WorkState FC_GetBusyState(void)
- FunctionalState FC_GetBlockProtectState(uint8_t ***BlockNum***)
- FC_Result FC_ProgramBlockProtectState(uint8_t ***BlockNum***)
- FC_Result FC_EraseBlockProtectState(uint8_t ***BlockGroup***)
- FC_Result FC_WritePage(uint32_t ***PageAddr***, uint32_t * ***Data***)
- FC_Result FC_EraseBlock(uint32_t ***BlockAddr***)
- FC_Result FC_EraseChip(void)

6.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) The security function restricts flash ROM data readout and debugging.
   FC_SetSecurityBit(), FC_GetSecurityBit().
2) The functions get the automatic operation status and each block protection status:
   FC_GetBusyState(), FC_GetBlockProtectState(),
3) The functions change the protection status of each block:
   FC_ProgramBlockProtectState(), FC_EraseBlockProtectState().
4) Use automatic operation command to write or erase the content of flash.
   FC_WritePage(), FC_EraseBlock(), FC_EraseChip().

6.2.3 Function Documentation

6.2.3.1 FC_SetSecurityBit

Set the value of SECBIT register.

Prototype:
void
FC_SetSecurityBit (FunctionalState ***NewState***)

Parameters:
NewState: Select the state of SECBIT register.
This parameter can be one of the following values:
- **DISABLE**: Protection function is not available.
- **ENABLE**: Protection function is available.

Description:
1) All the protection bits (the FLCS<BLPRO> bits) used for the write/erase-protection function are set to “1”.
2) The SECBIT <SECBIT> bit is set to”1”.

Only when the two conditions above are met at the same time, the security function that restricts flash ROM Data readout and debugging will be available. At this time, communication of JTAG/SW is prohibited, it means you can not use JTAG to debug, so please be careful when you want to use this API to set SECBIT<SEBIT> to “1”.

The SECBIT <SECBIT> bit is set to “1” at a power-on reset right after power-on.

Return:
None

6.2.3.2 FC_GetSecurityBit

Get the value of SECBIT register.

Prototype:
FunctionalState
FC_GetSecurityBit(void)

Parameters:
None

Description:
This API is used to get the state of the SECBIT register. If the value of SECBIT <SECBIT> bit is”1”, it returns **ENABLE**. If the value of SECBIT <SECBIT> bit is”0”, it returns **DISABLE**.

Return:
State of SECBIT register.
DISABLE: Protection function is not available.
ENABLE: Protection function is available.

6.2.3.3 FC_GetBusyState

Get the status of the flash auto operation.

Prototype:
WorkState
FC_GetBusyState (void)

Parameters:
None

Description:
When the flash memory is in automatic operation, it outputs "0" to indicate that it is busy. When the automatic operation is normally terminated, it returns to the ready state and outputs "1" to accept the next command.

Return:
Status of the flash automatic operation:
BUSY: Flash memory is in automatic operation.
DONE: Automatic operation is normally terminated. The next command can be sent and executed.

6.2.3.4 FC_GetBlockProtectState

Get the block protection status.

Prototype:
FunctionalState
FC_GetBlockProtectState(uint8_t ***BlockNum***)

Parameters:
BlockNum:The flash block number
- **FC_BLOCK_0** for block 0.
- **FC_BLOCK_1** for block 1.
- **FC_BLOCK_2** for block 2.
- **FC_BLOCK_3** for block 3.
- **FC_BLOCK_4** for block 4.
- **FC_BLOCK_5** for block 5.

Description:
Each protection bit represents the protection status of the corresponding block. When a bit is set to "1", it indicates that the block corresponding to the bit is protected. When the block is protected, it can't be written or erased. About the block configuration of the flash memory, please refer to overview.

Return:
Block protection status
DISABLE: Block is unprotected
ENABLE: Block is protected

6.2.3.5 FC_ProgramBlockProtectState

Program the protection bits

Prototype:
FC_Result
FC_ProgramProtectState(uint8_t ***BlockNum***)

Parameters:
BlockNum:The flash block number
- **FC_BLOCK_0** for block 0.
- **FC_BLOCK_1** for block 1.
- **FC_BLOCK_2** for block 2.
- **FC_BLOCK_3** for block 3.

- **FC_BLOCK_4** for block 4.
- **FC_BLOCK_5** for block 5.

Description:
This API is used to set the protection bit to “1” so that the corresponding block can be protected. When the block is protected, it can’t be written or erased.
One protection bit will be programmed when this API is executed each time.

Return:
Result of the operation to program the protection bit
FC_SUCCESS: Set the protection bit to “1” successfully.
FC_ERROR_PROTECTED: The protection bit is “1” already, and it doesn’t need to program it again.
FC_ERROR_OVER_TIME: Program block protection bit operation over time error.

6.2.3.6 FC_EraseBlockProtectState

Erase the protection bits

Prototype:
FC_Result
FC_EraseBlockProtectState(uint8_t ***BlockGroup***)

Parameters:
BlockGroup:The flash block group
- **FC_BLOCK_GROUP_1** for block 4, block 5
- **FC_BLOCK_GROUP_0** for the other blocks

Description:
This API is used to erase the protection bits (clear them to”0”) so that the corresponding blocks will not be protected.
One group of protection bits will be erased when this API is executed each time.

Return:
Result of the operation to erase the protection bits
FC_SUCCESS: Erase the protection bits successfully.
FC_ERROR_OVER_TIME: Erase block protection bits operation over time error.

6.2.3.7 FC_WritePage

Write data to the specified page

Prototype:
FC_Result
FC_WritePage(uint32_t ***PageAddr***, uint32_t * ***Data***)

Parameters:
PageAddr:The page start address
Data: The pointer to data buffer to be written into the page. The data size should be 256Byte.

Description:
This API is used to write data to specified page.
The TMPM365 contains 64 words in a page. The flash can only be written page by page.

The automatic page programming is allowed only once for a page already erased. No programming can be performed twice or more time irrespective of data value whether it is “1” or “0”.
***Note**: An attempt to rewrite a page two or more times without erasing the content can cause damages to the device.

Return:
Result of the operation to write data to the specified page.
FC_SUCCESS: data is written to the specified page accurately.
FC_ERROR_PROTECTED: The block is protected. The write operation can’t be executed.
FC_ERROR_OVER_TIME: Write operation over time error.

6.2.3.8 FC_EraseBlock

Erase the content of specified block.

Prototype:
FC_Result
FC_EraseBlock(uint32_t ***BlockAddr***)

Parameters:
BlockAddr: The block starts address.

Description:
This API is used to erase the content of specified block. Only unprotected blocks will be erased.

Return:
Result of the operation to erase the content of specified block.
FC_SUCCESS: the content of the specified block is erased successfully.
FC_ERROR_PROTECTED: The block is protected. The erase operation can’t be executed. The block will not be erased.
FC_ERROR_OVER_TIME: Erase operation over time error.

6.2.3.9 FC_EraseChip

Erase the content of the entire chip.

Prototype:
FC_Result
FC_EraseChip(void)

Parameters:
None

Description:
This API is used to erase the content of the entire chip. If all the blocks are unprotected, the entire chip will be erased. If parts of blocks are protected, only unprotected blocks will be erased.

Return:
Result of the operation to erase the content of the entire chip.
FC_SUCCESS: If all the blocks are unprotected, the entire chip is erased. If parts of blocks are protected, only unprotected blocks are erased
FC_ERROR_PROTECTED: All blocks are protected. The erase chip operation can’t be executed.
FC_ERROR_OVER_TIME: Erase Chip operation over time error.

6.2.4 Data Structure Description

None.

7. GPIO

7.1 Overview

For TOSHIBA TMPM365 general-purpose I/O ports, inputs and outputs can be specified in units of bits. Besides the general-purpose input/output function, all ports perform specified function.

The GPIO driver APIs provide a set of functions to configure each port, including such common parameters as input, output, pull-up, pull-down, open-drain, CMOS and so on.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm365_gpio.c , with /Libraries/TX03_Periph_Driver/inc/tmpm365_gpio.h containing the macros, data types, structures and API definitions for use by applications.

7.2 API Functions

7.2.1 Function List

- uint8_t GPIO_ReadData(GPIO_Port ***GPIO_x***);
- uint8_t GPIO_ReadDataBit(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***) ;
- void GPIO_WriteData(GPIO_Port ***GPIO_x***, uint8_t ***Data***) ;
- void GPIO_WriteDataBit(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***, uint8_t ***BitValue***) ;
- void GPIO_Init(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  GPIO_InitTypeDef **** GPIO_InitStruct***);
- void GPIO_SetOutput(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***);
- void GPIO_SetInput(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***);
- void GPIO_SetOutputEnableReg(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***);
- void GPIO_SetInputEnableReg(GPIO_Port **GPIO_x**, uint8_t **Bit_x**,
  FunctionalState ***NewState***);
- void GPIO_SetPullUp(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***);
- void GPIO_SetPullDown(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***);
- void GPIO_SetOpenDrain(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***);
- void GPIO_EnableFuncReg(GPIO_Port ***GPIO_x***, uint8_t ***FuncReg_x***, uint8_t ***Bit_x***);
- void GPIO_DisableFuncReg(GPIO_Port ***GPIO_x***, uint8_t ***FuncReg_x***, uint8_t ***Bit_x***);

7.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) Write/Read GPIO or GPIO pin are handled by GPIO_ReadData(), GPIO_ReadDataBit(), GPIO_WriteData() and GPIO_WriteDataBit().
2) Initialize and configure the common functions of each GPIO port are handled by GPIO_SetOutput(), GPIO_SetInput(),GPIO_SetOutputEnableReg(),

GPIO_SetInputEnableReg(), GPIO_SetPullUp(),GPIO_SetPullDown(), GPIO_SetOpenDrain() and GPIO_Init().

3) GPIO_EnableFuncReg() and GPIO_DisableFuncReg() handle other specified functions.

7.2.3 Function Documentation

7.2.3.1 GPIO_ReadData

Read specified GPIO Data register.

Prototype:
uint8_t
GPIO_ReadData(GPIO_Port ***GPIO_x***);

Parameters:
GPIO_x: Select GPIO port, which can be set as:

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Description:
This function will read specified GPIO Data register.

Return:
The value read from DATA register.

7.2.3.2 GPIO_ReadDataBit

Read specified GPIO pin.

Prototype:
uint8_t
GPIO_ReadDataBit(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***);

Parameters:
GPIO_x: Select GPIO port, which can be set as:

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D..
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7.

Description:
This function will read specified GPIO pin.

Return:
The value read from GPIO pin as:
- **GPIO_BIT_VALUE_0**: Value 0,
- **GPIO_BIT_VALUE_1**: Value 1.

7.2.3.3 GPIO_WriteData

Write specified value to GPIO Data register.

Prototype:
```
void
GPIO_WriteData(GPIO_Port  GPIO_x,
                    uint8_t Data);
```

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Data: The value will be written to GPIO DATA register.

Description:
This function will write new value to specified GPIO Data register.

Return:
None

7.2.3.4 GPIO_WriteDataBit

Write specified value of single bit to GPIO pin.

Prototype:

void
GPIO_WriteDataBit(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***,
uint8_t ***BitValue***) ;

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J. .
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7.

BitValue: The new value of GPIO pin, which can be set as:
- **GPIO_BIT_VALUE_0**: Clear GPIO pin,
- **GPIO_BIT_VALUE_1**: Set GPIO pin.

Description:
This function will write new bit value to specified GPIO pin.

Return:
None

7.2.3.5 GPIO_Init

Initialize GPIO port function.

Prototype:
void
GPIO_Init(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***,
GPIO_InitTypeDef * ***GPIO_InitStruct***);

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.

- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

GPIO_InitStruct: The structure containing basic GPIO configuration. (Refer to Data structure Description for details)

Description:
This function will be configure GPIO pin IO mode, pull-up, pull-down function and set this pin as open drain port or CMOS port. **GPIO_SetOutput()**, **GPIO_SetInput()**,**GPIO_SetPullUp ()**,**GPIO_SetPullDown()** and **GPIO_SetOpenDrain()** will be called by it.

Return:
None

7.2.3.6 GPIO_SetOutput

Set specified GPIO pin as output port.

Prototype:
```
void
GPIO_SetOutput(GPIO_Port  GPIO_x,
                    uint8_t Bit_x);
```

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,

- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

Description:
This function will set specified GPIO pin as output port.

Return:
None

7.2.3.7 GPIO_SetInput

Set specified GPIO Pin as input port.

Prototype:
void
GPIO_SetInput(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***);

Parameters:
GPIO_x: Select GPIO port, which can be set as:

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:

- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

Description:
This function will set specified GPIO pin as input port.

Return:
None

7.2.3.8 GPIO_SetOutputEnableReg

Enable or disable specified GPIO Pin output function.

Prototype:

```
void
GPIO_SetOutputEnableReg(GPIO_Port GPIO_x,
                        uint8_t Bit_x,
                        FunctionalState NewState);
```

Parameters:

GPIO_x: Select GPIO port, which can be set as:

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:

- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

NewState:

- **ENABLE :** Enable output state
- **DISABLE :** Disable output state

Description:

This function will enable output function for the specified GPIO pin when ***NewState*** is **ENABLE**, and disable specified GPIO pin output function when ***NewState*** is **DISABLE**.

Return:

None

7.2.3.9 GPIO_SetInputEnableReg

Enable or disable specified GPIO Pin input function.

Prototype:

```
void
GPIO_SetInputEnableReg(GPIO_Port GPIO_x,
                       uint8_t Bit_x,
```

FunctionalState ***NewState***);

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

NewState:
- **ENABLE :** Enable input state
- **DISABLE :** Disable input state

Description:
This function will enable input function for the specified GPIO pin when ***NewState*** is **ENABLE**, and disable specified GPIO pin input function when ***NewState*** is **DISABLE**.

Return:
None

7.2.3.10 GPIO_SetPullUp

Enable or disable specified GPIO Pin pull-up function.

Prototype:
void
GPIO_SetPullUp(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***,
FunctionalState ***NewState***);

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.

- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

NewState:
- **ENABLE :** Enable pullup state
- **DISABLE :** Disable pullup state

Description:
This function will enable pull-up function for the specified GPIO pin when ***NewState*** is **ENABLE**, and disable specified GPIO pin has pull-up function when ***NewState*** is **DISABLE**.

Return:
None

7.2.3.11 GPIO_SetPullDown

Enable or disable specified GPIO Pin pull-down function.

Prototype:
```
void
GPIO_SetPullDown(GPIO_Port GPIO_x,
                 uint8_t Bit_x,
                 FunctionalState NewState);
```

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PI :** GPIO port I.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,

- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

NewState:
- **ENABLE :** Enable pulldown state
- **DISABLE :** Disable pulldown state

Description:
This function will enable pull-down function for the specified GPIO pin when ***NewState*** is **ENABLE**, and disable specified GPIO pin has pull-down function when ***NewState*** is **DISABLE**.

Return:
None

7.2.3.12 GPIO_SetOpenDrain

Set specified GPIO Pin as open drain port or CMOS port.

Prototype:
```
void
GPIO_SetOpenDrain(GPIO_Port GPIO_x,
                  uint8_t Bit_x,
                  FunctionalState NewState);
```

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7,
- **GPIO_BIT_ALL:** All GPIO pins can be set.
- Combination of the effective bits.

NewState:
- **ENABLE :** enable open drain state
- **DISABLE :** disable open drain state

Description:

This function will set specified GPIO pin as open-drain port when ***NewState*** is **ENABLE**, and set specified GPIO pin as CMOS port when ***NewState*** is **DISABLE**.

Return:
None

7.2.3.13 GPIO_EnableFuncReg

Enable specified GPIO function.

Prototype:
void
GPIO_EnableFuncReg(GPIO_Port ***GPIO_x***,
uint8_t ***FuncReg_x***,
uint8_t ***Bit_x***) ;

Parameters:

GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

FuncReg_x: The number of GPIO function register, which can be set as:
- **GPIO_FUNC_REG_1** for GPIO function register 1,
- **GPIO_FUNC_REG_2** for GPIO function register 2,
- **GPIO_FUNC_REG_3** for GPIO function register 3.
- **GPIO_FUNC_REG_4** for GPIO function register 4.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7.
- Combination of the effective bits.

Description:
This function will enable GPIO pin specified function.

Return:
None

7.2.3.14 GPIO_DisableFuncReg

Disable specified GPIO function.

Prototype:

void
GPIO_DisableFuncReg(GPIO_Port ***GPIO_x***,
uint8_t ***FuncReg_x***,
uint8_t ***Bit_x***) ;

Parameters:

GPIO_x: Select GPIO port, which can be set as:

- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PH:** GPIO port H.
- **GPIO_PI :** GPIO port I.
- **GPIO_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.

FuncReg_x: The number of GPIO function register, which can be set as:

- **GPIO_FUNC_REG_1** for GPIO function register 1,
- **GPIO_FUNC_REG_2** for GPIO function register 2,
- **GPIO_FUNC_REG_3** for GPIO function register 3.
- **GPIO_FUNC_REG_4** for GPIO function register 4.

Bit_x: Select GPIO pin, which can be set as:

- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,
- **GPIO_BIT_4:** GPIO pin 4,
- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_7:** GPIO pin 7.
- Combination of the effective bits.

Description:

This function will disable GPIO pin specified function.

Return:

None

7.2.4 Data Structure Description

7.2.4.1 GPIO_InitTypeDef

Data Fields:

uint8_t
IOMode Set specified GPIO Pin as input port or output port, which can be set as:

- **GPIO_INPUT:** Set GPIO pin as input port
- **GPIO_OUTPUT:** Set GPIO pin as output port

- **GPIO_IO_MODE_NONE:** Don't change GPIO pin I/O mode.

uint8_t
PullUp Enable or disable specified GPIO Pin pull-up function, which can be set as:

- **GPIO_PULLUP_ENABLE :** Enable specified GPIO pin pull-up function**.**
- **GPIO_PULLUP_DISABLE:** Disable specified GPIO pin pull-up function**.**
- **GPIO_PULLUP_NONE:** Don't have pull-up function or needn't change.

uint8_t
OpenDrain Set specified GPIO Pin as open drain port or CMOS port, which can be set as:

- **GPIO_OPEN_DRAIN_ENABLE:** Set specified GPIO pin as open drain port**.**
- **GPIO_OPEN_DRAIN_DISABLE:** Set specified GPIO pin as CMOS port**.**
- **GPIO_OPEN_DRAIN_NONE:** Don't have open-drain function or needn't change.

uint8_t
PullDown Enable or disable specified GPIO Pin pull-down function, which can be set as:

- **GPIO_PULLDOWN_ENABLE:** Enable specified GPIO pin pull-down function**.**
- **GPIO_PULLDOWN_DISABLE:** Disable specified GPIO pin pull-down function**.**
- **GPIO_PULLDOWN_NONE:** Don't have pull-down function or needn't change.

8. SBI

8.1 Overview

This device contains some Serial Bus Interface channels. Each channel can operate in I2C bus mode with multi-master capability.
In I2C bus mode, the SBI is connected to external devices via SCL and SDA.
Data can be transferred in free data format by the SBI channels. In free data format, data is always sent by master-transmitter and received by slave-receiver.

The SBI driver APIs provide a set of functions to configure each channel such as setting self-address of the SBI channel, the clock division, the generation of ACK clock and to control the data transfer such as sending start condition or stop condition to I2C bus, data transmission or reception, and to indicate the status of each channel such as returning the state or the mode of each SBI channel.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm365_sbi.c, with /Libraries/TX03_Periph_Driver/inc/tmpm365_sbi.h containing the macros, data types, structures and API definitions for use by applications.

8.2 API Functions

8.2.1 Function List

- void SBI_Enable(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_Disable(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_SetI2CACK(TSB_SBI_TypeDef* ***SBIx***, FunctionalState ***NewState***);
- void SBI_InitI2C(TSB_SBI_TypeDef* ***SBIx***, SBI_InitI2CTypeDef* ***InitI2CStruct***);
- void SBI_SetI2CBitNum(TSB_SBI_TypeDef* ***SBIx***, uint32_t ***I2CBitNum***);
- void SBI_SWReset(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_ClearI2CINTReq(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_GenerateI2CStart(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_GenerateI2CStop(TSB_SBI_TypeDef* ***SBIx***);
- SBI_I2CState SBI_GetI2CState(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_SetIdleMode(TSB_SBI_TypeDef* ***SBIx***, FunctionalState ***NewState***);
- void SBI_SetSendData(TSB_SBI_TypeDef* ***SBIx***, uint32_t ***Data***);
- uint32_t SBI_GetReceiveData(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_SetI2CFreeDataMode(TSB_SBI_TypeDef* ***SBIx,*** FunctionalState ***NewState***);

8.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Configure and control the common functions of each SBI channel are handled by SBI_Enable(), SBI_Disable(), SBI_SetI2CACK(), SBI_SetI2CBitNum(), and SBI_InitI2C().
2) Transfer control of each TMRB channel is handled by SBI_ClearI2CINTReq(), SBI_GenerateI2Cstart(), SBI_GenerateI2Cstop(), SBI_IsI2ClastRxBitSet(), SBI_GetReceiveData().
3) The status indication of each SBI channel is handled by SBI_GetI2CState().
4) SBI_SWReset(), SBI_SetIdleMode() and SBI_EnableI2CfreeDataMode() handle other specified functions.

8.2.3 Function Documentation

Note: in all of the following APIs, parameter "TSB_SBI_TypeDef* ***SBIx***" can be one of the following values:
TSB_SBI0, TSB_SBI1.

8.2.3.1 SBI_Enable

Enable the specified SBI channel.

Prototype:
void
SBI_Enable(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
This function will enable the specified SBI channel selected by ***SBIx***.

Return:
None

8.2.3.2 SBI_Disable

Disable the specified SBI channel.

Prototype:
void
SBI_Disable(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
This function will disable the specified SBI channel selected by ***SBIx***.

Return:
None

8.2.3.3 SBI_SetI2CACK

Enable or disable the generation of ACK clock.

Prototype:
void
SBI_SetI2CACK(TSB_SBI_TypeDef* ***SBIx***,
FunctionalState ***NewState***);

Parameters:
SBIx is the specified SBI channel.
NewState sets the generation of ACK clock, which can be:

- **ENABLE** for generating of ACK clock

- **DISABLE** for no ACK clock

Description:
The function specifies the generation of ACK clock on I2C bus. The ACK clock will be generated if ***NewState*** is **ENABLE**. And the ACK clock will be not generated if ***NewState*** is **DISABLE**.

Return:
None

8.2.3.4 SBI_InitI2C

Initialize the specified SBI channel in I2C mode.

Prototype:
void
SBI_InitI2C(TSB_SBI_TypeDef* ***SBIx***,
SBI_InitI2CTypeDef* ***InitI2CStruct***);

Parameters:
SBIx is the specified SBI channel.
InitI2CStruct is the structure containing SBI configuration (refer to 10.2.4 Data Structure Description for details).

Description:
This function will initialize and configure the self-address, bit length of transfer data, clock division, the generation of ACK clock and the operation mode of I2C transfer for the specified SBI channel selected by ***SBIx***.

Return:
None

8.2.3.5 SBI_SetI2CBitNum

Specify the number of bits per transfer.

Prototype:
void
SBI_SetI2CBitNum(TSB_SBI_TypeDef* ***SBIx***,
uint32_t ***I2CBitNum***);

Parameters:
SBIx is the specified SBI channel.
I2CBitNum specifies the number of bits per transfer, max. 8.
This parameter can be one of the following values:

- **SBI_I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **SBI_I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;
- **SBI_I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;
- **SBI_I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **SBI_I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;

- **SBI_I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **SBI_I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;
- **SBI_I2C_DATA_LEN_7**, which means that the data length number of bits per transfer is 7.

Description:
The number of bits to be transferred each transaction can be changed by this function.

Return:
None

8.2.3.6 SBI_SWReset

Reset the state of the specified SBI channel.

Prototype:
void
SBI_SWReset(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
This function will generate a reset signal that initializes the serial bus interface circuit. After a reset, all control registers and status flags are initialized to their reset values.

Return:
None

8.2.3.7 SBI_ClearI2CINTReq

Clear SBI interrupt request in I2C bus mode.

Prototype:
void
SBI_ClearI2CINTReq(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
This function will clear the SBI interrupt, which has occurred, of the specified SBI channel.

Return:
None

8.2.3.8 SBI_GenerateI2CStart

Set I2c bus to Master mode and Generate start condition in I2C mod.

Prototype:
void
SBI_GenerateI2CStart(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
The function will set I2c bus to Master mode and send start condition on I2C bus.

Return:
None

8.2.3.9 SBI_GenerateI2CStop

Set I2c bus to Master mode and Generate stop condition in I2C mode.

Prototype:
void
SBI_GenerateI2CStop(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
The function will set I2c bus to Master mode and send stop condition on I2C bus.

Return:
None

8.2.3.10 SBI_GetI2CState

Get the SBI channel state in I2C bus mode.

Prototype:
SBI_I2CState
SBI_GetI2CState(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
This function can return the state of the SBI channel while it is working in I2C bus mode. Call the function in ISR of SBI interrupt, and adopt different process according to different return.

Return:
The state value of the SBI channel in I2C bus.

8.2.3.11 SBI_SetIdleMode

Enable or disable the specified SBI channel when system is in idle mode.

Prototype:
void
SBI_SetIdleMode(TSB_SBI_TypeDef* ***SBIx***,
FunctionalState ***NewState***);

Parameters:
SBIx is the specified SBI channel.
NewState specifies the state of the SBI when system is idle mode, which can be
- **ENABLE:** enables the SBI channel.
- **DISABLE:** disables the SBI channel.

Description:
The specified SBI channel can still working if ***NewState*** is **ENABLE** even if system enters idle mode. **DISABLE** can stop the working SBI if system enters idle mode.

Return:
None

8.2.3.12 SBI_SetSendData

Set data to be sent and start transmitting from the specified SBI channel.

Prototype:
void
SBI_SetSendData(TSB_SBI_TypeDef* ***SBIx***,
uint32_t ***Data***);

Parameters:
SBIx is the specified SBI channel.
Data is a byte-data to be sent. The maximum value is 0xFF.

Description:
This function will set the data to be sent from the specified SBI channel selected by ***SBIx***. It is appropriate to call the function after the transmission of the start condition, which can be done by **SBI_GenerateI2Cstart()**, or the reception of an ACK (usually causes an SBI interrupt), to send further data required by receiver.

Return:
None

8.2.3.13 SBI_GetReceiveData

Get data received from the specified SBI channel.

Prototype:
uint32_t
SBI_GetReceiveData(TSB_SBI_TypeDef* ***SBIx***);

Parameters:
SBIx is the specified SBI channel.

Description:
This function will set the data to be sent from the specified SBI channel selected by ***SBIx***. It is appropriate to call the function after the transmission of the start condition, which can be done by **SBI_GenerateI2Cstart()**, or the reception of an ACK (usually causes an SBI interrupt), to send further data required by receiver.

Return:
Data which has been received

8.2.3.14 SBI_SetI2CFreeDataMode

Set SBI channel working in I2C free data mode.

Prototype:
void
SBI_setI2CFreeDataMode(TSB_SBI_TypeDef* ***SBIx***,
FunctionalState ***NewState***);

Parameters:
SBIx is the specified SBI channel.
NewState specifies the state of the SBI when system is idle mode, which can be
- **ENABLE:** enables the SBI channel.
- **DISABLE:** disables the SBI channel.

Description:
The specified SBI channel can transfer data in free data format by calling this function. In free data format, master device always transmits data while slave device always receives data. If the SBI is needed to shift to transfer data in normal I2C format, call **SBI_InitI2C()**.

Return:
None

8.2.4 Data Structure Description

8.2.4.1 SBI_InitI2CTypeDef

Data Fields:

uint32_t
I2CSelfAddr specifies self-address of the SBI channel in I2C mode, the last bit of which can not be 1 and max. 0xFE.

uint32_t
I2CDataLen Specify data length of the SBI channel in I2C mode, which can be set as:
- **SBI_I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **SBI_I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;
- **SBI_I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;

- **SBI_I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **SBI_I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;
- **SBI_I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **SBI_I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;
- **SBI_I2C_DATA_LEN_7**, which means that the data length number of bits per transfer is 7.

uint32_t
I2CClkDiv specifies the division of the source clock for I2C transfer, which can be set as:

- **SBI_I2C_CLK_DIV_104**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 104;
- **SBI_I2C_CLK_DIV_136**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 136;
- **SBI_I2C_CLK_DIV_200**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 200;
- **SBI_I2C_CLK_DIV_328**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 328;
- **SBI_I2C_CLK_DIV_584**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 584;
- **SBI_I2C_CLK_DIV_1096**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 1096;
- **SBI_I2C_CLK_DIV_2120**, which means that the frequency of source clock for I2C transfer is quotient of fsys divided by 2120.

FunctionalState
I2CACKState Enable or disable the generation of ACK clock, which can be one of the following values:

- **ENABLE:** enables the generation of ACK clock.
- **DISABLE:** disables the generation of ACK clock.

8.2.4.2 SBI_I2CState

Data Fields:

uint32_t
All specifies state data in I2C mode

Bit Fields:

uint32_t
LastRxBit specifies last received bit monitor.

uint32_t
GeneralCall specifies general call detected monitor.

uint32_t
SlaveAddrMatch specifies slave address match monitor.

uint32_t
ArbitrationLost specifies arbitration last detected monitor.

uint32_t
INTReq specifies Interrupt request monitor.

uint32_t
BusState specifies bus busy flag.

uint32_t
TRx specifies transfer or Receive selection monitor.

uint32_t
MasterSlave specifies master or slave selection monitor.

9. TMRB

9.1 Overview

TOSHIBA TMPM365 contains 10 channels of a 16-bit up-counter, two 16-bit timer registers, two 16-bit capture registers, two comparators, a capture input control, a timer flip-flop and its associated control circuit. (TMRB0 through TMRB9). Each channel can operate in the following modes:

- 16-bit interval timer mode
- 16-bit event counter mode
- 16-bit programmable square-wave output mode (PPG)
- Timer synchronous mode (capable of setting output mode for each 4ch)

The use of the capture function allows TMRBs to perform the following three measurements:

- Frequency measurement
- Pulse width measurement
- Time difference measurement

The TMRB driver APIs provide a set of functions to configure each channel, such as setting the clock division, trailing timing and leading timing duration, capture timing and flip-flop function. And to control the running state of each channel such as controlling up-counter, the output of flip-flop and to indicate the status of each channel such as returning the factor of interrupt, value in capture registers and so on.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm365_tmrb.c, with /Libraries/TX03_Periph_Driver/inc/tmpm365_tmrb.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 Function List

- void TMRB_Enable(TSB_TB_TypeDef * ***TBx***);
- void TMRB_Disable(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetRunState(TSB_TB_TypeDef * ***TBx***, uint32_t ***Cmd***);
- void TMRB_Init(TSB_TB_TypeDef * ***TBx,*** TMRB_InitTypeDef * ***InitStruct***);
- void TMRB_SetCaptureTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***CaptureTiming***);
- void TMRB_SetFlipFlop(TSB_TB_TypeDef * ***TBx***,
  TMRB_FFOutputTypeDef * ***FFStruct***);
- TMRB_INTFactor TMRB_GetINTFactor(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetINTMask(TSB_TB_TypeDef * ***TBx***, uint32_t ***INTMask***);
- void TMRB_ChangeLeadingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***LeadingTiming***);
- void TMRB_ChangeTrailingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***TrailingTiming***);
- uint16_t TMRB_GetUpCntValue(TSB_TB_TypeDef * ***TBx***);
- uint16_t TMRB_GetCaptureValue(TSB_TB_TypeDef * ***TBx***, uint8_t ***CapReg***);
- void TMRB_ExecuteSWCapture(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetIdleMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);
- void TMRB_SetSyncMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);

- void TMRB_SetDoubleBuf(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);
- void TMRB_SetExtStartTrg(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***TrgMode***);
- void TMRB_SetClkInCoreHalt(TSB_TB_TypeDef * ***TBx***, uint8_t ***ClkState***);
- void TMRB_SetExtInput(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetDMAReq(TSB_TB_TypeDef * TBx, FunctionalState ***NewState***, uint8_t ***DMAReq***);
- void TMRB_SetFT0Sel (TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);

9.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Configure and control the common functions of each TMRB channel are handled by TMRB_Enable(), TMRB_Disable(), TMRB_Init(), TMRB_SetRunState(), TMRB_ChangeLeadingTiming() and TMRB_ChangeTrailingTiming().
2) Capture function of each TMRB channel is handled by TMRB_SetCaptureTiming(), and TMRB_ExecuteSWCapture().
3) The status indication of each TMRB channel is handled by TMRB_GetINTFactor(), TMRB_GetUpCntValue() and TMRB_GetCaptureValue().
4) TMRB_SetFlipFlop(), TMRB_SetINTMask(), TMRB_SetIdleMode(), TMRB_SetSyncMode(), TMRB_SetDoubleBuf(), TMRB_SetExtStartTrg() and TMRB_SetClkInCoreHalt (), TMRB_SetExtInput(), TMRB_SetDMAReq() , TMRB_SetFT0Sel() handle other specified functions.

9.2.3 Function Documentation

Note: in all of the following APIs, unless otherwise specified, the parameter:
"TSB_TB_TypeDef* ***TBx***" can be one of the following values:
TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB4, TSB_TB5, TSB_TB6, TSB_TB7, TSB_TB8, TSB_TB9.

9.2.3.1 TMRB_Enable

Enable the specified TMRB channel.

Prototype:
void
TMRB_Enable(TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

Description:
This function will enable the specified TMRB channel selected by ***TBx***.

Return:
None

9.2.3.2 TMRB_Disable

Disable the specified TMRB channel.

Prototype:
void
TMRB_Disable(TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

Description:
This function will disable the specified TMRB channel selected by ***TBx***.

Return:
None

9.2.3.3 TMRB_SetRunState

Start or stop counter of the specified TB channel.

Prototype:
void
TMRB_SetRunState(TSB_TB_TypeDef* ***TBx***,
uint32_t ***Cmd***)

Parameters:
TBx is the specified TMRB channel.
Cmd sets the state of up-counter, which can be:
- **TMRB_RUN:** starting counting
- **TMRB_STOP:** stopping counting

Description:
The up-counter of the specified TMRB channel starts counting if ***Cmd*** is **TMRB_RUN** and up-counter stops counting and the value in up-counter register is clear if ***Cmd*** is **TMRB_STOP**.

Return:
None

9.2.3.4 TMRB_Init

Initialize the specified TMRB channel.

Prototype:
void
TMRB_Init(TSB_TB_TypeDef* ***TBx***,
TMRB_InitTypeDef* ***InitStruct***)

Parameters:
TBx is the specified TMRB channel.
InitStruct is the structure containing basic TMRB configuration including count mode, source clock division, leading timing value, trailing timing value and up-counter work mode (refer to “Data Structure Description” for details).

Description:
This function will initialize and configure the count mode, clock division, up-counter setting, trailing timing and leading timing duration for the specified TMRB channel selected by ***TBx***.

Return:
None

9.2.3.5 TMRB_SetCaptureTiming

Configure the capture timing.

Prototype:
void
TMRB_SetCaptureTiming(TSB_TB_TypeDef* ***TBx***,
uint32_t ***CaptureTiming***)

Parameters:
TBx is the specified TMRB channel.

CaptureTiming specifies TMRB capture timing, which can be

- **TMRB_DISABLE_CAPTURE**: Disable the capture function of the specified TMRB channel.
- **TMRB_CAPTURE_IN_RISING**: Takes count values into capture register 0 (TBxCP0) upon rising of TBxIN pin input.
- **TMRB_CAPTURE_IN_RISING_FALLING**: Takes count values into capture register 0 (TBxCP0) upon rising of TBxIN pin input and takes count values into capture register 1 (TBxCP1) upon falling of TBxIN pin input.
- **TMRB_CAPTURE_OUTPUT_EDGE**: Takes count values into capture register 0 (TBxCP0) upon rising of TBxOUT pin input and takes count values into capture register 1 (TBxCP1) upon falling of TBxOUT pin input.

Description:
If ***CaptureTiming*** is set as **TMRB_CAPTURE_IN_RISING**, then at the time of the rising edge of input port TBxIN, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel.
If ***CaptureTiming*** is set as **TMRB_CAPTURE_IN_RISING_FALLING**, then at the time of the rising edge of input port TBxIN, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TBxIN pin input.
If ***CaptureTiming*** is set as **TMRB_CAPTURE_OUTPUT_EDGE**, then at the time of the rising edge of port TBxOUT pin, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TBxOUT pin input.
The flip-flop output of TMRB7, TMRB8 and TMRB9 can be used as the capture trigger of other channels.

TMRB0~1: TB7OUT
TMRB2~3: TB8OUT
TMRB4~6: TB9OUT

Return:
None

9.2.3.6 TMRB_SetFlipFlop

Configure the flip-flop function of the specified TMRB channel.

Prototype:
void
TMRB_SetFlipFlop(TSB_TB_TypeDef* ***TBx***,
TMRB_FFOutputTypeDef* ***FFStruct***)

Parameters:
TBx is the specified TMRB channel.
FFStruct is the structure containing TMRB flip-flop function configuration including flip-flop output level and flip-flop-reverse trigger (refer to "Data Structure Description" for details).

Description:
This function will set the timing of changing the flip-flop output of the specified TMRB channel. Also the level of the output can be controlled by this API.

Return:
None

9.2.3.7 TMRB_GetINTFactor

Indicate what causes the interrupt.

Prototype:
TMRB_INTFactor
TMRB_GetINTFactor(TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

Description:
This function should be used in ISR to indicate the factor of interrupt. Bit of **MatchLeadingTiming** indicates if the up-counter matches with leading timing value, Bit of **MatchTrailingTiming** Indicates if the up-counter matches with trailing timing value, and bit of **Overflow** indicates if overflow had occurred before the interrupt.

Return:
TMRB Interrupt factor. Each bit has the following meaning:
MatchLeadingTiming(Bit0): a match with the leading timing value is detected
MatchTrailingTiming(Bit1): a match with the trailing timing value is detected
OverFlow(Bit2): an up-counter is overflow

Note:
It is recommended to use the following method to process different interrupt factor

```
TMRB_INTFactor factor = TMRB_GetINTFactor(TSB_TB0);
if (factor.Bit.MatchLeadingTiming) {
    // Do A
}

if (factor.Bit.MatchTrailingTiming) {
    // Do B
}

if (factor.Bit.OverFlow) {
```

```
    // Do C
}
```

9.2.3.8 TMRB_SetINTMask

Mask the specified TMRB interrupt.

Prototype:
void
TMRB_SetINTMask(TSB_TB_TypeDef* ***TBx***,
uint32_t ***INTMask***)

Parameters:
TBx is the specified TMRB channel.
INTMask specifies the interrupt to be masked, which can be

- **TMRB_MASK_MATCH_TRAILINGTIMING_INT**: Mask the interrupt the factor of which is that the value in up-counter and trailing timing are match.
- **TMRB_MASK_MATCH_LEADINGTIMING_INT**: Mask the interrupt the factor of which is that the value in up-counter and leading timing are match.
- **TMRB_MASK_OVERFLOW_INT**: Mask the interrupt the factor of which is the occurrence of overflow.
- **TMRB_NO_INT_MASK**: Unmask the interrupt.

Description:
If **TMRB_MASK_MATCH_TRAILINGTIMING_INT** is selected, the interrupt of the specified TMRB channel will not happen when the value in up-counter and trailing timing are match.
If **TMRB_MASK_MATCH_LEADINGTIMING_INT** is selected, the interrupt of the specified TMRB channel will not happen when the value in up-counter and leading timing are match.
If **TMRB_MASK_OVERFLOW_INT** is selected, the interrupt of the specified TMRB channel will not happen even if there is an occurrence of overflow.
If **TMRB_NO_INT_MASK** is selected, all interrupt masks will be cleared.

Return:
None

9.2.3.9 TMRB_ChangeLeadingTiming

Change the value of leading timing for the specified channel.

Prototype:
void
TMRB_ChangeLeadingTiming(TSB_TB_TypeDef* ***TBx***,
uint32_t ***LeadingTiming***)

Parameters:
TBx is the specified TMRB channel.
LeadingTiming specifies the value of leading timing, max. is 0xFFFF.

Description:
This function will specify the absolute value of leading timing for the specified TMRB. The actual interval of leading timing depends on the configuration of CG and the value of ***ClkDiv*** (refer to “Data Structure Description” for details).

Return:
None

Note:
LeadingTiming can not exceed ***TrailingTiming***.

9.2.3.10 TMRB_ChangeTrailingTiming

Change the value of trailing timing for the specified channel.

Prototype:
void
TMRB_ChangeTrailingTiming(TSB_TB_TypeDef* ***TBx***,
uint32_t ***TrailingTiming***)

Parameters:
TBx is the specified TMRB channel.
TrailingTiming specifies the value of trailing timing, max. is 0xFFFF.

Description:
This function will specify the absolute value of trailing timing for the specified TMRB. The actual interval of trailing timing depends on the configuration of CG and the value of ***ClkDiv*** (refer to “Data Structure Description” for details).

Return:
None

Note:
TrailingTiming must be not smaller than ***LeadingTiming***. And the value of TBxRG0/1 must be set as TBxRG0 < TBxRG1 in PPG mode.

9.2.3.11 TMRB_GetUpCntValue

Get up-counter value of the specified TMRB channel.

Prototype:
uint16_t
TMRB_GetUpCntValue(TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

Description:
This function will return the value in up-counter of the specified TMRB channel.

Return:
The value of up-counter

9.2.3.12 TMRB_GetCaptureValue

Get the value of capture register0 or capture register1 of the specified TMRB channel.

Prototype:
uint16_t
TMRB_GetCaptureValue(TSB_TB_TypeDef* ***TBx***,
uint8_t ***CapReg***)

Parameters:
TBx is the specified TMRB channel.
CapReg is used to choose to return the value of capture register0 or to return the value of capture register1, which can be one of the following,

- **TMRB_CAPTURE_0:** specifying capture register0.
- **TMRB_CAPTURE_1:** specifying capture register1.

Description:
This function will return the value of capture register0 of the specified TMRB channel if ***CapReg*** is **TMRB_CAPTURE_0**, and will return the value of capture register1 of the specified TMRB channel if ***CapReg*** is **TMRB_CAPTURE_1**.

Return:
The captured value

9.2.3.13 TMRB_ExecuteSWCapture

Capture counter by software and take them into capture register 0 of the specified TMRB channel.

Prototype:
void
TMRB_ExecuteSWCapture(TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

Description:
This function will capture the up-counter of the specified TMRB channel by software and take the value into the capture register0.

Return:
None

9.2.3.14 TMRB_SetIdleMode

Enable or disable the specified TMRB channel when system is in idle mode.

Prototype:
void
TMRB_SetIdleMode(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***)

Parameters:
TBx is the specified TMRB channel.
NewState specifies the state of the TMRB when system is idle mode, which can be

- **ENABLE:** enables the TMRB channel,
- **DISABLE:** disables the TMRB channel.

Description:
The specified TMRB channel can still be running if ***NewState*** is **ENABLE** even if system enters idle mode. **DISABLE** can stop the running TMRB if system enters idle mode.

Return:
None

9.2.3.15 TMRB_SetSyncMode

Enable or disable the synchronous mode of specified TMRB channel.

Prototype:
void
TMRB_SetSyncMode(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***)

Parameters:
TBx is the specified TMRB channel, which can be
TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB4, TSB_TB5, TSB_TB6, TSB_TB7.

NewState specifies the state of the synchronous mode of the TMRB, which can be

- **ENABLE:** enables the synchronous mode,
- **DISABLE:** disables the synchronous mode.

Description:
If the synchronous mode is enabled for TMRB0 through TMRB3, their start timing is synchronized with TMRB0. If the synchronous mode is enabled for TMRB4 through TMRB7, their start timing is synchronized with TMRB4.

Return:
None

Note:
TMRB0 through TMRB3, TMRB4through TMRB7 must start counting by calling **TMRB_SetRunState()** before TMRB0, TMRB4 start counting, so that start timing can be synchronized.

9.2.3.16 TMRB_SetDoubleBuf

Enable or disable double buffering for the specified TMRB channel.

Prototype:
void
TMRB_SetDoubleBuf(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***)

Parameters:
TBx is the specified TMRB channel.
NewState specifies the state of double buffering of the TMRB, which can be

- **ENABLE:** enables double buffering,
- **DISABLE:** disables double buffering.

Description:
This function will enable or disable double buffering for the specified TMRB channel.
Return:
None

9.2.3.17 TMRB_SetExtStartTrg

Enable or disable external trigger TBxIN to start count and set the active edge.

Prototype:
void
TMRB_SetExtStartTrg (TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***,
uint8_t ***TrgMode***)

Parameters:
TBx is the specified TMRB channel.
NewState specifies the state external trigger, which can be
- **ENABLE:** use external trigger signal,
- **DISABLE:** use software start.

TrgMode specifies active edge of the external trigger signal., which can be
- **TMRB_TRG_EDGE_RISING:** Select rising edge of external trigger.
- **TMRB_TRG_EDGE_FALLING:** Select falling edge of external trigger.

Description:
This function will enable or disable external trigger to start count and set the active edge.

Return:
None

9.2.3.18 TMRB_SetClkInCoreHalt

Enable or disable clock operation in Core HALT during debug mode.

Prototype:
void
TMRB_SetClkInCoreHalt (TSB_TB_TypeDef* ***TBx***,
uint8_t ***ClkState***)

Parameters:
TBx is the specified TMRB channel.
ClkState specifies timer state in HALT mode, which can be
- **TMRB_RUNNING_IN_CORE_HALT:** clock not stops in Core HALT
- **TMRB_STOP_IN_CORE_HALT:** clock stops in Core HALT.

Description:
This function will set enable or disable clock operation in Core HALT during debug mode.

Return:
None

9.2.3.19 TMRB_SetExtInput

Set the external input source

Prototype:
void
TMRB_SetExtInput (TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

Description:
This function will set TBxIN0/1 as the external input for the specified TMRB channel.

Return:
None

9.2.3.20 TMRB_SetDMAReq

Enable or disable the selected DMA request for a TMRB channel.

Prototype:
void
TMRB_SetExtStartTrg (TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***,
uint8_t ***DMAReq***)

Parameters:
TBx is the specified TMRB channel.
NewState enable or disable the DMA request,which can be
- **ENABLE:** enable the DMA request,
- **DISABLE:** disable the DMA request.

DMAReq specifies DMA request of the external inputs, which can be
- **TMRB_DMA_REQ_CMP_MATCH:** Select DMA request : compare match.
- **TMRB_DMA_REQ_CAPTURE_1:** Select DMA request : input capture1.
- **TMRB_DMA_REQ_CAPTURE_0:** Select DMA request : input capture0.

Description:
This function will enable or disable the selected DMA request for the specified TMRB channel.

Return:
None

Note:
When mask configuration by TBxIM register is valid, DMA request does not issue even if it is enabled.

9.2.3.21 TMRB_ SetFT0Sel

Enable or Disable Select source clock T0 as source clock for tmrb.
Prototype:
void
TMRB_ SetFT0Sel (TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***)

Parameters:
TBx is the specified TMRB channel.
NewState New state of TBx source clock select.
- **ENABLE:** enables select phi T0.,
- **DISABLE:** disables select phi T0.

Description:
This function will enable or disable Select source clock T0 as source clock for tmrb.
Return:
None

9.2.4 Data Structure Description

9.2.4.1 TMRB_InitTypeDef

Data Fields:

uint32_t
Mode selects TMRB working mode between **TMRB_INTERVAL_TIMER** (internal interval timer mode) and **TMRB_EVENT_CNT** (external event counter).

uint32_t
ClkDiv specifies the division of the source clock for the internal interval timer, which can be set as:
- **TMRB_CLK_DIV_2**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 2;
- **TMRB_CLK_DIV_8**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 8;
- **TMRB_CLK_DIV_32**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 32.
- **TMRB_CLK_DIV_64**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 64;
- **TMRB_CLK_DIV_128**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 128.
- **TMRB_CLK_DIV_256**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 256;
- **TMRB_CLK_DIV_512**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 512

uint32_t
TrailingTiming specifies the trailing timing value to be written into TBnRG1, max. 0xFFFF.

uint32_t
UpCntCtrl selects up-counter work mode, which can be set as:

- **TMRB_FREE_RUN**, which means that the up-counter will not stop counting even when the value in it is match with trailing timing, until it reaches 0xFFFF, then it will be cleared and starting counting from 0,
- **TMRB_AUTO_CLEAR**, which means that the up-counter will restart counting from 0 immediately when the value in up-counter matches **TrailingTiming**.

uint32_t
LeadingTiming specifies the leading timing value to be written into TBnRG0, max. 0xFFFF, and it can not be set larger than **TrailingTiming**.

9.2.4.2 TMRB_FFOutputTypeDef

Data Fields:

uint32_t
FlipflopCtrl selects the level of flip-flop output which can be

- **TMRB_FLIPFLOP_INVERT:** setting output reversed by using software.
- **TMRB_FLIPFLOP_SET:** setting output to be high level.
- **TMRB_FLIPFLOP_CLEAR:** setting output to be low level.

uint32_t
FlipflopReverseTrg specifies the reverse trigger of the flip-flop output, which can be set as:

- **TMRB_DISALBE_FLIPFLOP**, which disables the flip-flop output reverse trigger,
- **TMRB_FLIPFLOP_TAKE_CATPURE_0**, which means that the reversing flip-flop output will be triggered when the up-counter value is taken into capture register 0,
- **TMRB_FLIPFLOP_TAKE_CATPURE_1**, which means that the reversing flip-flop output will be triggered when the up-counter value is taken into capture register 1,
- **TMRB_FLIPFLOP_MATCH_TRAILINGTIMING**, which means that the reversing flip-flop output will be triggered when the up-counter matches the trailing timing,
- **TMRB_FLIPFLOP_MATCH_LEADINGTIMING**, which means that the reversing flip-flop output will be triggered when the up-counter matches the leading timing.

9.2.4.3 TMRB_INTFactor

Data Fields:

uint32_t
All: TMRB interrupt factor.
Bit
uint32_t
MatchLeadingTiming: 1 a match with the leading timing value is detected
uint32_t
MatchTrailingTiming : 1 a match with the trailing timing value is detected
uint32_t
OverFlow : 1 an up-counter is overflow
uint32_t
Reserverd : 29 -

10. SIO/UART

10.1 Overview

This device has several serial I/O channels. Each channel can operate in I/O Interface mode (synchronous communication) and UART mode (asynchronous communication), which can be 7-bit length, 8-bit length and 9-bit length.
In 9-bit UART mode, a wakeup function can be used when the master controller can start up slave controllers via the serial link (multi-controller system).

The UART driver APIs provide a set of functions to configure each channel, including such common parameters as baud rate, bit length, parity check, stop bit, flow control, and to control transfer like sending/receiving data, checking error and so on.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm365_uart.c, with /Libraries/TX03_Periph_Driver/inc/tmpm365_uart.h containing the macros, data types, structures and API definitions for use by applications.

10.2 API Functions

10.2.1 Function List

- void UART_Enable(TSB_SC_TypeDef* ***UARTx***)
- void UART_Disable(TSB_SC_TypeDef* ***UARTx***)
- WorkState UART_GetBufState(TSB_SC_TypeDef* ***UARTx***, uint8_t ***Direction***)
- void UART_SWReset(TSB_SC_TypeDef* ***UARTx***)
- void UART_Init(TSB_SC_TypeDef* ***UARTx***, UART_InitTypeDef* ***InitStruct***)
- uint32_t UART_GetRxData(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetTxData(TSB_SC_TypeDef* ***UARTx***, uint32_t ***Data***)
- void UART_DefaultConfig(TSB_SC_TypeDef* ***UARTx***)
- UART_Err UART_GetErrState(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetWakeUpFunc(TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetIdleMode(TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetRxDMAReq (TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetTxDMAReq (TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_FIFOConfig(TSB_SC_TypeDef * UARTx, FunctionalState NewState)
- void UART_SetFIFOTransferMode(TSB_SC_TypeDef * UARTx, uint32_t TransferMode)
- void UART_TRxAutoDisable(TSB_SC_TypeDef * UARTx, UART_TRxDisable TrxAutoDisable
- void UART_RxFIFOINTCtrl(TSB_SC_TypeDef * UARTx, FunctionalState NewState)
- void UART_TxFIFOINTCtrl(TSB_SC_TypeDef * UARTx, FunctionalState NewState)
- void UART_RxFIFOByteSel(TSB_SC_TypeDef * UARTx, uint32_t BytesUsed)
- void UART_RxFIFOFillLevel(TSB_SC_TypeDef * UARTx,uint32_t RxFIFOLevel)
- void UART_RxFIFOINTSel(TSB_SC_TypeDef * UARTx, uint32_t RxINTCondition)
- void UART_RxFIFOClear(TSB_SC_TypeDef * UARTx)
- void UART_TxFIFOFillLevel(TSB_SC_TypeDef * UARTx, uint32_t TxFIFOLevel)
- void UART_TxFIFOINTSel(TSB_SC_TypeDef * UARTx, uint32_t TxINTCondition)
- void UART_TxFIFOClear(TSB_SC_TypeDef * UARTx)
- uint32_t UART_GetRxFIFOFillLevelStatus(TSB_SC_TypeDef * UARTx)
- uint32_t UART_GetRxFIFOOverRunStatus(TSB_SC_TypeDef * UARTx)
- uint32_t UART_GetTxFIFOFillLevelStatus(TSB_SC_TypeDef * UARTx)

- uint32_t UART_GetTxFIFOUnderRunStatus(TSB_SC_TypeDef * UARTx)
- void SIO_Enable(TSB_SC_TypeDef * SIOx)
- void SIO_Disable(TSB_SC_TypeDef * SIOx)
- uint8_t SIO_GetRxData(TSB_SC_TypeDef * SIOx)
- void SIO_SetTxData(TSB_SC_TypeDef * SIOx, uint8_t Data)
- void SIO_Init(TSB_SC_TypeDef * SIOx, uint32_t IOClkSel, SIO_InitTypeDef * InitStruct)

10.2.2 Detailed Description

Functions listed above can be divided into four parts:

1. Initialize and configure the common functions of each UART channel are handled by UART_Enable(), UART_Disable(),UART_,UART_Init() and UART_DefaultConfig(),SIO_Enable(), SIO_Disable(),SIO_,SIO_Init().
2. Transfer control and error check of each UART channel are handled by UART_GetBufState(), UART_GetRxData(), UART_SetTxData() and UART_GetErrState(),SIO_GetRxData(), SIO_SetTxData().
3. UART_SetRxDMAReq (),UART_SetTxDMAReq(), UART_SWReset(), UART_SetWakeUpFunc() and UART_SetIdleMode() handle other specified functions.
4. FIFO operation functions are UART_FIFOConfig(),UART_SetFIFOTransferMode(), UART_TrxAutoDisable(),UART_RxFIFOINTCtrl(),UART_TxFIFOINTCtrl(), UART_RxFIFOByteSel(),UART_RxFIFOFillLevel(),UART_RxFIFOINTSel(). UART_RxFIFOClear(),UART_TxFIFOFillLevel(),UART_TxFIFOINTSel(). UART_TxFIFOClear(),UART_GetRxFIFOFillLevelStatus(), UART_GetRxFIFOOverRunStatus(),UART_GetTxFIFOFillLevelStatus()and UART_GetTxFIFOUnderRunStatus().

10.2.3 Function Documentation

Note: in all of the following APIs, parameter “TSB_SC_TypeDef* ***UARTx***” can be one of the following values:
UART0, **UART1**.
parameter “TSB_SC_TypeDef* ***SIOx***” can be one of the following values:
SIO0, **SIO1**.

10.2.3.1 UART_Enable

Enable the specified UART channel.

Prototype:
void
UART_Enable(TSB_SC_TypeDef* ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will enable the specified UART channel selected by ***UARTx***.

Return:
None

10.2.3.2 UART_Disable

Disable the specified UART channel.

Prototype:
void
UART_Disable(TSB_SC_TypeDef* ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will disable the specified UART channel selected by ***UARTx***.

Return:
None

10.2.3.3 UART_GetBufState

Indicate the state of transmission or reception buffer.

Prototype:
WorkState
UART_GetBufState(TSB_SC_TypeDef* ***UARTx***,
uint8_t ***Direction***)

Parameters:
UARTx is the specified UART channel.
Direction select the direction of transfer, which can be one of:
- **UART_RX** for reception
- **UART_TX** for transmission

Description:
When ***Direction*** is **UART_RX**, the function returns the state of the reception buffer, which can be **DONE**, meaning that the data received has been saved into the buffer, or **BUSY**, meaning that the data reception is in progress. When ***Direction*** is **UART_TX**, the function returns state of the reception buffer, which can be **DONE**, meaning that the data to be set in the buffer has been sent, or **BUSY**, the data transmission is in progress.

Return:
DONE means that the buffer can be read or written.
BUSY means that the transfer is ongoing.

10.2.3.4 UART_SWReset

Reset the specified UART channel.

Prototype:
void
UART_SWReset(TSB_SC_TypeDef* ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:

This function will reset the specified UART channel selected by ***UARTx***.

Return:
None

10.2.3.5 UART_Init

Initialize and configure the specified UART channel.

Prototype:
void
UART_Init(TSB_SC_TypeDef* ***UARTx***,
UART_InitTypeDef* ***InitStruct***)

Parameters:
UARTx is the specified UART channel.
InitStruct is the structure containing basic UART configuration including baud rate, data bits per transfer, stop bits, parity, transfer mode and flow control (refer to “Data Structure Description” for details).

Description:
This function will initialize and configure the baud rate, the number of bits per transfer, stop bit, parity, transfer mode and flow control for the specified UART channel selected by ***UARTx***.

Return:
None

10.2.3.6 UART_GetRxData

Get data received from the specified UART channel.

Prototype:
uint32_t
UART_GetRxData(TSB_SC_TypeDef* ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will get the data received from the specified UART channel selected by ***UARTx***. It is appropriate to call the function after **UART_GetBufState(*UARTx,* UART_RX)** returns **DONE** or in an ISR of UART (serial channel).

Return:
Data which has been received

10.2.3.7 UART_SetTxData

Set data to be sent and start transmitting from the specified UART channel.

Prototype:

void
UART_SetTxData(TSB_SC_TypeDef* ***UARTx***,
uint32_t ***Data***)

Parameters:
UARTx is the specified UART channel.
Data is a frame to be sent, which can be 7-bit, 8-bit or 9-bit, depending on the initialization.

Description:
This function will set the data to be sent from the specified UART channel selected by ***UARTx***. It is appropriate to call the function after **UART_GetBufState(*UARTx,* UART_TX)** returns **DONE** or in an ISR of UART (serial channel).

Return:
None

10.2.3.8 UART_DefaultConfig

Initialize the specified UART channel in the default configuration.

Prototype:
void
UART_DefaultConfig(TSB_SC_TypeDef* ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will initialize the selected UART channel in the following configuration:
Baud rate: 115200 bps
Data bits: 8 bits
Stop bits: 1 bit
Parity: None
Flow Control: None
Both transmission and reception are enabled. And baud rate generator is used as source clock.

Return:
None

10.2.3.9 UART_GetErrState

Get error flag of the transfer from the specified UART channel.

Prototype:
UART_Err
UART_GetErrState(TSB_SC_TypeDef* ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will check whether an error occurs at the last transfer and return the result, which can be **UART_NO_ERR**, meaning no error, **UART_OVERRUN**, meaning overrun, **UART_PARITY_ERR**, meaning even or odd parity error, **UART_FRAMING_ERR**, meaning framing error, and **UART_ERRS**, meaning more than one error above.

Return:
UART_NO_ERR means there is no error in the last transfer.
UART_OVERRUN means that overrun occurs in the last transfer.
UART_PARITY_ERR means either even parity or odd parity fails.
UART_FRAMING_ERR means there is framing error in the last transfer.
UART_ERRS means that 2 or more errors occurred in the last transfer.

10.2.3.10 UART_SetWakeUpFunc

Enable or disable wake-up function in 9-bit mode of the specified UART channel.

Prototype:
void
UART_SetWakeUpFunc(TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of wake-up function.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable wake-up function of the specified UART channel selected by ***UARTx*** when ***NewState*** is **ENABLE**, and disable the wake-up function when ***NewState*** is **DISABLE**. Most of all, the wake-up function is only working in 9-bit UART mode.

Return:
None

10.2.3.11 UART_SetIdleMode

Enable or disable the specified UART channel when system is in idle mode.

Prototype:
void
UART_SetIdleMode(TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of the UART channel in system idle mode.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable the specified UART channel selected by ***UARTx*** in system idle mode when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:
None

10.2.3.12 UART_SetRxDMAReq

Enable or disable the specified UART channel DMA Request By receive interrupt INTRX.

Prototype:
void
UART_SetRxDMAReq (TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of the UART channel DMA Request.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable the Rx DMA Reuqest of the specified UART channel selected by ***UARTx*** DMA Request when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:
None

10.2.3.13 UART_SetTxDMAReq

Enable or disable the specified UART channel DMA Request By receive interrupt INTTX.

Prototype:
void
UART_SetTxDMAReq (TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of the UART channel DMA Request.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable the Tx DMA Request of the specified UART channel selected by ***UARTx*** DMA Request when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:

None

10.2.3.14 UART_FIFOConfig

Enable or disable the FIFO of specified UART channel.

Prototype:
void
UART_FIFOConfig(TSB_SC_TypeDef * ***UARTx,***
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of the UART FIFO.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable the FIFO of specified UART channel selected by ***UARTx*** when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:
None

10.2.3.15 UART_SetFIFOTransferMode

Transfer mode setting.

Prototype:
void
UART_SetFIFOTransferMode(TSB_SC_TypeDef * ***UARTx,***
uint32_t ***TransferMode***)

Parameters:
UARTx is the Selected the UART channel.
TransferMode is the Transfer mode.

This parameter can be one of the following values:
UART_TRANSFER_PROHIBIT, UART_TRANSFER_HALFDPX_RX, UART_TRANSFER_HALFDPX_TX, UART_TRANSFER_FULLDPX.

Description:
This function will set the transfer mode of specified UART channel selected by ***UARTx***. The UART transfer mode has only 4 modes which above displays.

Return:
None

10.2.3.16 UART_TRxAutoDisable

Controls automatic disabling of transmission and reception.

Prototype:

void
UART_TRxAutoDisable (TSB_SC_TypeDef * ***UARTx,***
UART_TRxDisable ***TRxAutoDisable***)

Parameters:
UARTx is the specified UART channel.
TRxAutoDisable is the Disabling transmission and reception or not.

This parameter can be one of the following values:
UART_RXTXCNT_NONE or **UART_RXTXCNT_AUTODISABLE**

Description:
This function will Control automatic disabling of transmission and reception, in specified UART channel selected by ***UARTx*** when ***TRxAutoDisable*** is **UART_RXTXCNT_AUTODISABLE**, and disable the channel when ***TRxAutoDisable*** is **UART_RXTXCNT_NONE**.

Return:
None

10.2.3.17 UART_RxFIFOINTCtrl

Enable or disable receive interrupt for receive FIFO.

Prototype:
void
UART_RxFIFOINTCtrl (TSB_SC_TypeDef * ***UARTx,***
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of receive interrupt for receive FIFO.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable receive interrupt for receive FIFO of specified UART channel selected by ***UARTx*** when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE.**

Return:
None

10.2.3.18 UART_TxFIFOINTCtrl

Enable or disable transmit interrupt for transmit FIFO.

Prototype:
void
UART_TxFIFOINTCtrl (TSB_SC_TypeDef * ***UARTx,***
FunctionalState ***NewState***)

Parameters:
UARTx is the specified UART channel.
NewState is the new state of transmit interrupt for receive FIFO.

This parameter can be one of the following values:
ENABLE or **DISABLE**

Description:
This function will enable transmit interrupt for receive FIFO of specified UART channel selected by ***UARTx*** when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE.**

Return:
None

10.2.3.19 UART_RxFIFOByteSel

Bytes used in receive FIFO.

Prototype:
void
UART_RxFIFOByteSel (TSB_SC_TypeDef * ***UARTx,***
uint32_t ***BytesUsed***)

Parameters:
UARTx is the specified UART channel.
BytesUsed is the Bytes used in receive FIFO.

This parameter can be one of the following values:
UART_RXFIFO_MAX or **UART_RXFIFO_RXFLEVEL**

Description:
This function will set numbers of bytes used in receive FIFO of specified UART channel selected by ***UARTx***, But the bytes of the number should be **UART_RXFIFO_MAX** or **UART_RXFIFO_RXFLEVEL.**

Return:
None

10.2.3.20 UART_RxFIFOFillLevel

Receive FIFO fill level to generate receive interrupts.

Prototype:
void
UART_RxFIFOFillLevel (TSB_SC_TypeDef * ***UARTx,***
uint32_t ***RxFIFOLevel***)

Parameters:
UARTx is the specified UART channel.
RxFIFOLevel is the Receive FIFO fill level.

This parameter can be one of the following values:
UART_RXFIFO4B_FLEVLE_4_2B, UART_RXFIFO4B_FLEVLE_1_1B, UART_RXFIFO4B_FLEVLE_2_2B, UART_RXFIFO4B_FLEVLE_3_1B.

Description:
This function will set Receive FIFO fill level for generate receive interrupts of specified UART channel selected by ***UARTx***, But the level should be **UART_RXFIFO4B_FLEVLE_4_2B, UART_RXFIFO4B_FLEVLE_1_1B,**

UART_RXFIFO4B_FLEVLE_2_2B, UART_RXFIFO4B_FLEVLE_3_1B.

Return:
None

10.2.3.21 UART_RxFIFOINTSel

Select RX interrupt generation condition.

Prototype:
void
UART_RxFIFOINTSel (TSB_SC_TypeDef * ***UARTx,***
uint32_t ***RxINTCondition***)

Parameters:
UARTx is the specified UART channel.
RxINTCondition is the RX interrupt generation condition.

This parameter can be one of the following values:
UART_RFIS_REACH_FLEVEL or **UART_RFIS_REACH_EXCEED_FLEVEL**

Description:
This function will set RX interrupt generation condition of specified UART channel selected by ***UARTx***, But the level should be **UART_RFIS_REACH_FLEVEL, UART_RFIS_REACH_EXCEED_FLEVEL.**

Return:
None

10.2.3.22 UART_RxFIFOClear

Clear Receive FIFO.

Prototype:
void
UART_RxFIFOClear (TSB_SC_TypeDef * ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will Clear Receive FIFO of specified UART channel selected by ***UARTx***.

Return:
None

10.2.3.23 UART_TxFIFOFillLevel

Transmit FIFO fill level to generate receive interrupts.

Prototype:
void
UART_TxFIFOFillLevel (TSB_SC_TypeDef * ***UARTx,***
uint32_t ***TxFIFOLevel***)

Parameters:
UARTx is the specified UART channel.
TxFIFOLevel is the Receive FIFO fill level.

This parameter can be one of the following values:
UART_TXFIFO4B_FLEVLE_0_0B, UART_TXFIFO4B_FLEVLE_1_1B, UART_TXFIFO4B_FLEVLE_2_0B, UART_TXFIFO4B_FLEVLE_3_1B

Description:
This function will set Transmit FIFO fill level for generate receive interrupts of specified UART channel selected by ***UARTx***, But the level should be **UART_TXFIFO4B_FLEVLE_0_0B, UART_TXFIFO4B_FLEVLE_1_1B, UART_TXFIFO4B_FLEVLE_2_0B, UART_TXFIFO4B_FLEVLE_3_1B.**

Return:
None

10.2.3.24 UART_TxFIFOINTSel

Select TX interrupt generation condition.

Prototype:
void
UART_TxFIFOINTSel (TSB_SC_TypeDef * ***UARTx,***
uint32_t ***TxINTCondition***)

Parameters:
UARTx is the specified UART channel.
TxINTCondition is the TX interrupt generation condition.

This parameter can be one of the following values:
UART_TFIS_REACH_FLEVEL or **UART_TFIS_REACH_EXCEED_FLEVEL**

Description:
This function will set TX interrupt generation condition of specified UART channel selected by ***UARTx***, But the level should be **UART_TFIS_REACH_FLEVEL, UART_TFIS_REACH_EXCEED_FLEVEL.**

Return:
None

10.2.3.25 UART_TxFIFOClear

Clear Transmit FIFO.

Prototype:
void
UART_TxFIFOClear (TSB_SC_TypeDef * ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will Clear Transmit FIFO of specified UART channel selected by ***UARTx***.

Return:
None

10.2.3.26 UART_GetRxFIFOFillLevelStatus

Indicate the status of receive FIFO fill level.

Prototype:
uint32_t
UART_GetRxFIFOFillLevelStatus (TSB_SC_TypeDef * ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will Indicate the receive FIFO fill level status, which UART channel selected by ***UARTx*.**

Return:
UART_TRXFIFO_EMPTY: TX FIFO fill level is empty.
UART_TRXFIFO_1B: TX FIFO fill level is 1 byte.
UART_TRXFIFO_2B: TX FIFO fill level is 2 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 3 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 4 bytes.

10.2.3.27 UART_GetRxFIFOOverRunStatus

Indicate the status of Receive FIFO overrun.

Prototype:
uint32_t
UART_GetRxFIFOOverRunStatus (TSB_SC_TypeDef * ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
This function will Indicate the Receive FIFO overrun status, which UART channel selected by ***UARTx*.**

Return:
UART_RXFIFO_OVERRUN: Flags for RX FIFO overrun.

10.2.3.28 UART_GetTxFIFOFillLevelStatus

Status of transmit FIFO fill level.

Prototype:
uint32_t
UART_GetTxFIFOFillLevelStatus (TSB_SC_TypeDef * ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
Status of transmit FIFO fill level.

Return:
UART_TRXFIFO_EMPTY: TX FIFO fill level is empty.
UART_TRXFIFO_1B: TX FIFO fill level is 1 byte.
UART_TRXFIFO_2B: TX FIFO fill level is 2 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 3 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 4 bytes.

10.2.3.29 UART_GetTxFIFOUnderRunStatus

Transmit FIFO under run.

Prototype:
uint32_t
UART_ GetTxFIFOUnderRunStatus (TSB_SC_TypeDef * ***UARTx***)

Parameters:
UARTx is the specified UART channel.

Description:
Transmit FIFO under run.

Return:
UART_TXFIFO_UNDERRUN: Flags for TX FIFO under-run.

10.2.3.30 SIO_Enable

Enable the specified SIO channel.

Prototype:
void
SIO_Enable (TSB_SC_TypeDef* ***SIOx***)

Parameters:
SIOx is the specified SIOx channel.

Description:
This function will enable the specified SIO channel selected by ***SIOx***.

Return:
None

10.2.3.31 SIO_Disable

Disable the specified SIO channel.

Prototype:
void
SIO _Disable(TSB_SC_TypeDef* ***SIOx***)

Parameters:
SIOTx is the specified SIO channel.

Description:
This function will disable the specified SIO channel selected by ***SIOx***.

Return:
None

10.2.3.32 SIO_GetRxData

Get data received from the specified SIO channel.

Prototype:
uint32_t
SIO_GetRxData(TSB_SC_TypeDef* ***SIOx***)

Parameters:
SIOx is the specified SIO channel.

Description:
This function will get the data received from the specified SIO channel selected by ***SIOx***.

Return:
Data which has been received, the data value range is 0x00 to 0xFF.

10.2.3.33 SIO_SetTxData

Set data to be sent and start transmitting from the specified SIO channel.

Prototype:
void
SIO_SetTxData(TSB_SC_TypeDef* ***SIOx***,
uint8_t ***Data***)

Parameters:
SIOx is the specified SIO channel.
Data is a frame to be sent,

Description:
This function will set the data to be sent from the specified SIO channel selected by ***SIOx***.

Return:
None

10.2.3.34 SIO_Init

Initialize and configure the specified SIO channel.

Prototype:

void
SIO_Init(TSB_SC_TypeDef* ***SIOx***,
uint32_t ***IOClkSel,***
SIO_InitTypeDef* ***InitStruct***)

Parameters:
SIOx is the specified SIO channel.
IOClkSel is the work clock
InitStruct is the structure containing basic SIO configuration including baud rate, transmission direction, transfer mode.

Description:
This function will initialize and configure the baud rate, transmission direction, transfer mode for the specified SIO channel selected by ***SIOx***.

Return:
None

10.2.4 Data Structure Description

10.2.4.1 UART_InitTypeDef

Data Fields:

uint32_t
BaudRate configures the UART communication baud rate ranging from 2400(bps) to 115200(bps) (*).

uint32_t
DataBits specifies data bits per transfer, which can be set as:
- **UART_DATA_BITS_7** for 7-bit mode
- **UART_DATA_BITS_8** for 8-bit mode
- **UART_DATA_BITS_9** for 9-bit mode

uint32_t
StopBits specifies the length of stop bit transmission in UART mode, which can be set as:
- **UART_STOP_BITS_1** for 1 stop bit
- **UART_STOP_BITS_2** for 2 stop bits

uint32_t
Parity specifies the parity mode, which can be set as:
- **UART_NO_PARITY** for no parity
- **UART_EVEN_PARITY** for even parity
- **UART_ODD_PARITY** for odd parity

uint32_t
Mode enables or disables reception, transmission or both, which can be set as one of the followings or both by using a logical OR operation:
- **UART_ENABLE_TX** for enabling transmission
- **UART_ENABLE_RX** for enabling reception

uint32_t
FlowCtrl specifies whether the hardware flow control mode is enabled or disabled (**). It can be set as:
- **UART_NONE_FLOW_CTRL** for no flow control

*: If the frequency of fperiph (refer to CG for details) is set too low or too high, the baud rate can not be configured correctly.

**: Only UART_NONE_FLOW_CTRL is included in this version.

10.2.4.2 SIO_InitTypeDef

Data Fields:

uint32_t
InputClkEdge Select the input clock edge.on the SCLK output mode this bit only can set to be 0(SIO_SCLKS_TXDF_RXDR).

uint32_t
IntervalTime Setting interval time of continuous transmission, which could be set as:

- **SIO_SINT_TIME_NONE** for none
- **SIO_SINT_TIME_SCLK_1** for 1*SCLK
- **SIO_SINT_TIME_SCLK_2** for 2*SCLK
- **SIO_SINT_TIME_SCLK_4** for 4*SCLK
- **SIO_SINT_TIME_SCLK_8** for 8*SCLK
- **SIO_SINT_TIME_SCLK_16** for 16*SCLK
- **SIO_SINT_TIME_SCLK_32** for 32*SCLK
- **SIO_SINT_TIME_SCLK_64** for 64*SCLK

uint32_t
TransferMode Setting transfer mode which could be transfer prohibited, which can be set as:

- **SIO_TRANSFER_PROHIBIT** for transfer prohibited.
- **SIO_TRANSFER_HALFDPX_RX** for half duplex(Receive).
- **SIO_TRANSFER_HALFDPX_TX** for half duplex(Transmit).
- **SIO_TRANSFER_FULLDPX** for full duplex.

uint32_t
TransferDir sets transfer direction which could be set as:

- **SIO_LSB_FRIST** for LSB FRIST in transmission
- **SIO_MSB_FRIST** for MSB FRIST in transmission.

uint32_t
Mode enables or disables receive, transmission or both, which can be set as one of the followings or both by using a logical OR operation:

- **SIO_ENABLE_TX** for enabling transmission
- **SIO_ENABLE_RX** for enabling reception

uint32_t
DoubleBuffer Double Buffer mode is enabled or disabled.

uint32_t
BaudRateClock Select the input clock for baud rate generator, which can be set as:

- **SIO_BR_CLOCK_T1**
- **SIO_BR_CLOCK_T4**
- **SIO_BR_CLOCK_T16**
- **SIO_BR_CLOCK_T64**

uint32_t
Divider division ratio "N",which can be set as:

- **SIO_BR_DIVIDER_1**
- **SIO_BR_DIVIDER_2**
- **SIO_BR_DIVIDER_3**
- **SIO_BR_DIVIDER_4**
- **SIO_BR_DIVIDER_5**
- **SIO_BR_DIVIDER_6**
- **SIO_BR_DIVIDER_7**
- **SIO_BR_DIVIDER_8**

- **SIO_BR_DIVIDER_9**
- **SIO_BR_DIVIDER_10**
- **SIO_BR_DIVIDER_11**
- **SIO_BR_DIVIDER_12**
- **SIO_BR_DIVIDER_13**
- **SIO_BR_DIVIDER_14**
- **SIO_BR_DIVIDER_15**
- **SIO_BR_DIVIDER_16**

11. USBD

11.1 Overview

TOSHIBA TMPM365 has an USB Device Controller which provide features as below:

1. Conforming to Universal Serial Bus Specification Rev.2.0
2. Supports Full-Speed (Low-Speed is not supported).
3. USB protocol processing.
4. Detects SOF/USB_RESET/SUSPEND/RESUME
5. Generates and checks packet IDs.
6. Checks CRC5, generate and checks CRC16.
7. Supports 4 transfer types(Control/ Interrupt/ Bulk/ Isochronous).
8. Supports 8 endpoints:

Endpoint0	Control	64byte x 1 FIFO
Endpoint1	Control/Interrupt/Bulk/Isochronous(IN)	64byte x 2 FIFO
Endpoint2	Control/Interrupt/Bulk/Isochronous(OUT)	64byte x 2 FIFO
Endpoint3	Control/Interrupt/Bulk/Isochronous(IN)	64byte x 2 FIFO
Endpoint4	Control/Interrupt/Bulk/Isochronous(OUT)	64byte x 2 FIFO
Endpoint5	Control/Interrupt/Bulk/Isochronous(IN)	64byte x 2 FIFO
Endpoint6	Control/Interrupt/Bulk/Isochronous(OUT)	64byte x 2 FIFO
Endpoint7	Control/Interrupt/Bulk/Isochronous(IN)	64byte x 2 FIFO

9. Supports Dual Packets Mode(except for Endpoint 0)
10. Interrupt source signal to Interrupt controller: INTUSB , INTUSBWKUP

All basic driver APIs are contained in \Libraries\TX03_USBD_Driver\src\usbd_hw.c, with \Libraries\TX03_USBD_Driver\inc\usbd_hw.h containing the macros, data types, structures and API definitions for use by applications.

11.2 API Functions

11.2.1 Function List

- USBD_INTStatus USBD_GetINTStatus(void) ;
- void USBD_SetINTMask(USBD_INTSrc ***IntSrc***, FunctionalState ***NewState***) ;
- void USBD_ClearINT(USBD_INTSrc ***IntSrc***) ;
- USBD_DMACConfig USBD_GetDMACConfig(void) ;
- void USBD_ConfigDMAC(USBD_DMACConfig ***Config***) ;
- USBD_DMACStatus USBD_GetDMACStatus(void) ;
- void USBD_ReadUDC2Reg(uint32_t ***Addr***, uint32_t * ***Data***) ;
- void USBD_WriteUDC2Reg(uint32_t ***Addr***, const uint32_t ***Data***) ;
- USBD_DMACAddr USBD_GetDMACMasterAddr(USBD_MasterMode ***MasterMode***) ;
- void USBD_SetDMACMasterAddr(USBD_MasterMode ***MasterMode***, USBD_DMACAddr ***Addr***) ;
- USBD_PowerCtrl USBD_GetPowerCtrlStatus(void) ;
- void USBD_SetPowerCtrl(USBD_PowerCtrl ***PowerCtrl***) ;
- void USBD_SetEPCMD(USBD_EPx ***EPx***, USBD_EPCMD ***Cmd***) ;
- USBD_EP0Status USBD_GetEP0Status(void) ;

- USBD_EPxStatus USBD_GetEPxStatus(USBD_EPx ***EPx***) ;
- void USBD_ConfigEPx(USBD_EPx ***EPx***, USBD_EPxConfig ***Config***) ;

11.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Initialization and configuration of the common functions of USBD are handled by:
   USBD_SetINTMask(), USBD_ClearINT(), USBD_SetEPCMD(),
   USBD_ConfigEPx(), USBD_SetPowerCtrl()
2) The functions relative with status are handled by:
   USBD_GetINTStatus(), USBD_GetEP0Status(), USBD_GetEPxStatus(),
   USBD_GetPowerCtrlStatus()
3) The functions relative with DMA operation are handled by:
   USBD_GetDMACConfig(), USBD_ConfigDMAC(), USBD_GetDMACStatus(),
   USBD_GetDMACMasterAddr(), USBD_SetDMACMasterAddr()
4) There are two special functions to access register in UDC2 module:
   USBD_ReadUDC2Reg(), USBD_WriteUDC2Reg()

11.2.3 Function Documentation

11.2.3.1 USBD_GetINTStatus

Get all the interrupt status of USBD.

Prototype:
USBD_INTStatus
USBD_GetINTStatus(void)

Parameters:
None

Description:
This function will get all the interrupt status of USBD.
For example, an UDC2 interrupt **USBD_INT_SETUP** at bit0, an UDC2AB interrupt **USBD_INT_SUSPEND_RESUME** at bit8, and so on.

Return:
The state of each interrupt source.
Refer to data structure of USBD_INTStatus for details.

11.2.3.2 USBD_SetINTMask

Set mask of the specified interrupt of USBD.

Prototype:
void
USBD_SetINTMask(USBD_INTSrc ***IntSrc***,
FunctionalState ***NewState***)

Parameters:
IntSrc: Specify the interrupt source, which can be set as:

- **USBD_INT_SETUP :** The int_setup signal of UDC2.

- **USBD_INT_STATUS_NAK :** The int_status_nak signal of UDC2.
- **USBD_INT_STATUS :** The int_status signal of UDC2.
- **USBD_INT_RX_ZERO :** The int_rx_zero signal of UDC2.
- **USBD_INT_SOF :** The int_sof signal of UDC2.
- **USBD_INT_EP0 :** The int_ep0 signal of UDC2.
- **USBD_INT_EP :** The int_ep signal of UDC2.
- **USBD_INT_NAK :** The int_nak signal of UDC2.

- **USBD_INT_SUSPEND_RESUME :**
  Interrupt generated each time the suspend_x signal of UDC2 changes.
- **USBD_INT_USB_RESET :**
  Interrupt generated when UDC2 has asserted the usb_reset signal.
- **USBD_INT_USB_RESET_END :**
  Interrupt generated when UDC2 has deasserted the usb_reset signal.
- **USBD_INT_MW_SET_ADD :**
  Interrupt generated when Master Write transfer address request.
- **USBD_INT_MW_END_ADD :**
  Interrupt generated when Master Write transfer finished.
- **USBD_INT_MW_TIMEOUT :**
  Interrupt generated when Master Write transfer timed out.
- **USBD_INT_MW_AHBERR :**
  Interrupt generated when AHB error occurred during the operation of Master Write transfer.
- **USBD_INT_MR_END_ADD :**
  Interrupt generated when Master Read transfer finished.
- **USBD_INT_MR_EP_DSET :**
  Interrupt generated when FIFO of EP for UDC2 Tx to be used for Master Read transfer becomes writable.
- **USBD_INT_MR_AHBERR :**
  Interrupt generated when AHB error occurred during the operation of Master Read transfer.
- **USBD_INT_UDC2_REG_READ :**
  Interrupt generated when read/write UDC2 registers completed.
- **USBD_INT_DMAC_REG_READ :**
  Interrupt generated when read/write DMAC special registers completed.
- **USBD_INT_MW_READERROR :**
  Interrupt generated when Endpoint read error occurred in Master Write.

NewState: Set the interrupt source mask state, which can be set as:
- **ENABLE :** Enable the interrupt specified by **IntSrc** above.
- **DISABLE:** Disable the interrupt specified by **IntSrc** above.

Description:
This function will set mask of the specified interrupt of USBD.
Unused interrupts must be disabled by calling this function.
The default settings for UDC2 interrupts(from **USBD_INT_SETUP** to **USBD_INT_NAK**) after power on reset are all "**ENABLE**".
The default settings for UDC2AB interrupts(from **USBD_INT_SUSPEND_RESUME** to **USBD_INT_MW_READERROR**) after power on reset are all "**DISABLE**".

Return:
None

11.2.3.3 USBD_ClearINT

Clear the specified interrupt flag of USBD.

Prototype:
void
USBD_ClearINT(USBD_INTSrc ***IntSrc***)

Parameters:
IntSrc: Specify the interrupt source, which can be set as:

- **USBD_INT_SETUP :** The int_setup signal of UDC2.
- **USBD_INT_STATUS_NAK :** The int_status_nak signal of UDC2.
- **USBD_INT_STATUS :** The int_status signal of UDC2.
- **USBD_INT_RX_ZERO :** The int_rx_zero signal of UDC2.
- **USBD_INT_SOF :** The int_sof signal of UDC2.
- **USBD_INT_EP0 :** The int_ep0 signal of UDC2.
- **USBD_INT_EP :** The int_ep signal of UDC2.
- **USBD_INT_NAK :** The int_nak signal of UDC2.

- **USBD_INT_SUSPEND_RESUME :**
  Interrupt generated each time the suspend_x signal of UDC2 changes.
- **USBD_INT_USB_RESET :**
  Interrupt generated when UDC2 has asserted the usb_reset signal.
- **USBD_INT_USB_RESET_END :**
  Interrupt generated when UDC2 has deasserted the usb_reset signal.
- **USBD_INT_MW_SET_ADD :**
  Interrupt generated when Master Write transfer address request.
- **USBD_INT_MW_END_ADD :**
  Interrupt generated when Master Write transfer finished.
- **USBD_INT_MW_TIMEOUT :**
  Interrupt generated when Master Write transfer timed out.
- **USBD_INT_MW_AHBERR :**
  Interrupt generated when AHB error occurred during the operation of Master Write transfer.
- **USBD_INT_MR_END_ADD :**
  Interrupt generated when Master Read transfer finished.
- **USBD_INT_MR_EP_DSET :**
  Interrupt generated when FIFO of EP for UDC2 Tx to be used for Master Read transfer becomes writable.
- **USBD_INT_MR_AHBERR :**
  Interrupt generated when AHB error occurred during the operation of Master Read transfer.
- **USBD_INT_UDC2_REG_READ :**
  Interrupt generated when read/write UDC2 registers completed.
- **USBD_INT_DMAC_REG_READ :**
  Interrupt generated when read/write DMAC special registers completed.
- **USBD_INT_POWERDETECT :**
  Interrupt generated when the status of the VBUSPOWER terminal is changed.
- **USBD_INT_MW_READERROR :**
  Interrupt generated when Endpoint read error occurred in Master Write.

Description:
This function will clear the specified interrupt flag of USBD.
Interrupt flag must be cleared by calling this function.

Return:
None

11.2.3.4 USBD_GetDMACConfig

Get the configuration of DMA Controller in USBD.

Prototype:
USBD_DMACConfig
USBD_GetDMACConfig(void)

Parameters:
None

Description:
This function will get the configuration of DMA Controllers in USBD, such as “Master Write Enable”, “Master Read Enable”, “Master Write Reset”, “Master Read Reset”.

Return:
The configuration of DMA Controllers in USBD.
Refer to data structure of USBD_DMACConfig for details.

11.2.3.5 USBD_ConfigDMAC

Configure the DMA Controller of USBD.

Prototype:
void
USBD_ConfigDMAC(USBD_DMACConfig ***Config***)

Parameters:
Config: Contain the configuration for DMA Controllers of USBD,
Refer to data structure of USBD_DMACConfig for details.

Description:
This function will configure the DMA Controllers of USBD, such as “enable Master Write ”, “enable Master Read”, “reset Master Write”, “reset Master Read”.

Return:
None

11.2.3.6 USBD_GetDMACStatus

Get the status of DMA Controller Master Operation.

Prototype:
USBD_DMACStatus
USBD_GetDMACStatus(void)

Parameters:
None

Description:
This function will get the status of DMA Controllers Master Operation.
For example, bit "**MW_Buf_Empty"** for "Master Write Buffer is Empty", bit "**MR_EP_DSet"** for "Data for Master Read Endpoint has been set".

Return:
The status of DMAC Master Operation.
Refer to data structure of USBD_DMACStatus for details.

11.2.3.7 USBD_ReadUDC2Reg

Read data from UDC2 registers which must be read through "UDC2 Read Request Register" of UDC2AB.

Prototype:
void
USBD_ReadUDC2Reg(uint32_t ***Addr***,
uint32_t * ***Data***)

Parameters:
Addr: The address of registers in UDC2 module,which can be set as:

- **UDC2_ADDR :** Address state and device address register.
- **UDC2_FRAME :** Frame register.
- **UDC2_COMMAND :** Command register to EP.
- **UDC2_BREQ_BMREQTYPE :**
  bRequeset-bmRequest type register of setup package.
- **UDC2_WVALUE :** wValue register of setup package.
- **UDC2_WINDEX :** wIndex register of setup package.
- **UDC2_WLENGTH :** wLength register of setup package.
- **UDC2_INT :**
  INT register for UDC2 interrupt signal and its mask bits.
- **UDC2_INTEP :**
  Contains flags for transmitting/receiving status of EPs (except for EP0).
- **UDC2_INTEP_MASK :** Mask settings for **UDC2_INTEP**.
- **UDC2_INTRX0 :**
  Flags to indicate Zero-Length data received at EPs.
- **UDC2_EPxMAXPACKETSIZE(x = 0 to 7) :** EPx_MaxPacketSize register.
- **UDC2_EPxSTATUS(x = 0 to 7) :** EPx status register.
- **UDC2_EPxDATASIZE(x = 0 to 7) :** EPx datasize register.
- **UDC2_EPxFIFO(x = 0 to 7) :** EPx FIFO register.
- **UDC2_INTNAK :**
  Contains flags to indicate the status of transmitting NAK at EPs (except for EP0).
- **UDC2_INTNAK_MASK :** Control mask settings for **UDC2_INTNAK**.

***Data*:** The pointer point to where the value of register will be stored. The UDC2 register only use low 16bits of an uint32_t type.

Description:
This function will read data from the specified UDC2 registers.
For example, the whole setup package from USB bus can be got by calling this function with parameter below one by one:
UDC2_BREQ_BMREQTYPE,
UDC2_WVALUE,

UDC2_WINDEX,
UDC2_WLENGTH

Return:
None

11.2.3.8 USBD_WriteUDC2Reg

Write data to UDC2 registers.

Prototype:
void
USBD_WriteUDC2Reg(uint32_t ***Addr***,
const uint32_t ***Data***)

Parameters:
Addr: The address of registers in UDC2 module, which can be set as:

- **UDC2_ADDR :** Address state register.
- **UDC2_FRAME :** Frame register.
- **UDC2_COMMAND :** Command to EP register.
- **UDC2_INT :**
  INT register for UDC2 interrupt signal and its mask bits.
- **UDC2_INTEP :**
  Contains flags for transmitting/receiving status of EPs (except for EP0).
- **UDC2_INTEP_MASK :** Control mask settings for **UDC2_INTEP**.
- **UDC2_INTRX0 :**
  Contains flags to indicate Zero-Length data received at EP.
- **UDC2_EP0MAXPACKETSIZE :** EP0_MaxPacketSize register.
- **UDC2_EP0FIFO :** EP0_FIFO register.
- **UDC2_EPxMAXPACKETSIZE (x = 1 to 7) :** EPx Max Packet Size register.
- **UDC2_EPxSTATUS (x = 0 to 7) :** EPx status register.
- **UDC2_EPxFIFO (x = 1 to 7) :** EPx FIFO register.
- **UDC2_INTNAK :**
  Contains flags to indicate the status of transmitting NAK at EPs (except for EP0).
- **UDC2_INTNAK_MASK :** Control mask settings for **UDC2_INTNAK**.

Data: The data which will be written to UDC2 register.

Description:
This function will write specified data to UDC2 registers which is specified by "***Addr*** " above.

Return:
None

11.2.3.9 USBD_GetDMACMasterAddr

Get the addresses of DMAC master operation.

Prototype:
USBD_DMACAddr
USBD_GetDMACMasterAddr(USBD_MasterMode ***MasterMode***)

Parameters:

MasterMode: Specify DMAC master addresses type, which can be set as:
- **USBD_MASTER_WRITE :** Master write.
- **USBD_MASTER_READ :** Master read.

Description:
This function will get the address registers relative with DMAC master operation, including “Master Write Start Address”, “Master Write End Address” and “Master Read Start Address”, “Master Read End Address”.

Return:
The addresses of DMAC Master Operation.
Refer to data structure of USBD_DMACAddr for details.

11.2.3.10 USBD_SetDMACMasterAddr

Set the addresses for DMAC master operation.

Prototype:
void
USBD_SetDMACMasterAddr(USBD_MasterMode ***MasterMode***,
USBD_DMACAddr ***Addr***)

Parameters:
MasterMode: Specify DMAC master addresses type,which can be set as:
- **USBD_MASTER_WRITE :** Master write.
- **USBD_MASTER_READ :** Master read.

Addr**:** The addresses for DMAC master operation.
Refer to data structure of USBD_DMACAddr for details.

Description:
This function will set the address registers relative with DMAC master operation, including “Master Write Start Address”, “Master Write End Address” and “Master Read Start Address”, “Master Read End Address”.

Return:
None

11.2.3.11 USBD_GetPowerCtrlStatus

Get the value of USBD power detect control.

Prototype:
USBD_PowerCtrl
USBD_GetPowerCtrlStatus(void)

Parameters:
None

Description:
This function will get the value of USBD power detect control register, for example, bits “USB_Reset”, “PHY_Suspend” and “PHY_Resetb”.

Return:
The status of USBD power detect control.

Refer to data structure of USBD_PowerCtrl for details.

11.2.3.12 USBD_SetPowerCtrl

Set the USBD power detect control.

Prototype:
void
USBD_SetPowerCtrl(USBD_PowerCtrl ***PowerCtrl***)

Parameters:
PowerCtrl: Contains the setting for USBD power detect control.
Refer to data structure of USBD_PowerCtrl for details.

Description:
This function will set the USBD power detect control register, for example, bits “USB_Reset”, “PHY_Suspend” and “PHY_Resetb”.

Return:
None

11.2.3.13 USBD_SetEPCMD

Send command to endpoints.

Prototype:
void
USBD_SetEPCMD(USBD_EPx ***EPx***,
USBD_EPCMD ***Cmd***)

Parameters:
EPx: Specify the target endpoint, which can be set as:
- **USBD_EP0 :** USBD Endpoint 0
- **USBD_EP1 :** USBD Endpoint 1
- **USBD_EP2 :** USBD Endpoint 2
- **USBD_EP3 :** USBD Endpoint 3
- **USBD_EP4 :** USBD Endpoint 4
- **USBD_EP5 :** USBD Endpoint 5
- **USBD_EP6 :** USBD Endpoint 6
- **USBD_EP7 :** USBD Endpoint 7

Cmd**:** The command which will be sent to endpoint, which can be set as:
when ***EPx*** is **USBD_EP0:**
- **USBD_CMD_SETUP_FIN :**
  The command should be issued when the DATA stage finishes or INT_STATUS_NAK was received.
- **USBD_CMD_EP_RESET:**
  The command for clearing the data and status of endpoints.
- **USBD_CMD_EP_STALL:**
  The command for setting the status of endpoints to "Stall".
- **USBD_CMD_ALL_EP_INVALID:**
  The command for setting the status of all endpoints other than EP0 to "Invalid".
- **USBD_CMD_USB_READY:**

The command for making connection with the USB cable. Issue this command at the point when communication with the host has become effective after confirming the connection with the cable.

- **USBD_CMD_SETUP_RECEIVED:**
  The command for informing UDC2 that the SETUP-Stage of a Control transfer was recognized. Issue this command after accepting the INT_SETUP interrupt and the request code was recognized.
- **USBD_CMD_EP_EOP:**
  The command for informing UDC2 that the transmit data has been written. Issue this command when transmitting data with byte size smaller than the maximum transfer bytes.
- **USBD_CMD_EP_FIFO_CLEAR:**
  The command for clearing the data of an endpoint.
- **USBD_CMD_EP_TX_0DATA:**
  The command for setting Zero-Length data to an endpoint. Issue this command when you want to transmit Zero-Length data.

when ***EPx*** is **USBD_EPy (y = 1 to 7) :**

- **USBD_CMD_SET_DATA0:**
  The command for clearing toggling of endpoints. While toggling is automatically updated by UDC2 in normal transfers, this command should be issued if it needs to be cleared by software.
- **USBD_CMD_EP_RESET:**
  The command for clearing the data and status of endpoints.
- **USBD_CMD_EP_STALL:**
  The command for setting the status of endpoints to "Stall".
- **USBD_CMD_EP_INVALID:**
  The command for setting the status of endpoints to "Invalid".
- **USBD_CMD_EP_DISABLE:**
  The command for making an endpoint disabled.
- **USBD_CMD_EP_ENABLE:**
  The command for making an endpoint disabled.
- **USBD_CMD_ALL_EP_INVALID:**
  The command for setting the status of all endpoints other than EP0 to "Invalid".
- **USBD_CMD_EP_EOP:**
  The command for informing UDC2 that the transmit data has been written. Issue this command when transmitting data with byte size smaller than the maximum transfer bytes.
- **USBD_CMD_EP_FIFO_CLEAR:**
  The command for clearing the data of an endpoint.
- **USBD_CMD_EP_TX_0DATA:**
  The command for setting Zero-Length data to an endpoint. Issue this command when you want to transmit Zero-Length data.

Description:
This function will send command to endpoints.

Return:
None

11.2.3.14 USBD_GetEP0Status

Get current EP0 status.

Prototype:
USBD_EP0Status
USBD_GetEP0Status(void)

Parameters:
None

Description:
This function will get current status of Endpoint0.
For example, if EP0 is in “Ready”, “Busy”, “Error”, “Stall” status. The information of “Data can be written into EP0_FIFO” can also be gotten.

Return:
Current Endpoint0 status.
Refer to data structure of USBD_EP0Status for details.

11.2.3.15 USBD_GetEPxStatus

Get current EPx status (x = 1 to 7)

Prototype:
USBD_EPxStatus
USBD_GetEPxStatus(USBD_EPx ***EPx***)

Parameters:
EPx: Specify the target endpoint, which can be set as:
- **USBD_EP1 :** USBD Endpoint 1
- **USBD_EP2 :** USBD Endpoint 2
- **USBD_EP3 :** USBD Endpoint 3
- **USBD_EP4 :** USBD Endpoint 4
- **USBD_EP5 :** USBD Endpoint 5
- **USBD_EP6 :** USBD Endpoint 6
- **USBD_EP7 :** USBD Endpoint 7

Description:
This function will get status for current endpoint x (x = 1 to 7).
For example, if EPx is in “Ready”, “Busy”, “Error”, “Stall” status, the transfer direction and transfer mode of EPx can also be gotten.

Return:
The status for the specified endpoint x (x = 1 to 7).
Refer to data structure of USBD_EPxStatus for details.

11.2.3.16 USBD_ConfigEPx

Configure EPx (x = 1 to 7).

Prototype:
void
USBD_ConfigEPx(USBD_EPx ***EPx***,
USBD_EPxConfig ***Config***)

Parameters:
EPx: Specify the target endpoint, which can be set as:
- **USBD_EP1 :** USBD Endpoint 1

- **USBD_EP2 :** USBD Endpoint 2
- **USBD_EP3 :** USBD Endpoint 3
- **USBD_EP4 :** USBD Endpoint 4
- **USBD_EP5 :** USBD Endpoint 5
- **USBD_EP6 :** USBD Endpoint 6
- **USBD_EP7 :** USBD Endpoint 7

Config: Contain the settting information for the specified EPx.
Refer to data structure of USBD_EPxConfig for details.

Description:
This function will configure the specified endpoint x (x = 1 to 7).
For example, we can configure the transfer direction and transfer mode of EPx.
It should be called when the standard request: Set_Configuration and Set_Interface are received.

Return:
None

11.2.4 Data Structure Description

11.2.4.1 USBD_DMACAddr

Data fields for this structure:

uint32_t
StartAddr : The start address for DMAC Master operation, which can be:
- **USBD_MW_START_ADDR** : Master Write Start Address
- **USBD_MR_START_ADDR** : Master Read Start Address

uint32_t
EndAddr: The end address for DMAC Master operation, which can be:
- **USBD_MW_END_ADDR** : Master Write End Address
- **USBD_MR_END_ADDR** : Master Read End Address

11.2.4.2 USBD_INTStatus

Data fields for this union:

uint32_t
All : To provide the easy interface to access all bits below.

Bit
uint32_t
Setup : ***1***
The int_setup signal of UDC2

uint32_t
Status_NAK : ***1***
The int_status_nak signal of UDC2

uint32_t
Status : ***1***

The int_status signal of UDC2

uint32_t
Rx_Zero : 1
The int_rx_zero signal of UDC2

uint32_t
SOF : 1
The int_sof signal of UDC2

uint32_t
EP0 : 1
The int_ep0 signal of UDC2

uint32_t
EP : 1
The int_ep signal of UDC2

uint32_t
NAK : 1
The int_nak signal of UDC2

uint32_t
Suspend_Resume : 1
Asserts 1 each time the suspend_x signal of UDC2 changes.
0: Status has not changed
1: Status has changed

uint32_t
USB_Reset : 1
Indicates whether UDC2 has asserted the usb_reset signal.
0: UDC2 has not asserted the usb_reset signal after this bit was cleared.
1: UDC2 has asserted the usb_reset signal.

uint32_t
USB_Reset_End : 1
Indicates whether UDC2 has deasserted the usb_reset signal.
0: UDC2 has not deasserted the usb_reset signal after this bit was cleared.
1: UDC2 has deasserted the usb_reset signal.

uint32_t
Reserved1 : 6
Place a 6bits gap to none used area. Read as undefined. Write as zero

uint32_t
MW_Set_Add : 1
Will be set to 1 when the data to be sent by Master Write transfer is set to the corresponding EP of Rx while the Master Write transfer is disabled.
0: Not detected
1: Master Write transfer address request

uint32_t
MW_End_Add : 1
Will be set to 1 when the Master Write transfer has finished.
0: Not detected

1: Master Write transfer finished

uint32_t
MW_TimeOut : ***1***
This status will be set to 1 when time-out has occurred during the operation of Master Write transfer.
0: Not detected
1: Master Write transfer timed out

uint32_t
MW_AHBErr : ***1***
This status will be set to 1 when the AHB error has occurred during the operation of Master Write transfer.After this interrupt has occurred, the Master Write transfer block needs to be reset by the mw_reset bit of DMAC Setting register.
0: Not detected
1: AHB error occurred

uint32_t
MR_End_Add : ***1***
Will be set to 1 when the Master Read transfer has finished.
0: Not detected
1: Master Read transfer finished

uint32_t
MR_EP_DSet : ***1***
Will be set to 1 when the FIFO of EP for UDC2 Tx to be used for Master Read transfer becomes writable (not full).
0: FIFO is not writable
1: FIFO is writable

uint32_t
MR_AHBErr : ***1***
This status will be set to 1 when the AHB error has occurred during the operation of Master Read transfer.After this interrupt has occurred, the Master Read transfer block needs to be reset by the mr_reset bit of DMAC Setting register.
0: Not detected
1: AHB error occurred

uint32_t
UDC2_Reg_Read : ***1***
Will be set to 1 when the UDC2 access executed by the setting of UDC2 Read Request register is completed and the value read to UDC2 Read Value register is set.Also set to 1 when Write access to the internal register of UDC2 is completed.
0: Not detected
1: Register read/write completed

uint32_t
DMAC_Reg_Read : ***1***
Will be set to 1 when the register access executed by the setting of DMAC Read Request register is completed and the value read to DMAC Read Value register is set.
0: Not detected
1: Register read completed

uint32_t
Reserved2 : ***2***
Place a 2bits gap to none used area. Read as undefined. Write as zero

uint32_t
PowerDetect : ***1***
When the status of the VBUSPOWER terminal changed, it is set to "1".
0: No Changed.
1: Status Changed.

uint32_t
MW_ReadError : ***1***
Will be set to 1 when the access to the endpoint has started Master Write transfer during the setting of common bus access (bus_sel bit of EPx_Status register is 0).
0: Not detected
1: Endpoint read error occurred in Master Write

uint32_t
Reserved3 : ***2***
Place a 2bits gap to none used area. Read as undefined. Write as zero

11.2.4.3 USBD_DMACConfig

Data fields for this union:

uint32_t
All : To provide the easy interface to access all bits below

Bit
uint32_t
MW_Enable : ***1***
Controls Master Write transfers. Enabling should be made when setting the transfer address is completed. It will be automatically disabled as the master transfer finishes. Since Master Write operations cannot be disabled with this register, use the mw_abort bit if the Master Write transfer should be stopped.
0: Disable
1: Enable

uint32_t
MW_Abort : ***1***
Controls Master Write transfers. Master Write operations can be stopped by setting 1 to this bit. When aborted during transfers, transfer of buffers for Master Write from UDC2 is interrupted and the mw_enable bit is cleared, stopping the Master Write transfer. Aborting completes when the mw_enable bit is disabled to 0 after setting this bit to 1.
0: No operation
1: Abort.

uint32_t
MW_Reset : ***1***
Initializes the Master Write transfer block. However, as the FIFOs of endpoints are not initialized, you need to access the Command register of UDC2 to initialize the corresponding endpoint separately from this reset. This reset

should be used after stopping the Master operation. This bit will be automatically cleared to 0 after being set to 1. Subsequent Master Write transfers should not be made until it is cleared.

0: No operation
1: Reset

uint32_t
Reserved1 : 1
Place a 1 bit gap to none used area. Read as undefined. Write as zero

uint32_t
MR_Enable : 1
Controls Master Read transfers. Enabling should be made when setting the transfer address is completed. It will be automatically disabled as the master transfer finishes. Since Master Read operations cannot be disabled with this register, use the mr_abort bit if the Master Read transfer should be stopped.

0: Disable
1: Enable

uint32_t
MR_Abort : 1
Controls Master Read transfers. Master Read operations can be stopped by setting 1 to this bit. When aborted during transfers, transfer of buffers for Master Read to UDC2 is interrupted and the mr_enable bit is cleared, stopping the Master Read transfer. Aborting completes when the mr_enable bit is disabled to 0 after setting this bit to 1.

0: No operation
1: Abort

uint32_t
MR_Reset : 1
Initializes the Master Read transfer block of UDC2AB. However, as the FIFOs of endpoints are not initialized, you need to access the Command register of UDC2 to initialize the corresponding endpoint separately from this reset. This reset should be used after stopping the Master operation. This bit will be automatically cleared to 0 after being set to 1. Subsequent Master Read transfers should not be made until it is cleared.

0: No operation
1: Reset

uint32_t
Reserved2 : 1 Place a 1 bit gap to none used area. Read as undefined. Write as zero

uint32_t
M_Burst_Type : 1
Selects the type of HBURST[2:0] when making a burst transfer in Master Write/Read transfers. The type of burst transfer made by UDC2AB is INCR8 (burst of 8 beat increment type). Accordingly, 0 (initial value) should be set in normal situation. However, in case INCR can only be used as the type of burst transfer based on the AHB specification of the system, set 1 to this bit. In that case, UDC2AB will make INCR transfer of 8 beat.

Please note the number of beat in burst transfers cannot be changed. Setting of this bit should be made in the initial setting of UDC2AB. The setting should not be changed after the Master Write/Read transfers started.

Note: UDC2AB does not make burst transfers only in Master Write/Read transfers. It combines burst transfers and single transfers. This bit affects the execution of burst transfers only.

0: INCR8
1: INCR

uint32_t
Reserved3 : 23
Place a 23bits gap to none used area. Read as undefined. Write as zero

11.2.4.4 USBD_DMACStatus

Data fields for this union:

uint32_t
All : To provide the easy interface to access all bits below.

Bit

uint32_t
***MW_EP_DSet*: 1**
This bit will be set to 1 when the data received is set to the Rx-EP of UDC2. It will turn to 0 when the entire data was read by the DMA for Master Write.

0: No data exists in the endpoint.
1: There is some data to be read in the endpoint

uint32_t
***MR_EP_DSet*: 1**
This bit will be set to 1 when the data to be transmitted is set to the Tx-EP of UDC2 by Master Read DMA transfer, making no room to write in the endpoint. It will turn to 0 when the data is transferred from UDC2 by the IN-Token from the host. While this bit is set to 0, DMA transfers to the endpoint can be made. (This bit is the eptx_dataset input signal with CLK_H synchronization.)

0: Data can be transferred into the endpoint.
1: There is no space to transfer data in the endpoint.

uint32_t
***MW_Buf_Empty*: 1**
Indicates whether or not the buffer for the Master Write DMA in UDC2AB is empty.

0: buffer contains some data.
1: buffer is empty

uint32_t
***MR_Buf_Empty*: 1**
Indicates whether or not the buffer for the Master Read DMA in UDC2AB is empty.

0: buffer contains some data.
1: buffer is empty

uint32_t
***MR_EP_Empty*: 1**
Indicates the endpoint for UDC2Rx is empty. Ensure that this bit is set to 1 when sending a NULL packet using the tx0 bit of UDC2 Setting register. (This bit is the eptx_empty input signal with CLK_H synchronization.)

0: the endpoint contains some data.

1: the endpoint is empty.

uint32_t
Reserved*:** ***27
Place a 27bits gap to none used area. Read as undefined. Write as zero

11.2.4.5 USBD_PowerCtrl

Data fields for this union:

uint32_t
All : To provide the easy interface to access all bits below

Bit
uint32_t
USB_Reset* :** ***1
The value of the usb_reset signal from UDC2 synchronized. (Read Only)
0: usb_reset = 0
1: usb_reset = 1

uint32_t
PW_Resetb* :** ***1
Software reset for UDC2AB. Setting this bit to 0 will make the PW_RESETB output pin asserted to 0. Resetting should be made while the master operation is stopped. Since this bit will not be automatically released, be sure to clear it.
0: Reset asserted
1: Reset deasserted

uint32_t
PW_Detect* :** ***1
Status of the VBUSPOWER input pin
0: USB bus disconnected
1: USB bus connected

uint32_t
PHY_Suspend*:** ***1
Setting this bit to 1 will make the PHYSUSPEND output signal asserted to 0 (CLK_H synchronization). It can be used as a pin for suspending PHY. Setting this bit to 1 makes the UDC2 register and DMAC Read Request register not accessible. It will be automatically cleared to 0 when resumed (when suspend_x of UDC2 is deasserted).
0: Not suspended
1: Suspended

uint32_t
Suspend_x*:** ***1
Detects the suspend signal (a value of the suspend_x signal from UDC2 synchronized).(Read Only).
0: Suspended (suspend_x = 0)
1: Unsuspended (suspend_x = 1)

uint32_t
PHY_Resetb*:** ***1
Setting this bit to 0 will make the PHYRESET output signal asserted to 1. The PHYRESET signal can be used to reset PHY. Since this bit will not be

automatically released, be sure to clear it to 1 after the specified reset time of PHY.
0: Reset asserted
1: Reset deasserted

uint32_t
PHY_Remote_Wakeup*: *1
This bit is used to perform the remote wakeup function of USB. Setting this bit to 1 makes it possible to assert the udc2_wakeup output signal (wakeup input pin of UDC2) to 1.
However, since setting this bit to 1 while no suspension is detected by UDC2 (when suspend_x = 1) will be ignored (not to be set to 1), be sure to set it only when suspension is detected. It will be automatically cleared to 0 when resuming the USB is completed (when suspend_x is deasserted).
0: No operation
1: Wakeup

uint32_t
Wakeup_En*: *1
Set this bit to '1' if you want the system (AHB end) to sleep to stop CLK_H when the USB is suspended. If this bit is set to 1, the WAKEUP signal will be asserted to 0 asynchronously when the suspended status is cancelled (suspend_x = 1) or the system is disconnected (VBUSPOWER = 0), making it available for resuming the system.
0: Do not assert the #WAKEUP signal
1: Assert the #WAKEUP signal

uint32_t
Reserved*: *24
Place a 24bits gap to none used area. Read as undefined. Write as zero

11.2.4.6 USBD_EP0Status

Data fields for this union:

uint32_t
All : To provide the easy interface to access all bits below

Bit
uint32_t
Reserved1*: *9
Place a 9bits gap to none used area. Read as undefined. Write as zero

uint32_t
Status*: *3
Indicates the present status of EP0. It will be cleared to "Ready" when the Setup-Token is received.
000: Ready (Indicates the status is normal)
001: Busy (To be set when returned "NAK" in the STATUS-Stage)
010: Error (To be set in case of CRC error in the received data, as well as when timeout has occurred after transmission of the data)
011: Stall (Returns "STALL" when data longer than the Length was requested in Control-RD transfers and the status will be set. It will be also set when "EP0-STALL" was issued by Command register.)
100 to 111: Reserved

uint32_t
Toggle*:** ***2
Indicates the present toggle value of EP0
00: DATA0
01: DATA1
10: Reserved
11: Reserved

uint32_t
Reserved2*:** ***1
Place a 1 bit gap to none used area. Read as undefined. Write as zero

uint32_t
EP0_Mask*:** ***1
Will be set to 1 after the Setup-Token is received. Will be cleared to 0 when the "Setup_Received" command is issued. No data will be written into the EP0_FIFO while this bit is 1.
0: Data can be written into EP0_FIFO
1: No data can be written into EP0_FIFO

11.2.4.7 USBD_EPxConfig, USBD_EPxStatus

Data fields for this union:

uint32_t
All : To provide the easy interface to access all bits below.

Bit
uint32_t
Num_MF* :** ***2
When the Isochronous transfer is selected, set how many times the transfer should be made in μ frames.
00: 1-transaction
01: 2-transaction
10: 3-transaction
11: Reserved

uint32_t
T_Type* :** ***2
Sets the transfer mode for this endpoint.
00: Control
01: Isochronous
10: Bulk
11: Interrupt

uint32_t
Reserved1* :** ***3
Place a 3bits gap to none used area. Read as undefined. Write as zero

uint32_t
Dir* :** ***1
Sets the direction of transfers for this endpoint.

0: OUT (Host-to-device)
1: IN (Device-to-host)
Note:
for EP1,3,5,7 must be set to ‘1’ as IN only.
for EP2,4,6 must be set to ‘0’ as OUT only.

uint32_t
Disable* :** ***1
Indicates whether transfers are allowed for EPx. If "Not Allowed," "NAK" will be always returned for the Token sent to this endpoint.
0: Allowed
1: Not Allowed

uint32_t
Status* :** ***3
Indicates the present status of EPx. By issuing EP_Reset from Command register, the status will be "Ready."
000: Ready (Indicates the status is normal)
001: Reserved
010: Error (To be set in case a receive error occurred in the data packet, or when timeout has occurred after transmission. However, it will not be set when "Stall" or "Invalid" has been set.)
011: Stall (To be set when "EP-Stall" was issued by Command register.)
100 to 110: Reserved

uint32_t
Toggle* :** ***2
Indicates the present toggle value of EPx.
00: DATA0
01: DATA1
10: DATA2
11: MDATA

uint32_t
Bus_Sel* :** ***1
Select the bus to access to the FIFO of EP1.
0: Common bus access
1: Direct access

uint32_t
Pkt_Mode* :** ***1
Selects the packet mode of EPx. Selecting the Dual mode makes it possible to retain two pieces of packet data for the EPx.
0: Single mode
1: Dual mode

12. WDT

12.1 Overview

The watchdog timer (WDT) is for detecting malfunctions (runaways) of the CPU caused by noises or other disturbances and remedying them to return the CPU to normal operation.

The WDT drivers API provide a set of functions to configure WDT, including such parameters as detection time, output if counter overflows, the state of WDT when enter IDLE mode and so on.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm365_wdt.c, with \Libraries/TX03_Periph_Driver\inc\tmpm365_wdt.h containing the API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- void WDT_SetDetectTime(uint32_t ***DetectTime***)
- void WDT_SetIdleMode(FunctionalState ***NewState***)
- void WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)
- void WDT_Init(WDT_InitTypeDef * ***InitStruct***)
- void WDT_Enable(void)
- void WDT_Disable(void)
- void WDT_WriteClearCode(void)

12.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) The Watchdog Timer basic function are handled by the WDT_SetDetectTime(),WDT_SetOverflowOutput(), WDT_Init(), WDT_Enable(), WDT_Disable(), and WDT_WriteClearCode() functions.
2) Run or stop the WDT counter when enter IDLE mode is handled by the WDT_SetIdleMode().

12.2.3 Function Documentation

12.2.3.1 WDT_SetDetectTime

Set detection time for WDT.

Prototype:
void
WDT_SetDetectTime(uint32_t ***DetectTime***)

Parameters:
***DetectTime*:** Set the detection time
This parameter can be one of the following values:

- **WDT_DETECT_TIME_EXP_15:** ***DetectTime*** is 2^15/fsys
- **WDT_DETECT_TIME_EXP_17:** ***DetectTime*** is 2^17/fsys
- **WDT_DETECT_TIME_EXP_19:** ***DetectTime*** is 2^19/fsys

- **WDT_DETECT_TIME_EXP_21:** ***DetectTime*** is 2^21/fsys
- **WDT_DETECT_TIME_EXP_23:** ***DetectTime*** is 2^23/fsys
- **WDT_DETECT_TIME_EXP_25:** ***DetectTime*** is 2^25/fsys

Description:
This function will set detection time for WDT.

Return:
None

12.2.3.2 WDT_SetIdleMode

Run or stop the WDT counter when the system enters IDLE mode.

Prototype:
void
WDT_SetIdleMode(FunctionalState ***NewState***)

Parameters:
NewState: Run or stop WDT counter.
This parameter can be one of the following values:
- **ENABLE**: Run the WDT counter.
- **DISABLE**. Stop the WDT counter.

Description:
This function will run the WDT counter when the system enters IDLE mode when ***NewState*** is **ENABLE**, and stop the WDT counter when the system enters IDLE mode when ***NewState*** is **DISABLE**.

Notes:
If CPU needs to enter the IDLE mode, this function must be called with appropriate parameter.

Return:
None

12.2.3.3 WDT_SetOverflowOutput

Set WDT to generate NMI interrupt or to reset when the counter overflows.

Prototype:
void
WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)

Parameters:
OverflowOutput: Select function of WDT when counter overflow.
This parameter can be one of the following values:
- **WDT_NMIINT**: Set WDT to generate NMI interrupt when counter overflow.
- **WDT_WDOUT**: Set WDT to generate reset when counter overflow.

Description:
This function will set WDT to generate NMI interrupt if the counter overflows when ***OverflowOutput*** is **WDT_NMIINT,** and set WDT to generate reset if the counter overflows when ***OverflowOutput*** is **WDT_WDOUT**.

Return:
None

12.2.3.4 WDT_Init

Initialize and configure WDT.

Prototype:
void
WDT_Init (WDT_InitTypeDef* ***InitStruct***)

Parameters:
InitStruct: The structure containing basic WDT configuration including detect time and WDT output when counter overflow. (Refer to "13.3.4 Data structure Description" for details)

Description:
This function will initialize and configure the WDT detection time and the output of WDT when the counter overflows. **WDT_SetDetectTime()** and **WDT_SetOverflowOutput()** will be called by it.

Return:
None

12.2.3.5 WDT_Enable

Enable the WDT function.

Prototype:
void
WDT_Enable(void)

Parameters:
None

Description:
This function will enable WDT.

Return:
None

12.2.3.6 WDT_Disable

Disable the WDT function.

Prototype:
void
WDT_Disable(void)

Parameters:
None

Description:

This function will disable WDT.

Return:
None

12.2.3.7 WDT_WriteClearCode

Write the clear code.

Prototype:
void
WDT_WriteClearCode (void)

Parameters:
None

Description:
This function will clear the WDT counter.

Return:
None

12.2.4 Data Structure Description

12.2.4.1 WDT_InitTypeDef

Data Fields:

uint32_t
DetectTime Set WDT detection time, which can be set as:

- **WDT_DETECT_TIME_EXP_15:** ***DetectTime*** is 2^15/fsys
- **WDT_DETECT_TIME_EXP_17:** ***DetectTime*** is 2^17/fsys
- **WDT_DETECT_TIME_EXP_19:** ***DetectTime*** is 2^19/fsys
- **WDT_DETECT_TIME_EXP_21:** ***DetectTime*** is 2^21/fsys
- **WDT_DETECT_TIME_EXP_23:** ***DetectTime*** is 2^23/fsys
- **WDT_DETECT_TIME_EXP_25:** ***DetectTime*** is 2^25/fsys

uint32_t
OverflowOutput Select the action when the WDT counter overflows, which can be set as:

- **WDT_WDOUT:** Set WDT to generate reset when the counter overflows.
- **WDT_NMIINT:** Set WDT to generate NMI interrupt when the counter overflows.