TOSHIBA

TOSHIBA TX00 Peripheral Driver User Guide (TMPM066/067/068)

Ver 1.0

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M066UG-01xE

RESTRICTIONS ON PRODUCT USE

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

TOSHIBA

Index

1. Introduction

TOSHIBA TX00 Peripheral Driver is a set of drivers for all peripherals found on the TOSHIBA TX00 series microcontrollers. TMPM06x(see Note below) Peripheral Driver is an important part of TOSHIBA TX00 Peripheral Driver, which is designed for TMPM06x series MCU.

TOSHIBA TX00 Peripheral Driver contains a collection of macros, data types, and structures for each peripheral.

The design goals of TOSHIBA TMPM06x(see Note below) Peripheral Driver:
- Completely written in C except the start-up routine and where not possible
- Cover all the peripherals on MCU

Note: TMPM06x stands for TMPM066/M067/M068 hereafter.

2. Organization of TOSHIBA TX00 Peripheral Driver

/Libraries
This folder contains all CMSIS files and TMPM06x Peripheral Drivers.

/Libraries/ TX00_CMSIS
This folder contains the device peripheral access layer of TMPM06x CMSIS files.

/Libraries/TX00_Periph_Driver
This folder contains all the source code of the drivers, the core of TOSHIBA TMPM06x Peripheral Driver.

/Libraries/TX00_Periph_Driver/inc
This folder contains all the header files of TMPM06x Peripheral Drivers for each peripheral.

/Libraries/TX00_Periph_Driver/src
This folder contains all the source files of TMPM06x Peripheral Drivers for each peripheral.

/Project
This folder contains template project and examples for using TMPM06x Peripheral Driver.

/Project/Template
This folder contains template project of TOSHIBA TMPM06x Peripheral Driver.

/Project/Examples
This folder contains a set of examples for using TMPM06x Peripheral Driver

/Utilities/TMPM06x-EVAL
This folder contains the configuration and driver files for hardware resources (e.g. led, key) on TMPM06x boards.

3. ADC

3.1 Overview

This device has 8 channels 10-bit A/D converter, the main functions include:

• Start by an internal or external timer trigger

• Fixed channel/scan mode

• Single/repeat mode

• AD monitoring 2ch

The ADC API provides a set of functions for using the TMPM06x ADC module. 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 TX00_Periph_Driver\src\tmpm06x_adc.c(*), with TX00_Periph_Driver\inc\tmpm06x_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 ***Conversion_Time***, uint32_t ***Prescaler_Output***)
- void ADC_Start(void)
- void ADC_SetScanMode(FunctionalState ***NewState***)
- void ADC_SetRepeatMode(FunctionalState ***NewState***)
- void ADC_SetINTMode(uint32_t ***INTMode***)
- ADC_State ADC_GetConvertState(void)
- void ADC_SetInputChannel(uint32_t ***InputChannel***)
- void ADC_SetChannelScanMode(ADC_ChannelScanMode ***ScanMode***)
- void ADC_SetIdleMode(FunctionalState ***NewState***)
- void ADC_SetVref(FunctionalState ***NewState***)
- void ADC_SetInputChannelTop(uint32_t ***TopInputChannel***)
- void ADC_StartTopConvert(void)
- void ADC_SetMonitor(uint8_t ***ADCMPx***, FunctionalState ***NewState***)
- void ADC_SetResultCmpReg(uint8_t ***ADCMPx***, uint32_t ***ResultComparison***)
- void ADC_SetMonitorINT(uint8_t ***ADCMPx***, ADC_ComparisonState ***NewState***)
- void ADC_SetHWTrg(uint32_t ***HwSource***, FunctionalState ***NewState***)
- void ADC_SetHWTrgTop(uint32_t ***HwSource***, FunctionalState ***NewState***)
- ADC_ResultTypeDef ADC_GetConvertResult (uint32_t ***ADREGx***)
- void ADC_SetCmpValue(uint8_t ***ADCMPx***, uint16_t ***value***)
- 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_SetChannelScanMode(), ADC_SetVref(), ADC_SetInputChannelTop(), ADC_SetMonitor(), ADC_SetResultCmpReg(), ADC_SetMonitorINT(), ADC_SetHWTrg(), ADC_SetHWTrgTop(), ADC_SetCmpValue().
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_SetDMAReq() and ADC_SetIdleMode() handle other specified functions.

3.2.3 Function Documentation

3.2.3.1 ADC_SWReset

Software reset ADC function.

Prototype:

void

ADC_SWReset(void)

Parameters:

None

Description:

This function will software reset ADC.

***Note:**

A software reset initializes other bits. Re-setting a mode register is needed.

Return:

None

3.2.3.2 ADC_SetClk

Set A/D conversion time and prescaler output.

Prototype:

void

ADC_SetClk(uint32_t ***Conversion_Time***,

uint32_t ***Prescaler_Output***)

Parameters:

Conversion_Time: Select ADC sample hold time.

This parameter can be one of the following values:

- **ADC_CONVERSION_35_CLOCK:** 35.5 conversion clock
- **ADC_CONVERSION_42_CLOCK:** 42 conversion clock
- **ADC_CONVERSION_68_CLOCK:** 68 conversion clock
- **ADC_CONVERSION_81_CLOCK:** 81 conversion clock

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_6:** fc / 6
- **ADC_FC_DIVIDE_LEVEL_8:** fc / 8
- **ADC_FC_DIVIDE_LEVEL_12:** fc / 12
- **ADC_FC_DIVIDE_LEVEL_16:** fc / 16
- **ADC_FC_DIVIDE_LEVEL_24:** fc / 24
- **ADC_FC_DIVIDE_LEVEL_48:** fc / 48
- **ADC_FC_DIVIDE_LEVEL_96:** fc / 96

Description:

This function will set ADC conversion time by ***Conversion_Time*** and prescaler output by ***Prescaler_Output***.

***Note:**

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.

A clock count is required to satisfy the condition that described below.

VREFH AVDD	Conversion time
2.7 to 3.6V	05—1 us or longer
2.3 to 3.6V	21—3 us or longer

Return:

None

3.2.3.3 ADC_Start

Start ADC function.

Prototype:

void

ADC_Start(void)

Parameters:

None

Description:

This function will start AD conversion.

***Note:**

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_SetChannelScanMode()** for the details.

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

Return:

None

3.2.3.4 ADC_SetScanMode

Set ADC scan mode.

Prototype:

void

ADC_SetScanMode(FunctionalState ***NewState***)

Parameters:

NewState: Specify ADC scan mode

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable or disable ADC scan mode by ***NewState*** setting.

Return:

None

3.2.3.5 ADC_SetRepeatMode

Set ADC repeat mode.

Prototype:

void

ADC_SetRepeatMode(FunctionalState ***NewState***)

Parameters:

NewState: Specify ADC repeat mode

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will enable or disable ADC repeat mode by ***NewState*** setting.

Return:

None

3.2.3.6 ADC_SetINTMode

Set ADC interrupt mode in fixed channel repeat conversion mode.

Prototype:

void

ADC_SetINTMode(uint32_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_4:** Generate interrupt once every 4 conversions.
- **ADC_INT_CONVERSION_8:** Generate interrupt once every 8 conversions.

Description:

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

***Note:**

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_GetConvertState

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

Prototype:

ADC_State

ADC_GetConvertState(void)

Parameters:

None

Description:

This function will read AD conversion completion / busy flag (both normal and top-priority).

Return:

A union with the state of AD conversion:

NormalBusy(Bit 0) : '1' means normal AD is converting

NormalComplete (Bit 1): '1' means normal AD conversion is completed.

TopBusy(Bit 2) : '1' means top-priority AD is converting

TopComplete (Bit 3): '1' means top-priority AD conversion is completed.

3.2.3.8 ADC_SetInputChannel

Set ADC input channel.

Prototype:

void

ADC_SetInputChannel(uint32_t ***InputChannel***)

Parameters:

InputChannel: Analog input channel, and the input channel also related with other settings.

This parameter can be one of the following values:

ADC_AN_0, ADC_AN_1, ADC_AN_2, ADC_AN_3,

ADC_AN_4, ADC_AN_5, ADC_AN_6, ADC_AN_7.

Description:

This function will specify ADC input channel by ***InputChannel*** setting. And the input channels also relate with mode setting.

In fixed channel mode (**ADC_SetScanMode(DISABLE)**), user can only select one of the 8 channels.

In channel scan mode (**ADC_SetScanMode(ENABLE)**), the input channels is different for 4 channel scan mode

(**ADC_SetChannelScanMode(ADC_SCAN_4CH)**)

and 8 channel scan mode (**ADC_SetChannelScanMode(ADC_SCAN_8CH)**).

***Note:**

Set channel scan mode: **ADC_SetScanMode(ENABLE)**.

Set 4 channel scan mode mode:

ADC_SetChannelScanMode(ADC_SCAN_4CH).

Set 8 channel scan mode mode:

ADC_SetChannelScanMode(ADC_SCAN_8CH).

Return:

None

3.2.3.9 ADC_SetChannelScanMode

Set ADC operation for scanning.

Prototype:

void

ADC_SetChannelScanMode(ADC_ChannelScanMode ***ScanMode***)

Parameters:

ScanMode: Specify operation mode for channel scanning.

The parameter can be one of the following values:

ADC_SCAN_4CH, ADC_SCAN_8CH

Description:

This function will specify different channel scan mode by ***ScanMode*** setting.

***Note:**

This function setting will change the input channel setting **ADC_SetInputChannel()**, please refer to the description of **ADC_SetInputChannel()** for details.

Return:

None

3.2.3.10 ADC_SetIdleMode

Set ADC operation in IDLE mode.

Prototype:

void

ADC_SetIdleMode(FunctionalState ***NewState***)

Parameters:

NewState: Specify AD conversion in IDLE mode.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will specify ADC enable or disable in system IDLE mode by ***NewState*** setting.

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 Vref on or off by ***NewState***.

***Note:**

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

Return:

None

3.2.3.12 ADC_SetInputChannelTop

Set ADC top-priority conversion analog input channel select.

Prototype:

void

ADC_SetInputChannelTop(uint32_t ***TopInputChannel***)

Parameters:

TopInputChannel: Analog input channel for top-priority conversion.

This parameter can be one of the following values:

ADC_AN_0, ADC_AN_1, ADC_AN_2, ADC_AN_3,

ADC_AN_4, ADC_AN_5, ADC_AN_6, ADC_AN_7.

Description:

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

***Note:**

Only one channel of **ADC_AN_0**~**ADC_AN_7** can be selected as top-priority conversion input each time.

Return:

None

3.2.3.13 ADC_StartTopConvert

Start ADC top-priority conversion.

Prototype:

void

ADC_StartTopConvert(void)

Parameters:

None

Description:

This function will start top-priority conversion.

***Note:**

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

Return:

None

3.2.3.14 ADC_SetMonitor

Set ADC monitor function.

Prototype:

void

ADC_SetMonitor(uint8_t ***ADCMPx***,

FunctionalState ***NewState***)

Parameters:

ADCMPx: Select AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

NewState: Specify ADC monitor function

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will specify ADC monitor channel by ***ADCMPx*** setting and specify ADC monitor function enable or disable by ***NewState*** setting.

Return:

None

3.2.3.15 ADC_SetResultCmpReg

Set ADC result comparison register or comparison register.

Prototype:

void

ADC_SetResultCmpReg(uint8_t ***ADCMPx***,

uint32_t ***ResultComparison***)

Parameters:

ADCMPx: Select AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

ResultComparison: Select the AD conversion result storage register that is to be compared with the comparison register if ADC monitor function is enabled.

The parameter can be one of the following values:

ADC_REG_0, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**

ADC_REG_4, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**

ADC_REG_SP

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will specify ADC monitor channel by ***ADCMPx*** setting and specify AD conversion result storage register that is to be compared with the comparison register by ***ResultComparison*** setting.

Return:

None

3.2.3.16 ADC_SetMonitorINT

Set ADC monitor interrupt.

Prototype:

void

ADC_SetMonitorINT(uint8_t ***ADCMPx***,

ADC_ComparisonState ***NewState***)

Parameters:

ADCMPx: Selet AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

NewState: Specify ADC monitor function interrupt condition

This parameter can be one of the following values:

ADC_COMPARISON_SMALLER or **ADC_COMPARISON_LARGER**.

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will specify ADC monitor interrupt by ***ADCMPx*** setting and specify ADC monitor function interrupt condition by ***NewState*** setting.

Return:

None

3.2.3.17 ADC_SetHWTrg

Hardware trigger for normal ADC enable or disable and Hardware Source for activating normal ADC setting.

Prototype:

void

ADC_SetHWTrg(uint32_t ***HwSource***,

FunctionalState ***NewState***)

Parameters:

HwSource: Hardware source for activating normal ADC.

This parameter can be one of the following values:

- **ADC_EXT_TRG:** External trigger
- **ADC_MATCH_TB_0:** Match with timer channel 0 register

NewState: enable or disable hardware source for activating normal ADC

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will specify hardware source for activating normal ADC setting by ***HwSource*** setting and specify hardware trigger for normal ADC monitor function enable or disable by ***NewState*** setting.

This function also has relation with TB0 setting.

***Note:**

The TMPM06x disables the external trigger used for hardware activation. Therefore do not use the function as **ADC_SetHWTrg(ADC_EXT_TRG, NewState)**.

If AD conversion is executed with the match triggers <ADHTG>(Refer to TMPM06x datasheet) and <HADHTG>(Refer to TMPM06x datasheet) of a 16-bit timer set to "1" by using a source for triggering hardware, A/D conversion can be activated at specified intervals by performing three steps shown below when the timer is idle:

- Select a source for triggering hardware.
- Enable hardware activation of AD conversion.
- Start the timer.

Do not make a top-priority AD conversion setting and a normal AD conversion setting simultaneously. Do not use functions **ADC_SetHWTrg(ADC_MATCH_TB_0, ENABLE)** and **ADC_SetHWTrgTop(ADC_MATCH_TB_1, ENABLE)** simultaneously.

Return:

None

3.2.3.18 ADC_SetHWTrgTop

Hardware trigger for top-priority ADC enable or disable and hardware source for activating top-priority ADC setting.

Prototype:

void

ADC_SetHWTrgTop(uint32_t ***HwSource***,
FunctionalState ***NewState***)

Parameters:

HwSource: Hardware source for activating top-priority ADC.

This parameter can be one of the following values:

- **ADC_EXT_TRG:** External trigger
- **ADC_MATCH_TB_1:** Match with timer channel 1 register

NewState: enable or disable hardware source for activating top-priority ADC

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will specify hardware source for activating top-priority ADC setting by ***HwSource*** setting and specify hardware trigger for top-priority ADC monitor function enable or disable by ***NewState*** setting.

This function also has relation with TB1 setting.

***Note:**

The TMPM06x disables the external trigger used for hardware activation.

Therefore do not use the function as

ADC_SetHWTrgTop(ADC_EXT_TRG, NewState).

Do not make a top-priority AD conversion setting and a normal AD conversion setting simultaneously. Do not use functions

ADC_SetHWTrg(ADC_MATCH_TB_0, ENABLE) and

ADC_SetHWTrgTop(ADC_MATCH_TB_1, ENABLE) simultaneously.

Return:

None

3.2.3.19 ADC_GetConvertResult

Read ADC register's result storage flag state, overrun state and result value.

Prototype:

ADC_ResultTypeDef

ADC_GetConvertResult(uint32_t ***ADREGx***)

Parameters:

ADREGx: Select ADC result register.

The parameter can be one of the following values:

ADC_REG_0, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**

ADC_REG_4, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**

ADC_REG_SP

Description:

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

***Note:**

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.

AD conversion result is stored in ***ADRGEx*** in different ADC mode as below table.

Table: Analog Input Channels and Related A/D Conversion Result Registers

Analog input channel (port A)	A/D conversion result register			
	Conversion modes other than shown to the right	Fixed channel repeat conversion mode (every one conversion)	Fixed channel repeat conversion mode (every four conversions)	Fixed channel repeat conversion mode (every eight conversions)
ADC_AN_0	ADC_REG_0	ADC_REG_0 fixed	ADC_REG_0—> ADC_REG_3	ADC_REG_0-> ADC_REG_7
ADC_AN_1	ADC_REG_1			
ADC_AN_2	ADC_REG_2			
ADC_AN_3	ADC_REG_3			
ADC_AN_4	ADC_REG_4			
ADC_AN_5	ADC_REG_5			
ADC_AN_6	ADC_REG_6			
ADC_AN_7	ADC_REG_7			

The ADC mode setting, please refer to relate APIs.

For top-priority AD conversion, the result is stored in ADC_REG_SP.

Return:

ADC result structure:

1. The state of ADC result stored state, which can be
   - **DONE**: AD conversion complete and result stored.
   - **BUSY**: During conversion.
2. The state of normal AD conversion complete, which can be
   - **ADC_NO_OVERRUN**: No Conversion over run.
   - **ADC_OVERRUN**: Conversion over run.
3. ADC result value.

3.2.3.20 ADC_SetCmpValue

Set ADC comparison register value.

Prototype:

```
void
ADC_SetCmpValue(uint8_t ADCMPx,
                uint16_t value)
```

Parameters:

ADCMPx: Select AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

value: The value set to ADC compare register, max is 0x03ff

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will set the ADC compare register value of ADC monitor channel which specify by ***ADCMPx*** setting.

The max setting value should not be larger than 0x03ff for ADC is only 10-bit.

***Note:**

ADC monitor function setting process:

1. **ADC_SetResultCmpReg(ADCMPx, ResultComparison)**
2. **ADC_SetCmpValue(ADCMPx, value)**
3. **ADC_SetMonitorINT(ADCMPx, ResultComparison)**

4. **ADC_SetMonitor(ADCMPx, ENABLE)**

After AD conversion finished, if the condition match **ADC_SetMonitorINT()** setting, ADC monitor interrupt will occurs(The interrupt enable, please refer to interrupt chapter of TMPM06x datasheet).

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:** Specify normal AD conversion DMA activaation factor. (Triggered by INTAD)
- **ADC_DMA_REQ_TOP:** Specify top-priority AD conversion DMA activation factor.(Triggered by INTADHP)
- **ADC_DMA_REQ_MONITOR1:** Specify AD monitor function 0 DMA activation factor.(Triggered by INTADM0)
- **ADC_DMA_REQ_MONITOR2:** Specify AD monitor function 1 DMA activation factor.(Triggered by INTADM1)

NewState: Specify AD conversion DMA activation factor.

The parameter can be one of the following values:

- **ENABLE:** Enable specified DMA activation factor.
- **DISABLE:** Disable specified DMA activation factor

Description:

This function will enable or disable DMA activation factor for normal or top-priority AD conversion.

Return:

None

3.2.4 Data Structure Description

3.2.4.1 ADC_ResultTypeDef

Data Fields:

WorkState

ADCResultStored specifies ADC result storage flag, which can be set as:

- **BUSY**, which means that ADC result has not been stored to the result register;
- **DONE**, which means that which ADC result has been stored to the result register.

ADC_OverrunState

ADCOverrunState specifies ADC overrun flag, which can be set as:

- **ADC_NO_OVERRUN**, which means that ADC is not overrun;
- **ADC_OVERRUN**, which means that ADC is overrun.

uint16_t

ADCResultValue specifies ADC result value,

3.2.4.2 ADC_State

Data Fields for this union:

uint32_t

All specifies AD conversion state.

Bit Fields:

uint32_t

NormalBusy(Bit 0)	Normal A/D conversion busy flag (MOD0<ADBFN>). ‘1’ means conversion is busy

uint32_t

NormalComplete (Bit 1)	Normal AD conversion complete flag (MOD0<EOCFN>). ‘1’ means conversion is completed

uint32_t

TopBusy(Bit 2)	Top-priority A/D conversion busy flag (MOD2<ADBFHP>). ‘1’ means conversion is busy

uint32_t

TopComplete (Bit 3)	Top-priority AD conversion complete flag (MOD2<EOCFHP>). ‘1’ means conversion is completed

uint32_t

Reserved (Bit 4 to Bit 31) reserved.

4. CG

4.1 Overview

The CG API provides a set of functions for using the TMPM06x 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 TX00_Periph_Driver\src\tmpm06x_cg.c, with TX00_Periph_Driver\inc\tmpm06x_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.

EHCLKIN : Clock input from the X1 pins

EHOSC : Output clock from the external high-speed oscillator

IHOSC : Output clock from the internal high-speed oscillator.(for SYS)

FOSCHI : Clock specified by CGOSCCR<HOSCON>

fosc : Clock specified by CGOSCCR<OSCSEL>

fpll : Clock multiplied by PLL.

fc : Clock specified by CGPLLSEL<PLLSEL> (high-speed clock).

fgear : Clock specified by CGSYSCR<GEAR[2:0]>.

fsys : Clock specified by CGSYSCR<GEAR[2:0]>.(system clock)

fperiph : Clock specified by CGSYSCR<FPSEL[2:0]>.

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)
- Result CG_SetPhiT0Level(CG_DivideLevel ***DividePhiT0FromFc***)
- CG_DivideLevel CG_GetPhiT0Level(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)
- void CG_SetFoscSrc(CG_FoscSrc ***Source***)
- CG_FoscSrc CG_GetFoscSrc(void)
- void CG_SetSTBYMode(CG_STBYMode ***Mode***)
- CG_STBYMode CG_GetSTBYMode(void)
- Result CG_SetFcSrc(CG_FcSrc ***Source***)
- CG_FcSrc CG_GetFcSrc(void)
- void CG_SetProtectCtrl(FunctionalState ***NewState***)
- void CG_SetADCClkSupply(FunctionalState NewState)
- void CG_SetFcPeriphA(uint32_t ***Periph***, FunctionalState ***NewState***)
- void CG_SetFcPeriphB(uint32_t ***Periph***, FunctionalState ***NewState***)
- void CG_SetFcOptional(uint32_t ***Periph***, FunctionalState ***NewState***)

4.2.2 Detailed Description

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

1) One group of APIs are in charge of clock selection, such as: CG_SetFgearLevel(), CG_GetFgearLevel(),CG_SetPhiT0Level(), CG_GetPhiT0Level(),CG_SetWarmUpTime(), CG_StartWarmUp(), CG_GetWarmUpState(),CG_SetFPLLValue(), CG_GetFPLLValue(),CG_SetPLL(), CG_GetPLLState(),CG_SetFoscSrc(), CG_GetFoscSrc(),CG_SetFcSrc(),CG_GetFcSrc(),CG_SetProtectCtrl().

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

3) The other APIs control clock supply for peripherals: CG_SetADCClkSupply(),CG_SetFcPeriphA(), CG_SetFcPeriphB(),CG_SetFcOptional ().

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_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**.**

4.2.3.4 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.

4.2.3.5 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_INT_HIGH**: internal high-speed oscillator is selected as timer source.
- **CG_WARM_UP_SRC_OSC_EXT_HIGH**: external high-speed oscillator is selected as timer source.

Time:

Number of warm-up cycle. It is between 0x0000 and 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 16MHz, and set warm-up time 5ms. */

So value = (warm-up time to set) / (input frequency cycle(s)) = 5ms / (1/16MHz) = 3125cycle = 0xC350.

Round lower 4 bit off, set 0xC35 to CGOSCCR<WUPT[11:0]>

Return:

None.

4.2.3.6 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

4.2.3.7 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_HIGH, 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.

4.2.3.8 CG_SetFPLLValue

Set PLL multiplying value

Prototype:

Result

CG_SetFPLLValue(uint32_t ***NewValue***)

Parameters:

NewValue:

- **CG_8M_MUL_12_FPLL:**
  Input clock 8MHz, output clock 96MHz (12 multiplying)
- **CG_10M_MUL_8_FPLL:**
  Input clock 10MHz, output clock 80MHz (8 multiplying)
- **CG_12M_MUL_8_FPLL:**
  Input clock 12MHz, output clock 96MHz (8 multiplying)
- **CG_16M_MUL_6_FPLL:**
  Input clock 16MHz, output clock 96MHz (6 multiplying)

Description:

This function sets PLL multiplying value.

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

4.2.3.9 CG_GetFPLLValue

Get the value of PLL setting.

Prototype:

uint32_t

CG_GetFPLLValue(void)

Parameters:

None

Description:

This function will get the PLL multiplying value.

If the other value is read from the register,it means the value is reserved.

Return:

The source of PLL multiplying value

- **CG_8M_MUL_12_FPLL:**
  Input clock 8MHz, output clock 96MHz (12 multiplying)
- **CG_10M_MUL_8_FPLL:**
  Input clock 10MHz, output clock 80MHz (8 multiplying)
- **CG_12M_MUL_8_FPLL:**
  Input clock 12MHz, output clock 96MHz (8 multiplying)
- **CG_16M_MUL_6_FPLL:**
  Input clock 16MHz, output clock 96MHz (6 multiplying)

4.2.3.10 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.

4.2.3.11 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.

4.2.3.12 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

4.2.3.13 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.

4.2.3.14 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

4.2.3.15 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.

4.2.3.16 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_FPLL**: fc source will be set to fpll

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.

4.2.3.17 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_FPLL: fc source is set to fpll.

4.2.3.18 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

4.2.3.19 CG_SetADCClkSupply

Enable or disable supplying clock fsys for ADC.

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

Parameters:
NewState: New state of clock fsys supply setting for ADC.
This parameter can be one of the following values:

- **ENABLE :** Enable ADC clock suppply
- **DISABLE :** Disable ADC clock suppply

Description:
This function will enable or disable supplying clock fsys for ADC.

Return:
None

4.2.3.20 CG_SetFcPeriphA

Enable or disable supplying clock fsys to peripheries.

Prototype:

void

CG_SetFcPeriphA(uint32_t ***Periph***,

FunctionalState ***NewState***)

Parameters:

Periph: The target peripheral that CG supplies clock fc for

This parameter can be one of the following values or their combination:

- **CG_FC_PERIPH_PORTH:** Clock control for PORT H
- **CG_FC_PERIPH_PORTJ:** Clock control for PORT J
- **CG_FC_PERIPH_TMRB0_3:** Clock control for TMRB ch0-3
- **CG_FC_PERIPH_TMRB4_6:** Clock control for TMRB ch4-6
- **CG_FC_PERIPH_TMR16A:** Clock control for TMR16A
- **CG_FC_PERIPH_I2C0:** Clock control for I2C ch0
- **CG_FC_PERIPH_SIO0:** Clock control for SIO ch0
- **CG_FC_PERIPH_TSPI:** Clock control for TSPI
- **CG_FC_PERIPH_DMAC:** Clock control for DMAC
- **CG_FC_PERIPH_ADC:** Clock control for ADC
- **CG_FC_PERIPH_USBD:** Clock control for USBD
- **CG_FC_PERIPH_TMRD:** Clock control for TMRD
- **CG_FC_PERIPHA_ALL:** ALL clock control

NewState

- **ENABLE**: Enable supplying clock fsys to peripheries.
- **DISABLE**: Disable supplying clock fsys to peripheries.

Description:

This function enables or disables supplying clock fsys to peripheries

Return:

None

4.2.3.21 CG_SetFcPeriphB

Enable or disable supplying clock fsys to peripheries.

Prototype:

void

CG_SetFcPeriphB(uint32_t ***Periph***,

FunctionalState ***NewState***)

Parameters:

Periph: The target peripheral that CG supplies clock fc for

This parameter can be one of the following values or their combination:

- **CG_FC_PERIPH_TMRB7:** Clock control for TMRB ch7
- **CG_FC_PERIPH_SIO1:** Clock control for SIO ch1
- **CG_FC_PERIPH_WDT:** Clock control for WDT
- **CG_FC_PERIPH_I2C1:** Clock control for I2C ch1
- **CG_FC_PERIPHB_ALL:** ALL clock control

NewState

- **ENABLE**: Enable supplying clock fsys to peripheries.
- **DISABLE**: Disable supplying clock fsys to peripheries.

Description:

This function enables or disables supplying clock fsys to peripheries

Return:

None

4.2.3.22 CG_SetFcOptional

Enable or disable supplying clock fsys to peripheries.

Prototype:

void

CG_SetFcOptional(uint32_t ***Periph***,

FunctionalState ***NewState***)

Parameters:

Periph: The target peripheral that CG supplies clock fc for

This parameter can be one of the following values or their combination:

- **CG_FPLL_PERIPH_TMRD:** Clock control for TMRD
- **CG_EHCLKSEL_8_24_48MHZ:** ECLK selection of CGRST
- **CG_USBSEL_PLL_CLOCKIN:** Clock control for USB clock selection
- **CG_USBENA_USB:** Clock control for USB
- **CG_FPLL_OPTIONAL_ALL:** ALL clock control

NewState

- **ENABLE**: Enable supplying clock fsys to peripheries.
- **DISABLE**: Disable supplying clock fsys to peripheries.

Description:

This function enables or disables supplying clock fsys to peripheries

Return:

None

4.2.4 Data Structure Description

None

5. INTIFAO INTIFSD AOREG

5.1 Overview

TOSHIBA TMPM06x has INTIF registers, which is for releasing from low-power consumption mode.

The interrupt control registers are allocated based on the factor. The table below shows the interrupt control registers of the MCU.

Management No	Address	INTIFAO Interrupt Control Register(AOBUS)	INTIFSD Interrupt Control register (IOBUS)
000	X + 0x00		–
~	~		–
015	X + 0x0F		–
016	Y + 0x10		STOP1INT_016 or IDLEINT_016
~	~		~
031	Y + 0x1F		STOP1INT_031 or IDLEINT_031
032	X + 0x20	STOP2INT_032	–
~	~	~	–
095	X + 0x5F	STOP2INT_095	–
096	Y + 0x60		STOP1INT_096 or IDLEINT_096
~	~		~
255	Y + 0xFF		STOP1INT_255 or IDLEINT_255

X=0x4003_8000 (AO bus area), Y=0x400F_4E00 (IO bus area)

The INTIF driver is CMSIS V4.00 compliant software, and has common function interface, which can be easily re-used by user.

Details on the driver interface, including APIs and relative data structure will list in this TMPM06x CMSIS INTIF driver specification.

5.2 API Functions

5.2.1 Function List

- void INTIFAO_SetSTBYReleaseINTSrc(INTIFAO_INTSrc INTSource, INTIFAO_INTActiveState ActiveState, FunctionalState NewState)
- INTIFAO_INTActiveState INTIFAO_GetSTBYReleaseINTState(INTIFAO_INTSrc INTSource);
- void INTIFAO_ClearINTReq(INTIFAO_INTSrc INTSource)
- INTIFSD_NMIFactor INTIFSD_GetNMIFlag(void)
- AO_ResetFlag AO_GetResetFlag(void)
- AO_ResetFlag1 AO_GetResetFlag1(void)

5.2.2 Detailed Description

The INTIFAO INTIFSD AOREG APIs can be broken into four groups by function:

4) The 2nd group of APIs handle settings of standby modes: INTIFAO_SetSTBYReleaseINTSrc (),INTIFAO_GetSTBYReleaseINTState ( ).

5) The other APIs control clock supply for peripherals: INTIFAO_GetSTBYReleaseINTState (),INTIFSD_GetNMIFlag (),AO_GetResetFlag (),AO_GetResetFlag1 ().

5.2.3 Function Documentation

5.2.3.1 INTIFAO_SetSTBYReleaseINTSrc

Set the INT source for releasing low power mode.

Prototype:

void

INTIFAO_SetSTBYReleaseINTSrc(INTIFAO_INTSrc ***INTSource***,
INTIFAO_INTActiveState ***ActiveState***,
FunctionalState ***NewState***)

Parameters:

INTSource: select the INT source for releasing standby mode

This parameter can be one of the following values:

- **INTIFAO_INT_SRC_0** : INT0
- **INTIFAO_INT_SRC_1** : INT1
- **INTIFAO_INT_SRC_2** : INT2
- **INTIFAO_INT_SRC_3** : INT3
- **INTIFAO_INT_SRC_4** : INT4
- **INTIFAO_INT_SRC_5** : INT5
- **INTIFAO_INT_SRC_I2CS :** I2CS interrupt
- **INTIFAO_INT_SRC_USBWKUP :** USBD interrupt

ActiveState: select the active state for release trigger.

For **NTIFAO_INT_SRC_I2CS, INTIFAO_INT_SRC_USBWKUP**, this parameter can only be

- **INTIFAO_INT_ACTIVE_STATE_RISGING**: active on rising edge

For the other interrupt source, this parameter can be one of the following values:

- **INTIFAO_INT_ACTIVE_STATE_L**: active on low level
- **INTIFAO_INT_ACTIVE_STATE_H**: active on high level
- **INTIFAO_INT_ACTIVE_STATE_FALLING**: active on falling edge
- **INTIFAO_INT_ACTIVE_STATE_RISGING**: active on rising edge
- **INTIFAO_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

5.2.3.2 INTIFAO_GetSTBYReleaseINTState

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

Prototype:

INTIFAO_INT_ActiveState

INTIFAO_GetSTBYReleaseINTSrc(INTIFAO_INTSrc ***INTSource***)

Parameters:

INTSource: select the release INT source

This parameter can be one of the following values:

- **INTIFAO_INT_SRC_0** : INT0
- **INTIFAO_INT_SRC_1** : INT1
- **INTIFAO_INT_SRC_2** : INT2
- **INTIFAO_INT_SRC_3** : INT3
- **INTIFAO_INT_SRC_4** : INT4
- **INTIFAO_INT_SRC_5** : INT5
- **INTIFAO_INT_SRC_I2CS :** I2CS interrupt
- **INTIFAO_INT_SRC_USBWKUP :** USBD interrupt

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:

INTIFAO_INT_ACTIVE_STATE_FALLING: active on falling edge

INTIFAO_INT_ACTIVE_STATE_RISING: active on rising edge

INTIFAO_INT_ACTIVE_STATE_BOTH_EDGES: active on both edges

INTIFAO_INT_ACTIVE_STATE_INVALID: invalid

5.2.3.3 INTIFAO_ClearINTReq

Clears the input INT request.

Prototype:

void

INTIFAO_ClearINTReq(INTIFAO_INTSrc ***INTSource***)

Parameters:

INTSource: select the release INT source.

This parameter can be one of the following values:

- **INTIFAO_INT_SRC_0** : INT0
- **INTIFAO_INT_SRC_1** : INT1
- **INTIFAO_INT_SRC_2** : INT2
- **INTIFAO_INT_SRC_3** : INT3
- **INTIFAO_INT_SRC_4** : INT4
- **INTIFAO_INT_SRC_5** : INT5
- **INTIFAO_INT_SRC_I2CS :** I2CS interrupt
- **INTIFAO_INT_SRC_USBWKUP :** USBD interrupt

Description:

This function will clear the INT request for releasing standby mode.

Return:

None

5.2.3.4 INTIFSD_GetNMIFlag

Get the NMI flag that shows who triggered NMI

Prototype:

INTIFSD_NMI_Factor

INTIFSD_GetNMIFlag (void)

Parameters:

None

Description:

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

Return:

NMI value:

DetectLowVoltage (Bit 16) means generated when detect low voltage by LVD

DetectOverVoltage (Bit 17) means generated when detect over voltage by LVD

WDT (Bit 18) means generated from WDT

5.2.3.5 AO_GetResetFlag

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

Prototype:

AO_ResetFlag

AO_GetResetFlag(void)

Parameters:

None

Description:

This function gets the reset flag showing what triggered reset.

Return:

Reset flag:

PowerOnReset (Bit0) means reset from power on reset

PinReset (Bit3) means reset from RESET

LVDReset (Bit5) means rest from LVD

5.2.3.6 AO_GetResetFlag1

Get the reset flag1 that shows the trigger of reset and clear the reset flag1

Prototype:

AO_ResetFlag1

AO_GetResetFlag1(void)

Parameters:

None

Description:

This function gets the reset flag1 showing what triggered reset.

Return:

Reset flag:

DebugReset (Bit0) means reset from SYSRESETREQ

WDTReset (Bit3) means reset from WDT

5.2.4 Data Structure Description

5.2.4.1 INTIFSD_NMIFactor

Data Fields:

uint32_t

All specifies NMI source generation state.

Bit Fields:

uint32_t

Reserved1 (Bit0~bit15) Reserved

uint32_t

DetectLowVoltage(Bit 16) means generated when detect low voltage by LVD.

uint32_t

DetectOverVoltage (Bit 17) means generated when detect over voltage by LVD.

uint32_t

Reserved2 (Bit18~bit31) Reserved

5.2.4.2 AO_ResetFlag

Data Fields:

Uint8_t

All specifies reset source.

Bit Fields:

Uint8_t

PowerOnReset (Bit0) Reset from power on reset

Uint8_t

Reserved1 (Bit1~bit2) Reserved

Uint8_t

ResetPinBit3) Reset from RESET pin

Uint8_t

Reserved2 (Bit4) Reserved

Uint8_t

LVDReset(Bit5) Rest by LVD

Uint8_t

Reserved3 (Bit6~bit7) Reserved

5.2.4.3 AO_ResetFlag1

Data Fields:

Uint8_t

All specifies reset source.

Bit Fields:

Uint8_t

DebugReset(Bit0) Reset from SYSRESETREQ

Uint8_t

Reserved1 (Bit1) Reserved

Uint8_t

WDTReset(Bit2) Reset from WDT

Uint8_t

Reserved2 (Bit3~bit7) Reserved

6. uDMAC

6.1 Overview

TMPM06x incorporates 1 units of built-in DMA controller.

The main functions for one unit are shown below:

Functions	Features		Descriptions
Channels	32 channels		-
Start trigger	Start by Hardware		DMA requests from peripheral functions
	Start by Software		Specified by DMAxChnlSwRequest register
Priority	Between channels	ch0 (high priority) > … > ch31 (high priority) > ch0 (Normal priority) > … > ch31 (Normal priority)	High-priority can be configured by DMAxChnlPriority-Set register
Transfer data size	8/16/32bit		Can be specified source and destination independently
The number of transfer	1 to 4095 times		-
Address	Transfer source address	Increment / fixed	Transfer source address and destination address can be selected to increment or fixed.
	transfer destination address	Increment / fixed	
Endian	Little Endian		-
Transfer type	Peripheral (register) → memory Memory → peripheral (register) Memory → memory		If you select memory to memory, hardware start for DMA start up is not supported. Refer to the DMACxConfiguration register for more information.
Interrupt function	Transfer end interrupt Error interrupt		Output for each unit
Transfer mode	Basic mode Automatic request mode Ping-pong mode Memory scatter / gather mode Peripheral scatter / gather mode		-

The uDMAC API provides a set of functions for using the TMPM06x uDMAC modules. It includes uDMAC transfer type set, channel set, mask set, primary/alternative data area set, channel priority, initialize data filling and so on.

This driver is contained in TX00_Periph_Driver\src\tmpm06x_udmac.c, with TX00_Periph_Driver\inc\tmpm06x_udmac.h containing the API definitions for use by applications.

***Note:** In this document, DMAC means uDMAC.

6.2 API Functions

6.2.1 Function List

- FunctionalState DMAC_GetDMACState(void)
- void DMAC_Enable(void)
- void DMAC_Disable(void)
- void DMAC_SetPrimaryBaseAddr(uint32_t ***Addr***)
- uint32_t DMAC_GetBaseAddr(DMAC_PrimaryAlt ***PriAlt***)
- void DMAC_SetSWReq(uint8_t ***Channel***)
- void DMAC_SetTransferType(DMAC_Channel ***Channel***,
  DMAC_TransferType ***Type***)
- DMAC_TransferType DMAC_GetTransferType(DMAC_Channel ***Channel***)
- void DMAC_SetMask(uint8_t Channel ,
  FunctionalState NewState)
- FunctionalState DMAC_GetMask(uint8_t Channel)
- void DMAC_SetChannel(uint8_t Channel ,
  FunctionalState NewState)
- FunctionalState DMAC_GetChannelState(uint8_t Channel)
- void DMAC_SetPrimaryAlt(uint8_t Channel
  DMAC_PrimaryAlt PriAlt)
- DMAC_PrimaryAlt DMAC_GetPrimaryAlt(uint8_t Channel)
- void DMAC_SetChannelPriority(uint8_t Channel ,
  DMAC_Priority Priority)
- DMAC_Priority DMAC_GetChannelPriority(uint8_t Channel)
- void DMAC_ClearBusErr(void)
- Result DMAC_GetBusErrState(void)
- void DMAC_FillInitData(uint8_t Channel ,
  DMAC_InitTypeDef * InitStruct)
- DMAC_Flag DMAC_GetINTFlag(void)
- DMACB_Flag DMACB_GetINTFlag(void)
- DMACC_Flag DMACC_GetINTFlag(void)

6.2.2 Detailed Description

Functions listed above can be divided into six parts:

1) uDMAC configuration by DMAC_SetTransferType(), DMAC_GetTransferType(), DMAC_SetMask(),DMAC_GetMask(),DMAC_SetChannel(),DMAC_GetChannelState(),DMAC_SetPrimaryAlt(),DMAC_GetPrimaryAlt(),DMAC_SetChannelPriority(),DMAC_GetChannelPriority().
2) uDMAC enable/disable by DMAC_GetDMACState(), DMAC_Enable(), DMAC_Disable().
3) uDMAC software trigger by DMAC_SetSWReq().
4) uDMAC bus error by DMAC_ClearBusErr(), DMAC_GetBusErrState().
5) uDMAC control data area filled by: DMAC_FillInitData(), DMAC_SetPrimaryBaseAddr(), DMAC_GetBaseAddr().
6) uDMAC factor flag by DMAC_GetINTFlag(), DMACB_GetINTFlag(), DMACC_GetINTFlag(),

6.2.3 Function Documentation

NOTE: For the parameter 'Channel' of all functions, if there isn't special explanation, the sentence 'Channel**: Select channel**'' will follow the content below:

Channel: Select channel.

The parameter can be one of the following values:

- **DMAC_UART0_RX:** UART0 **r**eception occurs an interrupt
- **DMAC_UART0_TX:** UART0 transmission occurs an interrupt
- **DMAC_UART1_RX:** UART1 reception occurs an interrupt
- **DMAC_UART1_TX:** UART1 transmission occurs an interrupt
- **DMAC_I2C0_RX: I**2C0 reception occurs an interrupt
- **DMAC_I2C0_TX:** I2C0 transmission occurs an interrupt
- **DMAC_I2C1_TX:** I2C1 reception occurs an interrupt
- **DMAC_I2C1_RX:** I2C1 transmission occurs an interrupt
- **DMAC_TSPI_RX:** TSPI reception occurs an interrupt
- **DMAC_TSPI_TX:** TSPI transmission occurs an interrupt
- **DMAC_TMRB0_CMP_MATCH:**TMRB0 compare match occurs an interrupt
- **DMAC_TMRB1_CMP_MATCH:**TMRB1 compare match occurs an interrupt
- **DMAC_TMRB2_CMP_MATCH:**TMRB2 compare match occurs an interrupt
- **DMAC_TMRB3_CMP_MATCH:**TMRB3 compare match occurs an interrupt
- **DMAC_TMRB4_CMP_MATCH:**TMRB4 compare match occurs an interrupt
- **DMAC_TMRB5_CMP_MATCH:**TMRB5 compare match occurs an interrupt
- **DMAC_TMRB6_CMP_MATCH:**TMRB6 compare match occurs an interrupt
- **DMAC_TMRB7_CMP_MATCH:**TMRB7 compare match occurs an interrupt
- **DMAC_TMRB0_INPUT_CAP0:** TMRB0 input capture 0 occurs an interrupt
- **DMAC_TMRB1_INPUT_CAP0:** TMRB1 input capture 0 occurs an interrupt
- **DMAC_TMRB2_INPUT_CAP0:** TMRB2 input capture 0 occurs an interrupt
- **DMAC_TMRB3_INPUT_CAP0:** TMRB3 input capture 0 occurs an interrupt
- **DMAC_TMRB4_INPUT_CAP0:** TMRB4 input capture 0 occurs an interrupt
- **DMAC_TMRB5_INPUT_CAP0:** TMRB5 input capture 0 occurs an interrupt
- **DMAC_TMRB6_INPUT_CAP0:** TMRB6 input capture 0 occurs an interrupt
- **DMAC_TMRB7_INPUT_CAP0:** TMRB7 input capture 0 occurs an interrupt
- **DMAC_TOP_PRIORITY_ADC:** ADC completion occurs an interrupt
- **DMAC_ADC_COMPLETION:** ADC completion occurs an interrupt

6.2.3.1 DMAC_GetDMACState

Get the state of specified DMAC unit.

Prototype:

FunctionalState

DMAC_GetDMACState(void)

Parameters:

None

Description:

This function will get the state of specified DMAC unit.

Return:

- **DISABLE :** The DMAC unit is disabled
- **ENABLE :** The DMAC unit is enabled

6.2.3.2 DMAC_Enable

Enable the DMA circuit.

Prototype:

void

DMAC_Enable(void)

Parameters:

None

Description:

This function will enable the DMA circuit.

Return:

None

6.2.3.3 DMAC_Disable

Disable the DMA circuit.

Prototype:

void

DMAC_Disable(void)

Parameters:

None

Description:

This function will disable the DMA circuit.

Return:

None

6.2.3.4 DMAC_SetPrimaryBaseAddr

Set the base address of the primary data of the DMA circuit.

Prototype:

void

DMAC_SetPrimaryBaseAddr(uint32_t ***Addr***)

Parameters:

None

Addr: The base address of the primary data, bit0 to bit9 must be 0.

Description:

This function will set the base address of the primary data of the DMA circuit.

Return:

None

6.2.3.5 DMAC_GetBaseAddr

Get the primary/alternative base address of the DMA circuit.

Prototype:

uint32_t

DMAC_GetBaseAddr(DMAC_PrimaryAlt ***PriAlt***)

Parameters:

None

PriAlt: Select base address type

This parameter can be one of the following values:

- **DMAC_PRIMARY :** Get primary base address
- **DMAC_ALTERNATE :** Get alternative base address

Description:

This function will get the primary/alternative base address of the DMA circuit.

Return:

The base address of primary/alternative data

6.2.3.6 DMAC_SetSWReq

Set software transfer request to the specified channel of the DMA circuit.

Prototype:

void

DMAC_SetSWReq(uint8_t ***Channel***)

Parameters:

None

Channel: Select channel.

Description:

This function will set software transfer request to the specified channel by ***Channel*** of the DMA circuit.

Return:

None

6.2.3.7 DMAC_SetTransferType

Set transfer type to the specified channel of the DMAC circuit.

Prototype:

void

DMAC_SetTransferType(uint8_t ***Channel***,

DMAC_TransferType ***Type***)

Parameters:

Channel: Select channel.

Type: Select transfer type.

This parameter can be one of the following values:

- **DMAC_BURST :** Single transfer is disabled, only burst transfer request can be used
- **DMAC_SINGLE :** Single transfer is enabled

Description:

This function will set transfer type to the specified channel of the DMAC circuit.

Return:

None

6.2.3.8 DMAC_GetTransferType

Get transfer type setting for the specified channel of the DMAC circuit

Prototype:

DMAC_TransferType

DMAC_GetTransferType(uint8_t ***Channel***)

Parameters:

Channel: Select channel.

Description:

This function will get transfer type setting for the specified channel of the DMAC circuit.

Return:

The transfer type with DMAC_TransferType type:

- **DMAC_BURST :** Single transfer is disabled, only burst transfer request can be used
- **DMAC_SINGLE :** Single transfer is enabled

6.2.3.9 DMAC_SetMask

Set mask for the specified channel of the DMA circuit.

Prototype:

void

DMAC_SetMask(uint8_t ***Channel*** ,

FunctionalState ***NewState***)

Parameters:

Channel: Select channel.

NewState: Clear or set the mask to enable or disable the DMA channel.

This parameter can be one of the following values:

- **ENABLE :** The DMA channel mask is cleared, DMA request is enable(valid)
- **DISABLE :** The DMA channel is masked, DMA request is disable(invalid)

Description:

This function will set mask for the specified channel of the DMA circuit.

Return:

None

6.2.3.10 DMAC_GetMask

Get mask setting for the specified channel of the DMA circuit.

Prototype:

FunctionalState

DMAC_GetMask(uint8_t ***Channel***)

Parameters:

Channel: Select channel.

Description:

This function will get mask setting for the specified channel of the DMA circuit.

Return:

The inverted mask setting:

- **ENABLE :** The DMA channel mask is cleared, DMA request is enable(valid)
- **DISABLE :** The DMA channel is masked, DMA request is disable(invalid)

6.2.3.11 DMAC_SetChannel

Enable or disable the specified channel of the DMA circuit.

Prototype:

void

DMAC_SetChannel(uint8_t ***Channel*** ,

FunctionalState ***NewState***)

Parameters:

Channel: Select channel.

NewState: Enable or disable the DMA channel.

This parameter can be one of the following values:

- **ENABLE :** The DMA channel will be enabled
- **DISABLE :** The DMA channel will be disabled

Description:

This function will enable or disable the specified channel of the DMA circuit. by ***NewState***.

Return:

None

6.2.3.12 DMAC_GetChannelState

Get the enable/disable setting for specified channel of the DMA circuit.

Prototype:

FunctionalState

DMAC_GetChannelState(uint8_t ***Channel***)

Parameters:

Channel: Select channel.

Description:

This function will get the enable/disable setting for specified channel of the DMA circuit.

Return:

The enable/disable setting for channel:

- **ENABLE :** The DMA channel is enabled
- **DISABLE :** The DMA channel is disabled

6.2.3.13 DMAC_SetPrimaryAlt

Set to use primary data or alternative data for specified channel of the DMA circuit.

Prototype:

void

DMAC_SetPrimaryAlt(uint8_t ***Channel*** ,

DMAC_PrimaryAlt ***PriAlt***)

Parameters:

Channel: Select channel.

PriAlt: Select primary data or alternative data for channel specified by 'ChannelA' above.

This parameter can be one of the following values:

- **DMAC_PRIMARY:** Channel will use primary data
- **DMAC_ALTERNATE:** Channel will use alternative data

Description:

This function will set to use primary data or alternative data for specified channel of the DMA circuit.

Return:

None

6.2.3.14 DMAC_GetPrimaryAlt

Get the setting of the using of primary data or alternative data for specified channel of the DMA circuit.

Prototype:

DMAC_PrimaryAlt

DMAC_GetPrimaryAlt(uint8_t ***Channel***)

Parameters:

Channel: Select channel.

Description:

This function will get the setting of the using of primary data or alternative data for specified channel of the DMA circuit.

Return:

The setting of the using of primary data or alternative data:

- **DMAC_PRIMARY:** Channel is using primary data
- **DMAC_ALTERNATE:** Channel is using alternative data

6.2.3.15 DMAC_SetChannelPriority

Set the priority for specified channel of the DMA circuit.

Prototype:

void

DMAC_SetChannelPriority(uint8_t ***Channel*** ,

DMAC_Priority ***Priority***)

Parameters:

Channel: Select channel.

Priority: Select Priority.

This parameter can be one of the following values:

- **DMAC_PRIOTIRY_NORMAL:** Normal priority.
- **DMAC_PRIOTIRY_HIGH:** High priority.

Description:

This function will set the priority for specified channel of the DMA circuit.

Return:

None

6.2.3.16 DMAC_GetChannelPriority

Get the priority setting for specified channel of the DMA circuit.

Prototype:

DMAC_Priority

DMAC_GetChannelPriority(uint8_t ***Channel***)

Parameters:

Channel: Select channel.

Description:

This function will get the priority setting for specified channel of the DMA circuit

Return:

The priority setting of channel:

- **DMAC_PRIOTIRY_NORMAL:** Normal priority.
- **DMAC_PRIOTIRY_HIGH:** High priority.

6.2.3.17 DMAC_ClearBusErr

Clear the bus error of the DMA circuit.

Prototype:

void

DMAC_ClearBusErr(void)

Parameters:

None

Description:

This function will clear the bus error of the DMA circuit.

Return:

None

6.2.3.18 DMAC_GetBusErrState

Get the bus error state of the DMA circuit.

Prototype:

Result

DMAC_GetBusErrState(void)

Parameters:

None

Description:

This function will get the bus error state of the DMA circuit.

Return:

The bus error state:

- **SUCCESS:** No bus error.
- **ERROR :** There is error in bus.

6.2.3.19 DMAC_FillInitData

Fill the DMA setting data of specified channel of the DMAC circuit to RAM.

Prototype:

void

DMAC_FillInitData(uint8_t ***Channel*** ,

DMAC_InitTypeDef * ***InitStruct***)

Parameters:

Channel: Select channel.

InitStruct: The structure contains the DMA setting values.

Description:

This function will fill the DMA setting data of specified channel of the DMAC circuit to RAM.

Return:

None

6.2.4 Data Structure Description

6.2.4.1 DMAC_InitTypeDef

Data fields:

uint32_t

SrcEndPointer: The final address of data source.

uint32_t

DstEndPointer: The final address of data destination.

DMAC_CycleCtrl

Mode: Set operation mode,

which can be:

- **DMAC_INVALID:** Invalid, DMA will stop the operation
- **DMAC_BASIC:** Basic mode
- **DMAC_AUTOMATIC:** Automatic request mode
- **DMAC_PINGPONG:** Ping-pong mode
- **DMAC_MEM_SCATTER_GATHER_PRI:** Memory scatter/gather mode (primary data)
- **DMAC_MEM_SCATTER_GATHER_ALT:** Memory scatter/gather mode (alternative data)
- **DMAC_PERI_SCATTER_GATHER_PRI:** Peripheral memory scatter/ gather mode (primary data)
- **DMAC_PERI_SCATTER_GATHER_ALT:** Peripheral memory scatter/ gather mode (alternative data)

DMAC_Next_UseBurst

NextUseBurst: Specifies whether to set "1" to the register DMAxChnlUseburstSet<chnl_useburst_set> bit to use burst transfer at the end of the DMA transfer using alternative data in the peripheral scatter/gather mode.

which can be:

- **DMAC_NEXT_NOT_USE_BURST:** Do not change the value of <chnl_useburst_set>.
- **DMAC_NEXT_USE_BURST:** Sets <chnl_useburst_set> to "1"

uint32_t

TxNum: Set the actual number of transfers. Maximum is 1024.

DMAC_Arbitration

ArbitrationMoment: Specifies the arbitration moment(R_Power).

After the specified numbers of transfers, an existence of a transfer request is checked. If there is a high-priority request, the control is switched to high-priority channel.

DMAC_BitWidth

SrcWidth: Set source bit width,

which can be:

- **DMAC_BYTE:** Data size of transfer is 1 byte.
- **DMAC_HALF_WORD:** Data size of transfer is 2 bytes.
- **DMAC_WORD:** Data size of transfer is 4 bytes

DMAC_IncWidth

SrcInc: Set increment of the source address,

which can be:

- **DMAC_INC_1B:** Address increment 1 byte.
- **DMAC_INC_2B:** Address increment 2 bytes.
- **DMAC_INC_4B:** Address increment 4 bytes.
- **DMAC_INC_0B:** Address does not increase

DMAC_BitWidth

DstWidth: Set destination bit width,

which can be:

- **DMAC_BYTE:** Data size of transfer is 1 byte
- **DMAC_HALF_WORD:** Data size of transfer is 2 bytes
- **DMAC_WORD:** Data size of transfer is 4 bytes

DMAC_IncWidth

DstInc: Set increment of the destination address,

which can be:

- **DMAC_INC_1B:** Address increment 1 byte
- **DMAC_INC_2B:** Address increment 2 bytes
- **DMAC_INC_4B:** Address increment 4 bytes
- **DMAC_INC_0B:** Address does not increase

6.2.4.2 DMAC_Flag

Data Fields for this union:

uint32_t

All The flag of DMA interrupt.

Bit Fields:

uint32_t

UART0Reception (Bit 0) the flag of UART0 reception occurs an interrupt.
'1' means occurs an interrupt

uint32_t

UART0Transmission (Bit 1) the flag of UART0 transmission occurs an interrupt
'1' means occurs an interrupt

uint32_t

UART1Reception (Bit 2) the flag of UART1 reception occurs an interrupt
'1' means occurs an interrupt

uint32_t

UART1Transmission (Bit 3) the flag of UART1 transmission occurs an interrupt
'1' means occurs an interrupt

uint32_t

I2C0Rx (Bit 4) the flag of I2C0 reception occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

I2C0Tx (Bit 5) the flag of I2C0 transmission occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

I2C1Rx (Bit 6) the flag of I2C1 reception occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

I2C1Tx (Bit 7) the flag of I2C1 transmission occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TSPIReception (Bit 8) the flag of TSPI reception occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TSPITransmission (Bit 9) the flag of TSPI transmission occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB0CompareMatch (Bit 10) the flag of TMRB0 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB1CompareMatch (Bit 11) the flag of TMRB1 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB2CompareMatch (Bit 12) the flag of TMRB2 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB3CompareMatch (Bit 13) the flag of TMRB3 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB4CompareMatch (Bit 14) the flag of TMRB4 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB5CompareMatch (Bit 15) the flag of TMRB5 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB6CompareMatch (Bit 16) the flag of TMRB6 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB7CompareMatch (Bit 17) the flag of TMRB7 compare match occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB0CompareMatch (Bit 18) the flag of TMRB0 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB1CompareMatch (Bit 19) the flag of TMRB1 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB1CompareMatch (Bit 19) the flag of TMRB1 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB2CompareMatch (Bit 20) the flag of TMRB2 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB3CompareMatch (Bit 21) the flag of TMRB3 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB4CompareMatch (Bit 22) the flag of TMRB4 input capture 0 occurs an interrupt

uint32_t

TMRB5CompareMatch (Bit 23) the flag of TMRB5 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB6CompareMatch (Bit 24) the flag of TMRB6 input capture 0 occurs an interrupt

‘1’ means occurs an interrupt

uint32_t

TMRB7CompareMatch (Bit 25) the flag of TMRB7 input capture 0 occurs an interrupt

'1' means occurs an interrupt

uint32_t

Reserved (Bit 16 to Bit 30) reserved.

uint32_t

TOPADCCompletion (Bit 31) the flag of TOP ADC completion occurs an interrupt

'1' means occurs an interrupt

uint32_t

ADCCompletion (Bit 32) the flag of ADC completion occurs an interrupt

'1' means occurs an interrupt

7. FC

7.1 Overview

TMPM06x device contains flash memory, the flash size is 128Kbytes.

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 refers to the MCU data sheet.

This driver is contained in \Libraries\TX00_Periph_Driver\src\tmpm06x_fc.c with \Libraries\TX00_Periph_Driver \inc\ tmpm06x_fc.h containing the API definitions for use by applications.

7.2 API Functions

7.2.1 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)
- FunctionalState FC_GetBlockProtectMask(uint8_t ***BlockNum***)
- void FC_MaskProtectBit(uint8_t BlockNum, FunctionalState ***NewState***)

7.2.2 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(),FC_GetBlockProtectMask(), FC_MaskProtectBit().

4) Use automatic operation command to write or erase the content of flash.

   FC_WritePage(), FC_EraseBlock(), FC_EraseChip().

7.2.3 Function Documentation

7.2.3.1 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 FCPSRA<BLK [3:0]> 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

7.2.3.2 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.

7.2.3.3 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.

7.2.3.4 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,

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

7.2.3.5 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,

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.

7.2.3.6 FC_EraseBlockProtectState

Erase the protection bits.

Prototype:

FC_Result

FC_EraseBlockProtectState(uint8_t ***BlockGroup***)

Parameters:

BlockGroup: The flash block group

- **FC_BLOCK_GROUP_0** for block 0 through 3.

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.

7.2.3.7 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 128Bytes.

Description:

This API is used to write data to specified page.

The TMPM06x contains 32 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.

7.2.3.8 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.

7.2.3.9 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.

7.2.3.10 FC_GetBlockProtectMask

Get the specified block protection bit mask state.

Prototype:

FunctionalState

FC_GetBlockProtectMask(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,

Description:

Get the mask protection bit represents the protection status of the corresponding block.

When a bit is set to "0", it indicates that the block corresponding to the bit is released. When a bit is set to "1", the corresponding bit is set to the protect state and this block becomes protect state.

Return:

State of mask the specified block protect bit.

DISABLE: Protection state is not available.

ENABLE: Protection state is available.

7.2.3.11 FC_MaskProtectBit

Mask the specified block protect bit.

Prototype:

Void

FC_MaskProtectBit(uint8_t BlockNum, FunctionalState NewState)

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,

NewState: Select the state of mask the specified block protect bit.

This parameter can be one of the following values:

- **DISABLE**: Protection state is not available.
- **ENABLE**: Protection state is available.

Description:

This API is used to set the mask protection bit to “1” so that the corresponding block can be protected.

When a bit is set to "0", it indicates that the block corresponding to the bit is released. When a bit is set to "1", the corresponding bit is set to the protect state and this block becomes protect state.

Return:

None

7.2.4 Data Structure Description

None

8. GPIO

8.1 Overview

For TOSHIBA TMPM06x 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/TX00_Periph_Driver/src/ tmpm06x _gpio.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x _gpio.h containing the macros, data types, structures and API definitions for use by applications.

8.2 Difference among TMPM066/067/068 in GPIO

- Port B/H/J is for TMPM066/M068 only.

8.3 API Functions

8.3.1 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***)
- void GPIO_SetInputVoltage(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***, uint8_t ***BitValue***)

8.3.2 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() and GPIO_SetInputVoltage() handle other specified functions.

8.3.3 Function Documentation

8.3.3.1 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

Description:

This function will read specified GPIO Data register.

Return:

The value read from DATA register.

8.3.3.2 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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.

8.3.3.3 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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

Description:

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

Return:

None

8.3.3.4 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- Combination of the effective bits

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

8.3.3.5 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- Combination of the effective bits.

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

Description:

This function will 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

8.3.3.6 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- Combination of the effective bits.

Description:

This function will set specified GPIO pin as output port.

Return:

None

8.3.3.7 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- Combination of the effective bits.

Description:

This function will set specified GPIO pin as input port.

Return:

None

8.3.3.8 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- 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

8.3.3.9 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- 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

8.3.3.10 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- 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 pull-up function when ***NewState*** is **DISABLE**.

Return:

None

8.3.3.11 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_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).
-

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:** GPIO pin[0:7],
- 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 pull-down function when ***NewState*** is **DISABLE**.

Return:

None

8.3.3.12 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 (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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:** GPIO pin[0:7],
- 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

8.3.3.13 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_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).

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,

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:** GPIO pin[0:7],
- Combination of the effective bits.

Description:

This function will enable GPIO pin specified function.

Return:

None

8.3.3.14 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_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B (only for TMPM066/M068).
- **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 (only for TMPM066/M068).
- **GPIO_PJ:** GPIO port J (only for TMPM066/M068).
-

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,

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:** GPIO pin[0:7],
- Combination of the effective bits.

Description:

This function will disable GPIO pin specified function.

Return:

None

8.3.3.15 GPIO_SetInputVoltage

Input voltage selection function of specified GPIO Pin.

Prototype:

void

GPIO_SetInputVoltage (GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x,***

uint8_t ***BitValue***)

Parameters:

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

- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PG:** GPIO port G.

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:** GPIO pin[0:7],
- Combination of the effective bits.

BitValue: The value of input Voltage range control register, which can be set as:

- **GPIO_BIT_VALUE_0** for 3V input,
- **GPIO_BIT_VALUE_1** for 1.8V input,

Description:

This function will control to input Voltage range of GPIO pin specified function.

Return:

None

8.3.4 Data Structure Description

8.3.4.1 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.

uint8_t

InputVoltage Input Voltage selection function, which can be set as:

- **GPIO_BIT_VALUE_0:** 3V input**.**
- **GPIO_BIT_VALUE_1:** 1.8V input**.**

8.3.4.2 GPIO_RegTypeDef

Data Fields:

uint8_t

PinDATA Port x data register, port data read and write by this variable.

uint8_t

PinCR Port x output control register:

- **"0":** output disable**.**
- **"1":** output enable**.**

uint8_t

PinFR[FRMAX] Function setting register. You will be able to use the functions assigned by setting "1"

uint8_t

PinOD Port x open drain control register:

- **"0":** CMOS
- **"1":** Open Drain

uint8_t

PinPUP Port x pull-up control register:

- **"0":** Pull-up disable**.**
- **"1":** Pull-up enable**.**

uint8_t

PinPDN Port x pull-down control register:

- **"0":** Pull-down disable**.**
- **"1":** Pull-down enable**.**

PinSEL Port x Input Voltage range control register:

- **"0":** 3V input**.**
- **"1":** 1.8V input**.**

uint8_t

PinPIE Port x input control register:

- **"0":** Input disable**.**
- **"1":** Input enable**.**

8.3.4.1 TSB_Port_TypeDef

Data Fields:

__IO uint32_t

DATA The "DATA" can be read and written.

__IO uint32_t

PinCR The "CR" can be read and written.

__IO uint32_t

PinFR[FRMAX] The "FR[FRMAX]" can be read and written.

uint32_t

RESERVED0 [RESER] Reserved

__IO uint32_t

PinOD The “OD” can be read and written.

__IO uint32_t

PinPUP The “PUP” can be read and written.

__IO uint32_t

PinPDN The “PDN” can be read and written.

__IO uint32_t

SEL The “SEL” can be read and written

__IO uint32_t

PinPIE Port x input control register.

9. I2C

9.1 Overview

The TMPM06x contains I2C Bus Interface with 2 channels (I2C0~1).
The I2C bus is connected to external devices via SCL and SDA, and it can communicate With multiple devices.
Data can be transferred in free data format by the I2C channels. In free data format, data is always sent by master-transmitter and received by slave-receiver.

The I2C driver APIs provide a set of functions to configure each channel such as setting self-address of the I2C 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 I2C channel.

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm06x_i2c.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_i2c.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 Function List

- void I2C_SetACK(TSB_I2C_TypeDef* ***I2Cx***, FunctionalState ***NewState***);
- void I2C_Init(TSB_I2C_TypeDef* ***I2Cx***, I2C_InitTypeDef* ***InitI2CStruct***);
- void I2C_SetBitNum(TSB_I2C_TypeDef* ***I2Cx***, uint32_t ***I2CBitNum***);
- void I2C_SWReset(TSB_I2C_TypeDef* ***I2Cx***);
- void I2C_ClearINTReq(TSB_I2C_TypeDef* ***I2Cx***);
- void I2C_GenerateStart(TSB_I2C_TypeDef* ***I2Cx***);
- void I2C_GenerateStop(TSB_I2C_TypeDef* ***I2Cx***);
- I2C_State I2C_GetState(TSB_I2C_TypeDef* ***I2Cx***);
- void I2C_SetSendData(TSB_I2C_TypeDef* ***I2Cx***, uint32_t ***Data***);
- uint32_t I2C_GetReceiveData(TSB_I2C_TypeDef* ***I2Cx***);
- void I2C_SetFreeDataMode(TSB_I2C_TypeDef* ***I2Cx,*** FunctionalState ***NewState***);
- FunctionalState I2C_GetSlaveAddrMatchState(TSB_I2C_TypeDef * ***I2Cx***);
- void I2C_SetPrescalerClock(TSB_I2C_TypeDef * ***I2Cx***, uint32_t ***PrescalerClock***);
- void I2C_SetINTReq(TSB_I2C_TypeDef * ***I2Cx***,FunctionalState ***NewState***);
- FunctionalState I2C_GetINTStatus(TSB_I2C_TypeDef * ***I2Cx***);
- void I2C_ClearINTOutput(TSB_I2C_TypeDef * ***I2Cx***);

9.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Configure and control the common functions of each I2C channel are handled by I2C_SetACK(), I2C_SetBitNum(),I2C_SetPrescalerClock()and I2C_Init().
2) Transfer control of each I2C channel is handled by I2C_ClearINTReq(), I2C_Generatestart(), I2C_Generatestop(),I2C_SetSendData(), I2C_GetReceiveData(),I2C_SetINTReq(),I2C_ClearINTOutput().
3) The status indication of each I2C channel is handled by I2C_GetState(),I2C_GetSlaveAddrMatchState() and I2C_GetINTStatus().
4) I2C_SWReset() and I2C_ SetFreeDataMode() handle other specified functions.

9.2.3 Function Documentation

Note**: in all of the following APIs, parameter "TSB_I2C_TypeDef* ***I2Cx" can be one of the following values:

TSB_I2C0, TSB_I2C1

9.2.3.1 I2C_SetACK

Enable or disable the generation of ACK clock.

Prototype:

void

I2C_SetACK(TSB_I2C_TypeDef* ***I2Cx***,

FunctionalState ***NewState***)

Parameters:

I2Cx is the specified I2C 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

9.2.3.2 I2C_Init

Initialize the specified I2C channel in I2C mode.

Prototype:

void

I2C_Init(TSB_I2C_TypeDef* ***I2Cx***,

I2C_InitTypeDef* ***InitI2CStruct***)

Parameters:

I2Cx is the specified I2C channel.

InitI2CStruct is the structure containing I2C configuration (refer to 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 I2C channel selected by ***I2Cx***.

Return:

None

9.2.3.3 I2C_SetBitNum

Specify the number of bits per transfer.

Prototype:

void

I2C_SetBitNum(TSB_I2C_TypeDef* ***I2Cx***,

uint32_t ***I2CBitNum***)

Parameters:

I2Cx is the specified I2C channel.

I2CBitNum specifies the number of bits per transfer, max. 8.

This parameter can be one of the following values:

- **I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;
- **I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;
- **I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;
- **I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;

- **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

9.2.3.4 I2C_SWReset

Reset the state of the specified I2C channel.

Prototype:

void

I2C_SWReset(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C 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

9.2.3.5 I2C_ClearINTReq

Clear I2C interrupt request in I2C bus mode.

Prototype:

void

I2C_ClearINTReq(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

This function will clear the I2C interrupt, which has occurred, of the specified I2C channel.

Return:

None

9.2.3.6 I2C_GenerateStart

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

Prototype:

void

I2C_GenerateStart(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

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

Return:

None

9.2.3.7 I2C_GenerateStop

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

Prototype:

void

I2C_GenerateStop(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

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

Return:

None

9.2.3.8 I2C_GetState

Get the I2C channel state in I2C bus mode.

Prototype:

I2C_State

I2C_GetState(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

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

Return:

The state value of the I2C channel in I2C bus.

9.2.3.9 I2C_SetSendData

Set data to be sent and start transmitting from the specified I2C channel.

Prototype:

void

I2C_SetSendData(TSB_I2C_TypeDef* ***I2Cx***,

uint32_t ***Data***)

Parameters:

I2Cx is the specified I2C 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 I2C channel selected by ***I2Cx***. It is appropriate to call the function after the transmission of the start condition, which can be done by **I2C_Generatestart()**, or the reception of an ACK (usually causes an I2C interrupt), to send further data required by receiver.

Return:

None

9.2.3.10 I2C_GetReceiveData

Get data received from the specified I2C channel.

Prototype:

uint32_t

I2C_GetReceiveData(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

This function will set the data to be sent from the specified I2C channel selected by ***I2Cx***. It is appropriate to call the function after the transmission of the start condition, which can be done by **I2C_Generatestart()**, or the reception of an ACK (usually causes an I2C interrupt), to send further data required by receiver.

Return:

Data which has been received

9.2.3.11 I2C_SetFreeDataMode

Set I2C channel working in I2C free data mode.

Prototype:

void

I2C_SetFreeDataMode(TSB_I2C_TypeDef* ***I2Cx***,

FunctionalState ***NewState***)

Parameters:

I2Cx is the specified I2C channel.

NewState specifies the state of the I2C when system is idle mode, which can be

- **ENABLE:** enables the I2C channel.
- **DISABLE:** disables the I2C channel.

Description:

The specified I2C 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 I2C is needed to shift to transfer data in normal I2C format, call **I2C_Init()**.

Return:

None

9.2.3.12 I2C_GetSlaveAddrMatchState

Get slave address match detection state.

Prototype:

FunctionalState

I2C_ GetSlaveAddrMatchState(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

Get slave address match detection state.

Return:

The state of match detection
ENABLE: Slave address is matched.
DISABLE: Slave address is unmatched.

9.2.3.13 I2C_SetPrescalerClock

Set prescaler clock of the specified I2C channel.

Prototype:

void

I2C_SetPrescalerClock(TSB_I2C_TypeDef* ***I2Cx***,

uint32_t ***PrescalerClock***)

Parameters:

I2Cx is the specified I2C channel.

PrescalerClock is the prescaler clock value.

This parameter can be one of the following values:

- **I2C_PRESCALER_DIV_1** to **I2C_PRESCALER_DIV_32**

Description:

This function will set prescaler clock of the specified I2C channel,

The system clock(fsys) is divided according to ***PrescalerClock*** as the prescaler clock(fprsck), and the prescaler clock is further divided by ***I2CClkDiv*** (refer to Data Structure Description for details).and used as the serial clock for I2C transfer,

Make sure the prescaler clock in the range between 50ns and 150ns.

Return:

None

9.2.3.14 I2C_SetINTReq

Enable or disable interrupt request of the I2C channel.

Prototype:

void

I2C _SetINTReq(TSB_I2C_TypeDef* ***I2Cx***,

FunctionalState ***NewState***)

Parameters:

I2Cx is the specified I2C channel.

NewState: Specify I2C interrupt setting

This parameter can be one of the following values:

- **ENABLE :** Enable I2C interrupt
- **DISABLE :** Disable I2C interrupt

Description:

This function will enable or disable I2C interrupt request.

Return:

None

9.2.3.15 I2C_GetINTStatus

Get interrupt generation state.

Prototype:

FunctionalState

I2C_GetINTStatus(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

This function will get the state of I2C interrupt generation.

Return:

The state of interrupt generation
ENABLE: I2C interrupt has been generated
DISABLE: I2C has not interrupt

9.2.3.16 I I2C_ClearINTOutput

Clear the I2C interrupt output.

Prototype:

void

I2C_ClearINTOutput(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:

I2Cx is the specified I2C channel.

Description:

This function will clear the I2C interrupt output, which has occurred, of the specified I2C channel.

Return:

None

9.2.4 Data Structure Description

9.2.4.1 I2C_InitTypeDef

Data Fields:

uint32_t

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

uint32_t

I2CDataLen Specify data length of the I2C channel in I2C mode, which can be set as:

- **I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;

- **I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;
- **I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;
- **I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;
- **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 prescaler clock for I2C transfer, which can be set as:

- **I2C_SCK_CLK_DIV_20**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 20;
- **I2C_SCK_CLK_DIV_24**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 24;
- **I2C_SCK_CLK_DIV_32**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 32;
- **I2C_SCK_CLK_DIV_48**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 48;
- **I2C_SCK_CLK_DIV_80**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 80;
- **I2C_SCK_CLK_DIV_144**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 144;
- **I2C_SCK_CLK_DIV_272**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 272;
- **I2C_SCK_CLK_DIV_528**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 528;

uint32_t

PrescalerClkDiv specifies the division of the system clock for generating the fprsck, which can be set as:

- **I2C_PRESCALER_DIV_1**, which means that the frequency of the prescaler clock is quotient of fsys divided by 1
- **I2C_PRESCALER_DIV_2**, which means that the frequency of the prescaler clock is quotient of fsys divided by 2
- **I2C_PRESCALER_DIV_3**, which means that the frequency of the prescaler clock is quotient of fsys divided by 3
- **I2C_PRESCALER_DIV_4**, which means that the frequency of the prescaler clock is quotient of fsys divided by 4
- **I2C_PRESCALER_DIV_5**, which means that the frequency of the prescaler clock is quotient of fsys divided by 5
- **I2C_PRESCALER_DIV_6**, which means that the frequency of the prescaler clock is quotient of fsys divided by 6
- **I2C_PRESCALER_DIV_7**, which means that the frequency of the prescaler clock is quotient of fsys divided by 7
- **I2C_PRESCALER_DIV_8**, which means that the frequency of the prescaler clock is quotient of fsys divided by 8

- **I2C_PRESCALER_DIV_9**, which means that the frequency of the prescaler clock is quotient of fsys divided by 9
- **I2C_PRESCALER_DIV_10**, which means that the frequency of the prescaler clock is quotient of fsys divided by 10
- **I2C_PRESCALER_DIV_11**, which means that the frequency of the prescaler clock is quotient of fsys divided by 11
- **I2C_PRESCALER_DIV_12**, which means that the frequency of the prescaler clock is quotient of fsys divided by 12
- **I2C_PRESCALER_DIV_13**, which means that the frequency of the prescaler clock is quotient of fsys divided by 13
- **I2C_PRESCALER_DIV_14**, which means that the frequency of the prescaler clock is quotient of fsys divided by 14
- **I2C_PRESCALER_DIV_15**, which means that the frequency of the prescaler clock is quotient of fsys divided by 15
- **I2C_PRESCALER_DIV_16**, which means that the frequency of the prescaler clock is quotient of fsys divided by 16
- **I2C_PRESCALER_DIV_17**, which means that the frequency of the prescaler clock is quotient of fsys divided by 17
- **I2C_PRESCALER_DIV_18**, which means that the frequency of the prescaler clock is quotient of fsys divided by 18
- **I2C_PRESCALER_DIV_19**, which means that the frequency of the prescaler clock is quotient of fsys divided by 19
- **I2C_PRESCALER_DIV_20**, which means that the frequency of the prescaler clock is quotient of fsys divided by 20
- **I2C_PRESCALER_DIV_21**, which means that the frequency of the prescaler clock is quotient of fsys divided by 21
- **I2C_PRESCALER_DIV_22**, which means that the frequency of the prescaler clock is quotient of fsys divided by 22
- **I2C_PRESCALER_DIV_23**, which means that the frequency of the prescaler clock is quotient of fsys divided by 23
- **I2C_PRESCALER_DIV_24**, which means that the frequency of the prescaler clock is quotient of fsys divided by 24
- **I2C_PRESCALER_DIV_25**, which means that the frequency of the prescaler clock is quotient of fsys divided by 25
- **I2C_PRESCALER_DIV_26**, which means that the frequency of the prescaler clock is quotient of fsys divided by 26
- **I2C_PRESCALER_DIV_27**, which means that the frequency of the prescaler clock is quotient of fsys divided by 27
- **I2C_PRESCALER_DIV_28**, which means that the frequency of the prescaler clock is quotient of fsys divided by 28
- **I2C_PRESCALER_DIV_29**, which means that the frequency of the prescaler clock is quotient of fsys divided by 29
- **I2C_PRESCALER_DIV_30**, which means that the frequency of the prescaler clock is quotient of fsys divided by 30
- **I2C_PRESCALER_DIV_31**, which means that the frequency of the prescaler clock is quotient of fsys divided by 31
- **I2C_PRESCALER_DIV_32**, which means that the frequency of the prescaler clock is quotient of fsys divided by 32

***Note**: Make sure the prescaler clock in the range between 50ns and 150ns.

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.

9.2.4.2 I2C_State

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.

10. I2CS

10.1 Overview

When the I2C block is used in salve mode, if the frame address on the I2C bus matches a self-address (slave address) in low-power consumption mode(STOP1), an interrupt(INTI2CS) occurs to clear low-power consumption mode.

The I2CS driver APIs provide a set of functions to configure I2CS channel such as setting self-address of the I2CS channel, 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 channel such as returning the state or the mode of I2CS channel.

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm06x_i2cs.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_i2cs.h containing the macros, data types, structures and API definitions for use by applications.

10.2 API Functions

10.2.1 Function List

- void I2CS_SetACK(FunctionalState ***NewState)***;
- void I2CS_Reset(FunctionalState ***NewState***);
- void I2CS_INTRelease(FunctionalState ***NewState***);
- I2CS_State I2CS_GetState(void);
- void I2CS_Set2ndSlaveAddState(FunctionalState ***NewState***);
- I2CS_MatchAdd I2CS_GetMatchAddState(void);
- void I2CS_Set1stSlaveAdd(uint8_t ***Data***);
- uint8_t I2CS_Get1stSlaveAdd(void);
- void I2CS_Set2ndSlaveAdd(uint8_t ***Data***);
- uint8_t I2CS_Get2ndSlaveAdd(void);

10.2.2 Detailed Description

Functions listed above can be divided into four parts:

5) Configure and control the common functions of I2CS channel are handled by I2CS_SetACK (), I2CS_Set1stSlaveAdd (), I2CS_Get1stSlaveAdd () and I2CS_Set2ndSlaveAdd (), I2CS_Get2ndSlaveAdd ().
6) Transfer control of I2CS channel is handled by I2CS_INTRelease
7) The status indication of I2CS channel is handled by I2CS_GetState (), I2CS_Set2ndSlaveAddState () and I2CS_GetMatchAddState ().
8) I2CS_Reset () handle other specified functions.

10.2.3 Function Documentation

10.2.3.1 I2CS_SetACK

Enable or disable the generation of ACK clock.

Prototype:

void

I2CS_SetACK(FunctionalState ***NewState***)

Parameters:

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 I2CS 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

10.2.3.2 I2CS_Reset

Reset the state of the specified I2CS channel.

Prototype:

void

I2CS_Reset (FunctionalState ***NewState***)

Parameters:

NewState reset the state, which can be:

- **ENABLE** reset done
- **DISABLE** reset release

Description:

The function reset the state of the specified I2CS channel. The reset will be done if ***NewState*** is **ENABLE**. And the reset will be released if ***NewState*** is **DISABLE**.

Return:

None

10.2.3.3 I2CS_INTRelease

Release I2CS interrupt.

Prototype:

void

I2CS_INTRelease (FunctionalState ***NewState***)

Parameters:

NewState reset the state, which can be:

- **ENABLE** release interrupt
- **DISABLE** nothing

Description:

The function release I2CS interrupt. The interrupt will be released if ***NewState*** is **ENABLE**.

Return:

None

10.2.3.4 I2CS_GetState

Get the I2CS state in I2CS mode.

Prototype:

I2CS_State

I2CS_GetState (void)

Parameters:

None

Description:

The function to get the I2CS state in I2CS mode.

Return:

The state of the I2CS:
INTRelease: Release the I2CS interrupt
GeneralCallDet: State of General call detection
TRx: State of slave transmit or receive
I2RES: State of I2CBUS reset
ARK : ACK output status

10.2.3.5 I2CS_Set2ndSlaveAddState

Enable or disable using 2nd slave address of the I2CS channel.

Prototype:

void

I2CS_Set2ndSlaveAddState (FunctionalState ***NewState***)

Parameters:

NewState enable or disable using 2nd slave address, which can be:

- **ENABLE** using 2nd slave address
- **DISABLE** not using 2nd slave address

Description:

The function enable or disable using 2nd slave address. The 2nd slave address will be used if ***NewState*** is **ENABLE**. And the 2nd slave address will not be used if ***NewState*** is **DISABLE**.

Return:

None

10.2.3.6 I2CS_GetMatchAddState

Get the I2CS match slave address state.

Prototype:

I2CS_MatchAdd

I2CS_GetMatchAddState (void)

Parameters:

None

Description:

The function to get the I2CS match slave address state.

Return:

The state of the I2CS match slave address in I2CS bus:
Match1stSlaveAdd: Match or not 1st slave address
Match2ndSlaveAdd: Match or not 2nd slave address

10.2.3.7 I2CS_Set1stSlaveAdd

Set 1st slave address.

Prototype:

void

I2CS_Set1stSlaveAdd (uint8_t ***Data***)

Parameters:

Data: 1st slave address, max 0x7F

Description:

The function to set 1st slave address.

Return:

None

10.2.3.8 I2CS_Get1stSlaveAdd

Get 1st slave address.

Prototype:

uint8_t

I2CS_Get1stSlaveAdd (void)

Parameters:

None

Description:

The function to get 1st slave address.

Return:

The 1st slave address.

10.2.3.9 I2CS_Set2ndSlaveAdd

Set 2nd slave address.

Prototype:

void

I2CS_Set2ndSlaveAdd (uint8_t ***Data***)

Parameters:

Data: 2nd slave address, max 0x7F

Description:

The function to set 2nd slave address.

Return:

None

10.2.3.10 I2CS_Get t2nd SlaveAdd

Get t 2nd slave address.

Prototype:

uint8_t

I2CS_Get2ndSlaveAdd (void)

Parameters:

None

Description:

The function to get 2nd slave address.

Return:

The 2nd slave address.

10.2.4 Data Structure Description

10.2.4.1 I2CS_State

Data Fields:

Uint8_t

All specifies I2CS state data***.***

. **Bit Fields**:

Uint8_t

INTRelease (Bit0) means release the I2CS interrupt.

Uint8_t

GeneralCallDet (Bit 1) means state of General call detection.

Uint8_t

Reserved (Bit 2) Reserved

.

Uint8_t

TRx (Bit3) means state of slave transmit or receive

.

Uint8_t

I2RES (Bit4) means state of I2CBUS reset.

.

Uint8_t

ARK (Bit5) means ACK output status

.

Uint8_t

GeneralCall (Bit6) means general call detection on or off

.

Uint8_t

StopStart (Bit7) means stop or Start condition detection

10.2.4.2 I2CS_MatchAdd

Data Fields:

Uint8_t

All specifies match slave address state***.***

. **Bit Fields**:

Uint8_t

Reserved1 (Bit0) Reserved.

Uint8_t

Match1stSlaveAdd (Bit 1) means match or not 1st slave address.

Uint8_t

Match2ndSlaveAdd (Bit 2) means match or not 2nd slave address

.

Uint8_t

Reserved2 (Bit3~Bit7) Reserved

.

11. LVD

11.1 Overview

TMPM06x has Low voltage detection circuit (LVD). The voltage detection circuit generates a reset signal or an interrupt signal by detecting a decreasing/increasing voltage.

The LVD driver APIs provide a set of functions to enable or disable the LVD function, configure detection voltage and get the detection voltage interrupt status.

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm06x_lvd.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_lvd.h containing the macros, data types, structures and API definitions for use by applications.

11.2 API Functions

11.2.1 Function List

- void LVD_EnableVD1(void)
- void LVD_DisableVD1(void)
- void LVD_SetVD1Level(uint32_t ***VDLevel***)
- LVD_VDStatus LVD_GetVD1Status(void)
- void LVD_EnableVD2(void)
- void LVD_DisableVD2(void)
- void LVD_SetVD2Level(uint32_t ***VDLevel***)
- LVD_VDStatus LVD_GetVD2Status(void)
- void LVD_SetVD1ResetOutput(FunctionalState ***NewState***)
- void LVD_SetVD1INTOutput(FunctionalState ***NewState***)
- uint32_t LVD_GetVD1INTCondition(void);
- void LVD_SetVD2ResetOutput(FunctionalState ***NewState***)
- void LVD_SetVD2INTOutput(FunctionalState ***NewState***)

- uint32_t LVD_GetVD2INTCondition(void);

11.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) Configure LVD are handled by LVD_EnableVD1(), LVD_DisableVD1(), LVD_SetVD1Level(); LVD_EnableVD2(), LVD_DisableVD2(), LVD_SetVD2Level(); LVD_SetVD1ResetOutput(), LVD_SetVD1INTOutput(),LVD_SetVD2ResetOutput(), LVD_SetVD2INTOutput().
2) Get the power supply voltage detection status and interrupt generation condition info by LVD_GetVD1Status(),LVD_GetVD1INTCondition (),LVD_GetVD2INTCondition ().

11.2.3 Function Documentation

11.2.3.1 LVD_EnableVD1

Enable the operation of voltage detection 1.

Prototype:

void

LVD_EnableVD1(void)

Parameters:

None.

Description:

This function will enable the voltage detection 1 operation.

Return:

None.

11.2.3.2 LVD_DisableVD1

Disable the operation of voltage detection 1.

Prototype:

void

LVD_DisableVD1(void)

Parameters:

None.

Description:

This function will disable the voltage detection 1 operation.

Return:

None.

11.2.3.3 LVD_SetVD1Level

Select the level for detection voltage 1.

Prototype:

void

LVD_SetVD1Level(uint32_t ***VDLevel***)

Parameters:

VDLevel is the level of voltage detection 1.

This parameter can be one of the following values:

- **LVD_VDLVL1_200:** Voltage detection level is from 2.0 ± 0.2V.
- **LVD_VDLVL1_210:** Voltage detection level is from 2.1 ± 0.2V.

Description:

This function will set the level of voltage detection 1.

Return:

None.

11.2.3.4 LVD_GetVD1Status

Get voltage detection 1 status.

Prototype:

LVD_VDStatus

LVD_GetVD1Status(void)

Parameters:

None.

Description:

This function will get voltage detection 1 status.

Return:

LVD_VDStatus: The voltage detection 1 status, which can be one of:

LVD_VD_UPPER: Power-supply voltage is the same as detection voltage or higher.

LVD_VD_LOWER: Power-supply voltage is the same as detection voltage or lower.

11.2.3.5 LVD_EnableVD2

Enable the operation of voltage detection 2.

Prototype:

void

LVD_EnableVD2(void)

Parameters:

None.

Description:

This function will enable the voltage detection 2 operation.

Return:

None.

11.2.3.6 LVD_DisableVD2

Disable the operation of voltage detection 2.

Prototype:

void

LVD_DisableVD2(void)

Parameters:

None.

Description:

This function will disable the voltage detection 2 operation.

Return:

None.

11.2.3.7 LVD_SetVD2Level

Select the detection voltage 2 level.

Prototype:

void

LVD_SetVD2Level(uint32_t ***VDLevel***)

Parameters:

VDLevel is the voltage detection 2 level.

This parameter can be one of the following values:

- **LVD_VDLVL2_220:** Voltage detection level is from 2.20 ± 0.2V.
- **LVD_VDLVL2_230:** Voltage detection level is from 2.30 ± 0.2V.
- **LVD_VDLVL2_240:** Voltage detection level is from 2.40 ± 0.2V.
- **LVD_VDLVL2_250:** Voltage detection level is from 2.50 ± 0.2V.
- **LVD_VDLVL2_260:** Voltage detection level is from 2.60 ± 0.2V.
- **LVD_VDLVL2_270:** Voltage detection level is from 2.70 ± 0.2V.
- **LVD_VDLVL2_280:** Voltage detection level is from 2.80 ± 0.2V.
- **LVD_VDLVL2_290:** Voltage detection level is from 2.90 ± 0.2V.

Description:

This function will set the level of voltage detection 2.

Return:

None.

11.2.3.8 LVD_GetVD2Status

Get voltage detection 2 status.

Prototype:

LVD_VDStatus

LVD_GetVD2Status(void)

Parameters:

None.

Description:

This function will get voltage detection 2 status.

Return:

LVD_VDStatus: The voltage detection 2 status, which can be one of:

LVD_VD_UPPER: Power supply voltage is upper than the detection voltage.

LVD_VD_LOWER: Power supply voltage is lower than the detection voltage.

11.2.3.9 LVD_SetVD1ResetOutput

Enable or disable LVD reset output of voltage detection 1.

Prototype:

void

LVD_SetVD1ResetOutput(FunctionalState ***NewState***)

Parameters:

NewState: new state of LVD reset output.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function enables or disables LVD reset output of voltage detection 1.

Return:

None.

11.2.3.10 LVD_SetVD1INTOutput

Enable or disable LVD interrupt output of voltage detection 1.

Prototype:

void

LVD_SetVD1INTOutput(FunctionalState ***NewState***)

Parameters:

NewState: new state of LVD interrupt output.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function enables or disables LVD interrupt output of voltage detection 1.

Return:

None.

11.2.3.11 LVD_GetVD1INTCondition

Get voltage detection 1 interrupt generation condition.

Prototype:

uint32_t

LVD_GetVD1INTCondition (void)

Parameters:

None.

Description:

This function will get voltage detection 1 interrupt generation condition.

Return:

The voltage detection interrupt generation condition, which can be

LVD_INTSEL_LOWER: Only lower than the setting voltage when voltage decreasing.

LVD_INTSEL_LOWER_UPPER: Both lower and upper than the setting voltage when voltage decreasing.

11.2.3.12 LVD_SetVD2ResetOutput

Enable or disable LVD reset output of voltage detection 2.

Prototype:

void

LVD_SetVD2ResetOutput(FunctionalState ***NewState***)

Parameters:

NewState: new state of LVD reset output.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function enables or disables LVD reset output of voltage detection 2.

Return:

None.

11.2.3.13 LVD_SetVD2INTOutput

Enable or disable LVD interrupt output of voltage detection 2.

Prototype:

void

LVD_SetVD2INTOutput(FunctionalState ***NewState***)

Parameters:

NewState: new state of LVD interrupt output.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function enables or disables LVD interrupt output of voltage detection 2.

Return:

None.

11.2.3.14 LVD_GetVD2INTCondition

Get voltage detection 2 interrupt generation condition.

Prototype:

uint32_t

LVD_GetVD2INTCondition (void)

Parameters:

None.

Description:

This function will get voltage detection 2 interrupt generation condition.

Return:

The voltage detection interrupt generation condition, which can be

LVD_INTSEL_LOWER: Only lower than the setting voltage when voltage decreasing.

LVD_INTSEL_LOWER_UPPER: Both lower and upper than the setting voltage when voltage decreasing.

11.2.4 Data Structure Description

None

12. TMR16A

12.1 Overview

TOSHIBA TMPM06x has 2 channels TMR16A that contains the following functions:

- Match interrupt
- Square waveform output
- Read capture.

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm06x_tmr16a.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_tmr16a.h containing the macros, data types, structures and API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- void TMR16A_SetClkInCoreHalt(TSB_T16A_TypeDef * ***T16Ax***, uint8_t ***ClkState***);
- void TMR16A_SetRunState(TSB_T16A_TypeDef * ***T16Ax***, uint32_t ***Cmd***);
- void TMR16A_SetSrcClk(TSB_T16A_TypeDef * ***T16Ax***, uint32_t ***SrcClk***);
- void TMR16A_SetFlipFlop(TSB_T16A_TypeDef * ***T16Ax***, TMR16A_FFOutputTypeDef * ***FFStruct***);
- void TMR16A_ChangeCycle(TSB_T16A_TypeDef * ***T16Ax***, uint32_t ***Cycle***);
- uint16_t TMR16A_GetCaptureValue(TSB_T16A_TypeDef* ***T16Ax***);

12.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) Configure and control the common functions of each TMR16A channel are handled by TMR16A_SetSrcClk(),TMR16A_SetRunState()and TMR16A_ChangeCycle().
2) The status indication of each TMR16A channel is handled by TMR16A_GetCaptureValue().

3) TMR16A_SetFlipFlop(),TMR16A_SetClkInCoreHalt () handle other specified functions.

12.2.3 Function Documentation

***Note**: in all of the following APIs, unless otherwise specified, the parameter:

"TSB_T16A_TypeDef * ***T16Ax***" can be one of the following values:

TSB_T16A0, TSB_T16A1.

12.2.3.1 TMR16A_SetClkInCoreHalt

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

Prototype:

void

TMR16A_SetClkInCoreHalt (TSB_T16A_TypeDef* ***T16Ax***,

uint8_t ***ClkState***)

Parameters:

T16Ax is the specified TMR16A channel.

ClkState specifies timer state in HALT mode, which can be

- **TMR16A_RUNNING_IN_CORE_HALT:** clock not stops in Core HALT
- **TMR16A_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

12.2.3.2 TMR16A_SetRunState

Start or stop counter of the specified T16A channel.

Prototype:

void

TMR16A_SetRunState(TSB_T16A_TypeDef* ***T16Ax***,

uint32_t ***Cmd***)

Parameters:

T16Ax is the specified TMR16A channel.

Cmd sets the state of up-counter, which can be:

- **TMR16A_RUN:** starting counting
- **TMR16A_STOP:** stopping counting

Description:

The up-counter of the specified TMR16A channel starts counting if ***Cmd*** is **TMR16A_RUN** and up-counter stops counting and the value in up-counter register is clear if ***Cmd*** is **TMR16A_STOP**.

Return:

None

12.2.3.3 TMR16A_SetSrcClk

Specifies a source clock.

Prototype:

void

TMR16A_SetSrcClk(TSB_T16A_TypeDef* ***T16Ax***,

uint32_t ***SrcClk***)

Parameters:

T16Ax is the specified TMR16A channel.

SrcClk specifies the state of the TMR16A source clock, which can be

- **TMR16A_SYSCK:** Select Source clock to SYSCK,
- **TMR16A_PRCK:** Select source clock to PRCK.

Description:

This function can select TMR16A channel's source clock.

Return:

None

12.2.3.4 TMR16A_SetFlipFlop

Configure the flip-flop function of the specified TMR16A channel.

Prototype:

void

TMR16A_SetFlipFlop(TSB_T16A_TypeDef* ***T16Ax***,

TMR16A_FFOutputTypeDef* ***FFStruct***)

Parameters:

T16Ax is the specified TMR16A channel.

FFStruct is the structure containing TMR16A 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 TMR16A channel. Also the level of the output can be controlled by this API.

Return:

None

12.2.3.5 TMR16A_ChangeCycle

Change the value of cycle for the specified channel.

Prototype:

void

TMR16A_ChangeCycle(TSB_T16A_TypeDef* ***T16Ax***,

uint32_t ***Cycle***)

Parameters:

T16Ax is the specified TMR16A channel.

Cycle specifies the value of cycle, max is 0xFFFF.

Description:

This function will specify the absolute value of cycle for the specified TMR16A. The actual interval of cycle depends on the configuration of CG and the value of ***ClkDiv***

Return:

None

12.2.3.6 TMR16A_GetCaptureValue

Get the value of capture register of the specified TMR16A channel.

Prototype:

uint16_t

TMR16A_GetCaptureValue(TSB_T16A_TypeDef* ***T16Ax***)

Parameters:

T16Ax is the specified TMR16A channel.

Description:

This function will return the value of capture register of the specified TMR16A channel.

Return:

The captured value.

12.2.1 Data Structure Description

12.2.1.1 TMR16A_FFOutputTypeDef

Data Fields:

uint32_t

TMR16AFlipflopCtrl selects the level of flip-flop output which can be

- **TMR16A_FLIPFLOP_INVERT:** setting output reversed by using software.
- **TMR16A_FLIPFLOP_SET:** setting output to be high level.
- **TMR16A_FLIPFLOP_CLEAR:** setting output to be low level.

uint32_t

TMR16AFlipflopReverseTrg specifies the reverse trigger of the flip-flop output, which can be set as:

- **TMR16A_DISALBE_FLIPFLOP**, which disables the flip-flop output reverse trigger.
- **TMR16A_FLIPFLOP_MATCH_CYCLE**, which means that the reversing flip-flop output will be triggered when the up-counter matches the cycle.

13. TMRD

13.1 Overview

TMPM06x has TMRD module, which consists of one TMRD block. TMRD block consists of 2 clock setting circuits (prescaler) supplying clocks to the timer unit, 2 timer units (TMR0 and TMR1) and H-SW control circuit. The functions are as follows.

- 16-bit interval timer
- 16-bit programmable pulse generation (PPG)
- Synchronous cycle signal output function
- H-SW control circuit

16-bit interval timer has the following two modes:

- Timer mode that TMR0 and TMR1 operate independently
- Interlock timer mode that can start TMRD0 and TMRD1 at the same time

16-bit programmable pulse generation has the following two modes:

- PPG mode that TMR0 and TMR1 independently output preprogrammed pulses
- PPG mode that can change the phase relation between the pulse output by TMR0 and that output by TMR1 in the range from -180 degree to +180 degree

Synchronous cycle signal output function:

- This timer can output signals that is synchronous with a timing of rectangular wave output (PPG).

H-SW control circuit:

- H-SW control circuit incorporates the circuit that generates output patterns synchronizing with rectangular wave outputs (PPG) generated by timer units with cycle synchronous signals at the timing of rectangular wave output (PPG) cycle.

The TMRD driver APIs provide a set of functions to configure TMRD module, such as setting the clock operation, timing parameters, PPG output phase, wave edge adjust, DMA request setting and set the compare 0 interrupt source, up counter's clearing way and so on.

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm06x_tmrd.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_tmrd.h containing the macros, data types, structures and API definitions for use by applications.

13.2 API Functions

13.2.1 Function List

- void TMRD_Enable(TMRD_UNIT_Channel ***CHx***)
- void TMRD_Disable(TMRD_UNIT_Channel ***CHx***)
- void TMRD_SetRunStateInHalt(uint8_t ***RunState***)
- void TMRD_SetRunStateInIdle(TMRD_UNIT_Channel ***CHx***,
  uint8_t ***RunState***)
- void TMRD_SetMode(uint8_t ***Mode***)
- void TMRD_SetClkDivision(TMRD_UNIT_Channel ***CHx***,
  uint8_t ***ClkDiv***)
- void TMRD_SetUpCntCtrl(TMRD_UNIT_Channel ***CHx***,
  uint8_t ***UpCntCtrl***)

- void TMRD_SetPPGInitLeadingEdge(uint8_t ***PPGChannel***,
  uint8_t ***WaveEdge***)
- void TMRD_SetCMPRegWritePath(
  TMRD_UNIT_Channel ***CHx***,
  uint8_t ***WritePath***)
- void TMRD_SetCMP0INTSrc(TMRD_UNIT_Channel ***CHx***,
  uint8_t ***INTSrc***)
- void TMRD_SetRunState(TMRD_UNIT_Channel ***CHx***,
  uint8_t ***RunState***)
- void TMRD_SetPhaseRelation(uint8_t ***PhaseRelation***)
- void TMRD_EnableUpdateCMPReg(
  TMRD_UNIT_Channel ***CHx***)
- void TMRD_SetDMAReq(TMRD_UNIT_Channel ***CHx***,
  FunctionalState ***NewState***)
- void TMRD_SetInitTiming(TMRD_UNIT_Channel ***CHx***,
  TMRD_TimingTypeDef * ***TimingStruct***)
- void TMRD_ChangeTiming(uint8_t ***TimingType***,
  uint16_t ***Timing***)
- uint16_t TMRD_GetTiming(uint8_t ***TimingType***)
- void TMRD_SetBitModulationCycle(
  TMRD_UNIT_Channel ***CHx***,
  uint8_t ***BitModCycle***)
- void TMRD_SetBitModUpdateTiming(
  TMRD_UNIT_Channel ***CHx***,
  FunctionalState ***NewState***)
- void TMRD_SetLOutHSWMode(
  TMRD_UNIT_Channel ***CHx***,
  TMRD_UNIT_Phase ***PHx***,
  Uint8_t ***LMode***)
- uint8_t TMRD_GetLOutHSWMode(
  TMRD_UNIT_Channel ***CHx***,
  TMRD_UNIT_Phase ***PHx***)
- void TMRD_EnableHSWMode(
  TMRD_UNIT_Channel ***CHx***,
  TMRD_UNIT_Phase ***PHx***)
- void TMRD_DisableHSWMode(
  TMRD_UNIT_Channel ***CHx***,

TMRD_UNIT_Phase ***PHx***)

- FunctionalState TMRD_GetHSWModeState(
  TMRD_UNIT_Channel ***CHx***,
  TMRD_UNIT_Phase ***PHx***)
- void TMRD_SetTDDIRLevel(
  TMRD_UNIT_Channel ***CHx***,
  TMRD_UNIT_Phase ***PH***
  uint8_t ***Level***)
- uint8_t TMRD_GetTDDIRLevel(
  TMRD_UNIT_Channel ***CHx***,
  TMRD_UNIT_Phase ***PHx***)

13.2.2 Detailed Description

Functions listed above can be divided into six parts:

1) Configure and control the common functions of each TMRD channel are handled by TMRD_Enable(), TMRD_Disable(), TMRD_SetUpCntCtrl (), TMRD_SetMode() and TMRD_SetRunState().
2) Clock source and division setting are handled by TMRD_SetRunStateInHalt(), TMRD_SetRunStateInIdle(), TMRD_SetClkDivision().
3) PPG output and phase control are handled by TMRD_SetPPGInitLeadingEdge(), TMRD_SetPhaseRelation(), TMRD_SetBitModulationCycle(), TMRD_SetBitModUpdateTiming().
4) Compare register and timer register control are handled by TMRD_SetCMPRegWritePath(), TMRD_EnableUpdateCMPReg(), TMRD_SetInitTiming(), TMRD_ChangeTiming() and TMRD_GetTiming().
5) Interrupt and DMA feature are handled by TMRD_SetCMP0INTSrc() and TMRD_SetDMAReq().
6) H-SW mode setting are handled by TMRD_SetLOutHSWMode(), TMRD_GetLOutHSWMode (), TMRD_EnableHSWMode(), TMRD_DisableHSWMode(), TMRD_GetHSWModeState(), TMRD_SetTDDIRLevel() and TMRD_GetTDDIRLevel().

13.2.3 Function Documentation

Note: in all of the following APIs, unless otherwise specified, the parameter:
"TMRD_UNIT_Channel ***CHx***" can be one of the following values:

TMRD_UNIT_CH_0 or TMRD_UNIT_CH_1.

Note: in all of the following APIs, unless otherwise specified, the parameter:
"TMRD_UNIT_Phase ***PHx***" can be one of the following values:

TMRD_UNIT_PH_0 or TMRD_UNIT_PH_1.

13.2.3.1 TMRD_Enable

Enable clock signal input to TMRD channel.

Prototype:

void

TMRD_Enable(TMRD_UNIT_Channel ***CHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

Description:

This function will enable the clock signal input to TMRD channel selected by ***TDx*** and ***CHx***.

Return:

None

13.2.3.2 TMRD_Disable

Disable the clock signal input to TMRD channel.

Prototype:

void

TMRD_Disable(TMRD_UNIT_Channel ***CHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

Description:

This function will disable the clock signal input to TMRD channel selected by ***TDx*** and ***CHx***.

Return:

None

13.2.3.3 TMRD_SetRunStateInHalt

Set TMRD operation if a HALT instruction is executed during debugging.

Prototype:

void

TMRD_SetRunStateInHalt(uint8_t ***RunState***)

Parameters:

TDx is the ***specified*** TMRD unit.

RunState sets the TMRD operation status, which can be:

- **TMRD_RUN:** Operation if a HALT instruction is executed during debugging.
- **TMRD_STOP:** Stop if a HALT instruction is executed during debugging.

Description:

This function will set the operation if a HALT instruction is executed during debugging.

Return:

None

13.2.3.4 TMRD_SetRunStateInIdle

Set TMRD operation during the IDLE mode.

Prototype:

void

TMRD_SetRunStateInIdle(TMRD_UNIT_Channel ***CHx***,

uint8_t ***RunState***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

RunState sets the TMRD operation status, which can be:

- **TMRD_RUN:** Operation during the IDLE mode.
- **TMRD_STOP:** Stop during the IDLE mode.

Description:

This function will set the TMRD operation during the IDLE mode.

Return:

None

13.2.3.5 TMRD_SetMode

Set the operation mode for TMRD.

Prototype:

void

TMRD_SetMode(uint8_t ***Mode***)

Parameters:

TDx is the specified TMRD unit.

Mode specifies TMRD operation mode, which can be

- **TMRD_MODE_BOTH_TMR**: TMR0: Timer mode, TMR1: Timer mode.
- **TMRD_MODE_0TMR_1PPG**: TMR0: Timer mode, TMR1: PPG mode.
- **TMRD_MODE_0PPG_1TMR**: TMR0: PPG mode, TMR1: Timer mode.
- **TMRD_MODE_BOTH_PPG**: TMR0: PPG mode, TMR1: PPG mode.
- **TMRD_MODE_INTERLOCK_TMR**: Timer mode in which TMR0 and TMR1 start simultaneously.
- **TMRD_MODE_INTERLOCK_PPG_2CH**: Interlock PPG mode in which TMR0 and TMR1 to operate together. (ch00 of TMR0 and ch10 of TMR1 are updated simultaneously)
- **TMRD_MODE_INTERLOCK_PPG_3CH**: Interlock PPG mode in which TMR0 and TMR1 to operate together. (ch00 of TMR0 and all channels of TMR1 are updated simultaneously)

Description:

This function will set the operation mode for TMR0 and TMR1. If operation mode is set to interlock PPG mode, the phase relationships between pulse generated by TMR1 and TMR0 can be changed.

Return:

None

Note:

If set the operation mode to **TMRD_MODE_INTERLOCK_PPG_2CH** or **TMRD_MODE_INTERLOCK_PPG_3CH**, TMRDCLK0 and TMRDCLK1 clock cannot be configured respectively, and TMRDCLK1 and TMRDCLK0 must have the same frequency.

13.2.3.6 TMRD_SetClkDivision

Set the clock prescaler of TMRD.

Prototype:

void

TMRD_SetClkDivision(TMRD_UNIT_Channel ***CHx,***

uint8_t ***ClkDiv***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

ClkDiv is the prescaler factor, which can be

- **TMRD_CLK_DIV_1**: TMRDCLK = ftmrd.
- **TMRD_CLK_DIV_2**: TMRDCLK = ftmrd/2.
- **TMRD_CLK_DIV_4**: TMRDCLK = ftmrd/4.
- **TMRD_CLK_DIV_8**: TMRDCLK = ftmrd/8.
- **TMRD_CLK_DIV_16**: TMRDCLK = ftmrd/16.

Description:

This function will select a clock prescaler of TMR0 and TMR1.

Return:

None

Note:

In the interlock PPG mode, setting the clock prescaler of TMR1 becomes invalid, and the prescaler of TMR1 will keep the same value with TMR0. So setting the prescaler of TMR0 by using this function is enough.

13.2.3.7 TMRD_SetUpCntCtrl

Select the timer up-counter operation when math the cycle.

Prototype:

Void

TMRD_SetUpCntCtrl(TMRD_UNIT_Channel ***CHx,***

uint8_t ***UpCntCtrl***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

UpCntCtrl is the operation mode of counter zero clearing, which can be

- **TMRD_FREE_RUN**: Operate as a free-run counter even if a match is detected.
- **TMRD_AUTO_CLEAR**: Zero cleared if a match is detected.

Description:

This function will select the up-counter operation when the match signal of CP00/CP10 is generated. If choose **TMRD_FREE_RUN**, the counter will be zero cleared when arrived at the maximum value 0xFFFF, and if choose **TMRD_AUTO_CLEAR**, the up-counter will be zero cleared when match the CP00/CP01.

Return:

None

Note:

In the PPG and interlock PPG mode, setting ***UpCntCtrl*** to **TMRD_FREE_RUN** becomes invalid.

13.2.3.8 TMRD_SetPPGInitLeadingEdge

Set the initial leading/trailing edge of PPG channels.

Prototype:

void

TMRD_SetPPGInitLeadingEdge(uint8_t ***PPGChannel***,

unt8_t ***WaveEdge***)

Parameters:

TDx is the specified TMRD unit.

PPGChannel is the specified PPG output channel, which can be

- **TMRD_PPG_CHANNEL_00**: PPG output signal 00.
- **TMRD_PPG_CHANNEL_01**: PPG output signal 01.
- **TMRD_PPG_CHANNEL_10**: PPG output signal 10.
- **TMRD_PPG_CHANNEL_11**: PPG output signal 11.

WaveEdge specifies the initial wave edge of leading edge, which can be

- **TMRD_WAVE_EDGE_RISING**: Leading edge is rising edge and trailing edge is falling edge.
- **TMRD_WAVE_EDGE_FALLING**: Leading edge is falling edge and trailing edge is rising edge.

Description:

This function will set the initial setting of leading edge/trailing edge of a PPG output signal specified by ***WaveEdge***.

Return:

None

13.2.3.9 TMRD_SetCMPRegWritePath

Set a write path to update the compare register.

Prototype:

void

TMRD_SetCMPRegWritePath(TMRD_UNIT_Channel ***CHx,***

uint8_t ***WritePath***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

WritePath is the path to update compare registers, which can be

- **TMRD_CMP_WRITE_DIRECT**: Directly written to the compare register by CPU instructions.
- **TMRD_CMP_WRITE_INDIRECT**: Write to compare register via the timer register. Enable flag is required.

Description:

This function will set a write path to update the compare register. When set ***WritePath*** to **TMRD_CMP_WRITE_DIRECT,** at the time when data is written into the timer register (TDmnRGx), the same value is written to the corresponding compare register (TDmnCPx) simultaneously. In this case, it is not necessary for an enable flag.

If set ***WritePath*** to **TMRD_CMP_WRITE_INDIRECT,** enable update flag is required by using function TMRD_EnableUpdateCMPReg(). About the update occasion, please refer to the description below in each mode:

In the timer mode, interlock timer mode:

- TDmnMOD<TDCLE>="0": A value in the compare register (TDmnCPx) is updated to the value of the timer register (TDmnRGx) when the COUNTER overflows.
- TDmnMOD<TDCLE>="1": A value in the compare register (TDmnCPx) is updated to the value of the timer register (TDmnRGx) when the value set in the comparator00/10 (CP00/CP10) matches the value in the counter.

In the PPG mode:

When the value set in the comparator00/10 (CP00/10) matches the value in the counter, the value in the compare register (TDmnCPx) is updated to the value in the timer register (TDmnRGx).

In the interlock PPG mode:

For TMR0, the update method is the same as in PPG mode, which described above.

For TMR1, when the value set in the comparator05 (CP05) matches the value in the compare register, the value of the compare register (TD1mCPx) is updated to the value of the timer register (TD1mRGx).

Return:

None

Note:

In the interlock PPG mode, writing path of TMR1 is selected as the values in TMR0, so only setting the writing path of TMR0 is enough.

13.2.3.10 TMRD_SetCMP0INTSrc

Set the interrupt source of compare interrupt 0.

Prototype:

void

TMRD_SetCMP0INTSrc(TMRD_UNIT_Channel ***CHx,***

uint8_t ***INTSrc***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

INTSrc selects the interrupt factor, which can be

- **TMRD_INT_NONE**: No interrupt factor.
- **TMRD_INT_MATCH_CYCLE**: A match signal from CP00/CP10.
- **TMRD_INT_MATCH_PHASE**: A match signal from CP05(only TMR0 has).
- **TMRD_INT_UC_OVERFLOW**: Overflow of COUNTER.

Description:

This function will choose the interrupt source of compare interrupt 0.

Return:

None

Note:

In the PPG mode, **TMRD_INT_UC_OVERFLOW** is invalid as an interrupt factor.

In the interlock PPG mode, **TMRD_INT_MATCH_CYCLE** of TMR1 becomes invalid as an interrupt factor.

TMRD_INT_MATCH_PHASE is invalid as an interrupt factor of TMR1.

13.2.3.11 TMRD_SetRunState

Set the count operation of TMRD.

Prototype:

Void

TMRD_SetRunState(TMRD_UNIT_Channel ***CHx,***

uint8_t ***RunState***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

RunState is the counter operation of TMRD, which can be:

- **TMRD_RUN:** Starts the count operation of TMRDx.
- **TMRD_STOP:** Stop the count operation of TMRDx and zero clears COUNTER.

Description:

This function will set the count operation of TMRD.

Return:

None

Note:

In interlock timer mode and interlock PPG mode, this function becomes invalid for TMR1, because TMR1 will start operation in tandem with COUNTER0 of TMR0.

13.2.3.12 TMRD_SetPhaseRelation

Set the phase relation of phase 1 to phase 0.

Prototype:

void

TMRD_SetPhaseRelation(uint8_t ***PhaseRelation***)

Parameters:

TDx is the specified TMRD unit.

PhaseRelation is the phase relation of phase 1 to phase 0, which can be:

- **TMRD_PHASE_DELAY_OR_SAME:** Phase 1 delays or the same as phase 0.
- **TMRD_PHASE_FAST_OR_SAME:** Phase 1 is fast or the same as phase 0.

Description:

This function will set the phase relation of phase 1 to phase 0.

Return:

None

Note:

This function is valid only in the interlock PPG mode. The output of the phase 0 and the phase 1 cannot be switched in the timer mode, the interlock mode and the PPG mode.

13.2.3.13 TMRD_EnableUpdateCMPReg

Enable to update the compare register.

Prototype:

void

TMRD_EnableUpdateCMPReg(uint8_t ***UpdateCHx***)

Parameters:

TDx is the specified TMRD unit.

UpdateCHx Enable flag for values of the compare registers:

- **TMRD_UPDATE_CH_00:** Update enable flag of CPRG00/ CPRG01/ CPRG02/ CPRG05.
- **TMRD_UPDATE_CH_01:** Update enable flag of CPRG03/ CPRG04.
- **TMRD_UPDATE_CH_10:** Update enable flag of CPRG10/ CPRG11/ CPRG12.
- **TMRD_UPDATE_CH_11:** Update enable flag of CPRG13/ CPRG14.

- combination of the channels above.

Description:

This function will set a enable flag to update the compare register, for more details, please refer to the section about TMRD_SetCMPRegWritePath().

Return:

None

Note:

In the interlock PPG mode, when use this function to set the update enable flag of TMR0, the enable flag of TMR1 will set at the same time, please don't use this function to set TMR1 again. And this flag will be zero cleared when a match of compare register 05(CP05) is detected.

13.2.3.14 TMRD_SetDMAReq

Enable or disable the DMA request of INTTDxnCMP0 signal.

Prototype:

void

TMRD_SetDMAReq(TMRD_UNIT_Channel ***CHx,***

FunctionalState ***NewState***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

NewState is the state of DMA request, which can be:

- **ENABLE:** Enable the DMA request.
- **DISABLE:** Disable the DMA request.

Description:

This function will enable and disable the DMA request of INTTDxnCMP0, which is a factor to generate a DMA request.

Return:

None

13.2.3.15 TMRD_SetInitTiming

Initialize TMRD timing parameters.

Prototype:

void

TMRD_SetInitTiming(TMRD_UNIT_Channel ***CHx,***

TMRD_TimingTypeDef* ***TimingStruct***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

TimingStruct is the pointer of TMRD timing parameters structure. Please refer to Data Structure for details.

Description:

This function will initialize TMRD timing parameters, which is included cycle timing, leading timing, trailing timing, phase shift timing and so on.

Return:

None

Note:

Please refer to Data Structure Description to get the setting range of each parameter in structure ***TimingStruct.***

CPRG0: ***TimingStruct***. Cycle

CPRG1: ***TimingStruct***. LeadingTiming0

CPRG2: (***TimingStruct***.TrailingTiming0<<4)||(***TimingStruct***.BitModulationRate0&0x0f)

CPRG3: ***TimingStruct***. LeadingTiming1

CPRG4: (***TimingStruct***.TrailingTiming1<<4)||(TimingStruct->BitModulationRate1&0x0f)

CPRG5: ***TimingStruct***. PhaseShiftTiming(invalid in TMR1)

13.2.3.16 TMRD_ChangeTiming

Change the specified TMRD timing value.

Prototype:

void

TMRD_ChangeTiming(uint8_t ***TimingType***,

uint16_t ***Timing***)

Parameters:

TDx is the specified TMRD unit.

TimingType specifies the timing parameter type of TMRD, which can be

- **TMRD_TIMING_TD0_CYCLE:** TMR0 cycle timing, CPRG00.
- **TMRD_TIMING_00_LEADING:** Signal 00 leading timing (CPRG01).
- **TMRD_TIMING_00_TRAILING:** Signal 00 trailing timing (CPRG02).
- **TMRD_TIMING_01_LEADING:** Signal 01 leading timing (CPRG03).
- **TMRD_TIMING_01_TRAILING:** Signal 01 trailing timing (CPRG04).
- **TMRD_TIMING_PHASE_SHIFT:** Phase shift timing (CPRG05).
- **TMRD_TIMING_TD1_CYCLE:** TMR1 cycle timing (CPRG10).
- **TMRD_TIMING_10_LEADING:** Signal 10 leading timing (CPRG11).
- **TMRD_TIMING_10_TRAILING:** Signal 10 trailing timing (CPRG12).
- **TMRD_TIMING_11_LEADING:** Signal 11 leading timing (CPRG13).
- **TMRD_TIMING_11_TRAILING:** Signal 11 trailing timing (CPRG14).

Timing is the 16 bits timing value.

Description:

This function will change the specified TMRD timing value. It is very useful for users to set a small quantity or one of timing value.

Return:

None

Note:

About the value of ***Timing***, please refer to Data Structure Description to get the setting range of each parameter.

13.2.3.17 TMRD_GetTiming

Get the specified TMRD timing value.

Prototype:

uint16_t

TMRD_GetTiming(uint8_t ***TimingType***)

Parameters:

TDx is the specified TMRD unit.

TimingType specifies the timing parameter type of TMRD, which can be

- **TMRD_TIMING_TD0_CYCLE:** TMR0 cycle timing, CPRG00.
- **TMRD_TIMING_00_LEADING:** Signal 00 leading timing (CPRG01).
- **TMRD_TIMING_00_TRAILING:** Signal 00 trailing timing (CPRG02).
- **TMRD_TIMING_01_LEADING:** Signal 01 leading timing (CPRG03).
- **TMRD_TIMING_01_TRAILING:** Signal 01 trailing timing (CPRG04).
- **TMRD_TIMING_PHASE_SHIFT:** Phase shift timing (CPRG05).
- **TMRD_TIMING_TD1_CYCLE:** TMR1 cycle timing (CPRG10).
- **TMRD_TIMING_10_LEADING:** Signal 10 leading timing (CPRG11).
- **TMRD_TIMING_10_TRAILING:** Signal 10 trailing timing (CPRG12).
- **TMRD_TIMING_11_LEADING:** Signal 11 leading timing (CPRG13).
- **TMRD_TIMING_11_TRAILING:** Signal 11 trailing timing (CPRG14).

Description:

This function will get the timing value from the compare register specified by ***TimingType***.

Return:

Specified timing value that is in compare register.

13.2.3.18 TMRD_SetBitModulationCycle

Set the 1bit modulation cycle.

Prototype:

void

TMRD_SetBitModulationCycle (uint8_t ***PPGChannel***,

uint8_t ***BitModCycle***)

Parameters:

TDx is the specified TMRD unit.

PPGChannel is the specified PPG output channel, which can be

- **TMRD_PPG_CHANNEL_00**: PPG output signal 00.
- **TMRD_PPG_CHANNEL_01**: PPG output signal 01.
- **TMRD_PPG_CHANNEL_10**: PPG output signal 10.
- **TMRD_PPG_CHANNEL_11**: PPG output signal 11.

BitModCycle specifies the 1-bit modulation cycle of TMRD, which can be

- **TMRD_1BITMOD_CYCLE_NONE**: without 1-bit modulation.
- **TMRD_1BITMOD_CYCLE_2TIMES**: the cycle defined with CP00/CP10 2fold.
- **TMRD_1BITMOD_CYCLE_4TIMES**: the cycle defined with CP00/CP10 4fold.
- **TMRD_1BITMOD_CYCLE_8TIMES**: the cycle defined with CP00/CP10 8fold.
- **TMRD_1BITMOD_CYCLE_16TIMES**: the cycle defined with CP00/CP10 16fold.

Description:

This function will select to 1-bit modulation cycle of output signal in the PPG mode and the interlock PPG mode.

Return:

None

Note:

In the timer mode and the interlock mode, fuction TMRD_SetBitModulationCycle () is ignored.

13.2.3.19 TMRD_SetBitModUpdateTiming

Set the 1bit modulation update timing.

Prototype:

void

TMRD_SetBitModUpdateTiming (uint8_t ***PPGChannel***,

FunctionalState ***NewState***)

Parameters:

TDx is the specified TMRD unit.

PPGChannel is the specified PPG output channel, which can be

- **TMRD_PPG_CHANNEL_00**: PPG output signal 00.
- **TMRD_PPG_CHANNEL_01**: PPG output signal 01.
- **TMRD_PPG_CHANNEL_10**: PPG output signal 10.
- **TMRD_PPG_CHANNEL_11**: PPG output signal 11.

NewState specified the update timing of each PPG channel, which can be:

- **DISABLE:** 1-bit modulation cycle.
- **ENABLE:** when detecting the match with CPxx.

Description:

This function will select to 1-bit modulation update timeing is 1-bit modulation or when detecting the match with CPxx in the PPG mode and the interlock PPG mode.

Return:

None

Note:

In the timer mode and the interlock mode, fuction TMRD_SetBitModUpdateTiming () is ignored.

13.2.3.20 TMRD_SetLOutHSWMode

Set the H-SW Mode when TDOUTIN is Low.

Prototype:

void

TMRD_SetLOutHSWMode(TMRD_UNIT_Channel ***CHx***,
TMRD_UNIT_Phase ***PHx***,
uint8_t ***LMode***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

LMode specified the H-SW mode when TDOUTIN is low, which can be:

- **TMRD_LOUT_HSW_CCW_CW:** ccw or cw mode.
- **TMRD_LOUT_HSW_BRAKE:** short brake mode.
- **TMRD_LOUT_HSW_STOP:** stop mode.

Description:

This function will set the H-SW Mode when TDOUTIN is Low.

Return:

None

13.2.3.21 TMRD_GetLOutHSWMode

Get the H-SW Mode when TDOUTIN is Low.

Prototype:

uint8_t

TMRD_GetLOutHSWMode(TMRD_UNIT_Channel ***CHx***,

TMRD_UNIT_Phase ***PHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

Description:

This function will get the H-SW Mode when TDOUTIN is Low.

Return:

The H-SW mode

- **TMRD_LOUT_HSW_CCW_CW:** ccw or cw mode.
- **TMRD_LOUT_HSW_BRAKE:** short brake mode.
- **TMRD_LOUT_HSW_STOP:** stop mode.

13.2.3.22 TMRD_EnableHSWMode

Enable H-SW mode.

Prototype:

void

TMRD_EnableHSWMode(TMRD_UNIT_Channel ***CHx***,

TMRD_UNIT_Phase ***PHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

Description:

This function will enable H-SW mode.

Return:

None

13.2.3.23 TMRD_DisableHSWMode

Disable H-SW mode.

Prototype:

void

TMRD_DisableHSWMode(TMRD_UNIT_Channel ***CHx***,

TMRD_UNIT_Phase ***PHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

Description:

This function will Disable H-SW mode.

Return:

None

13.2.3.24 TMRD_GetHSWModeState

Get H-SW mode.

Prototype:

FunctionalState

TMRD_GetHSWModeState(TMRD_UNIT_Channel ***CHx***,

TMRD_UNIT_Phase ***PHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

Description:

This function will get H-SW mode state.

Return:

The state of H-SW mode

- **ENABLE:** H-SW mode is enable.

- **DISABLE:** H-SW mode is disable.

13.2.3.25 TMRD_SetTDDIRLevel

Set the level of TDDIR

Prototype:

void

TMRD_SetTDDIR(TMRD_UNIT_Channel ***CHx***,
TMRD_UNIT_Phase ***PHx***,
uint8_t ***Level***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

Level is the setting level, which can be

- **TMRD_LOW_LEVEL:** Outputs "Low" level.
- **TMRD_HIGH_LEVEL:** Outputs "High" level.

Description:

This function will set the level of TDDIR.

Return:

None

13.2.3.26 TMRD_GetTDDIRLevel

Get the level of TDDIR

Prototype:

uint8_t

TMRD_GetTDDIR(TMRD_UNIT_Channel ***CHx***,
TMRD_UNIT_Phase ***PHx***)

Parameters:

TDx is the specified TMRD unit.

CHx is the specified TMRD channel.

PHx is the specified TMRD phase.

Description:

This function will get the level of TDDIR.

Return:

The level of TDDIR

- **TMRD_LOW_LEVEL:** Outputs "Low" level.
- **TMRD_HIGH_LEVEL:** Outputs "High" level.

13.2.4 Data Structure Description

13.2.4.1 TMRD_TimingTypeDef

Data Fields:

uint16_t

Cycle specifies the TMR0/1 cycle value, it will set to CPRG00/10.

uint16_t

LeadingTiming0 specifies the signal 00/10 leading timing (CPRG01/11).

uint16_t

TrailingTiming0 specifies the signal 00/10 trailing timing (CPRG02/12).

uint8_t

BitModulationRate0 specifies the rate of 1-bit modulation in channel 0.

uint16_t

LeadingTiming1 specifies the signal 01/11 leading timing (CPRG03/13).

uint16_t

TrailingTiming1 specifies the signal 01/11 trailing timing (CPRG04/14).

uint16_t

PhaseShiftTiming specifies the phase shift timing (CPRG05, only in TMRD0).

uint8_t

BitModulationRate1 specifies the rate of 1-bit modulation in channel 1.

Note:

Please refer to the tables below to get the setting range of each parameter in different mode.

14. TMRB

14.1 Overview

TOSHIBA TMPM06x contains 8 channels of TMRB (TMRB0 through TMRB7). Each channel consists of a 16-bit up-counter, two 16-bit timer registers (Double-buffered), two 16-bit capture registers, two comparators, a capture input control, a timer flip-flop and its associated control circuit. Each channel can operate in the following modes:

- Interval timer mode
- Event counter mode
- Programmable pulse generation (PPG) mode
- Programmable pulse generation (PPG) external trigger mode

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

- Frequency measurement
- Pulse width measurement

The TMRB driver APIs provide a set of functions to configure each channel, such as setting the clock division, trailingtiming and leadingtiming 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/TX00_Periph_Driver/src/tmpm06x_tmrb.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_tmrb.h containing the macros, data types, structures and API definitions for use by applications.

14.2 API Functions

14.2.1 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***);

- TMRB_INTMask TMRB_GetINTMask(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_GetRegisterValue(TSB_TB_TypeDef * ***TBx***, uint8_t ***Reg***);
- 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_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_SetDMAReq(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***DMAReq***);

14.2.2 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_GetINTMask(),TMRB_GetRegisterValue(),TMRB_GetUpCntValue() and TMRB_GetCaptureValue().
4) TMRB_SetFlipFlop(), TMRB_SetINTMask(), TMRB_SetSyncMode(), TMRB_SetDoubleBuf(), TMRB_SetExtStartTrg() and TMRB_SetClkInCoreHalt (), TMRB_SetDMAReq() handle other specified functions.

14.2.3 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.

14.2.3.1 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

14.2.3.2 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

14.2.3.3 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

14.2.3.4 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, leadingtiming value, trailingtiming 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, trailingtiming and leadingtiming duration for the specified TMRB channel selected by ***TBx***.

Return:

None

14.2.3.5 TMRB_SetCaptureTiming

Configure the capture timing.

Prototype:

void

TMRB_SetCaptureTiming(TSB_TB_TypeDef* ***TBx***,

uint32_t ***CaptureTiming***)

Parameters:

TBx is the specified TMRB channel.

TSB_TB0, TSB_TB1, TSB_TB2,

TSB_TB3, TSB_TB4, TSB_TB5.

CaptureTiming specifies TMRB capture timing, which can be

- **TMRB_DISABLE_CAPTURE**: Disable the capture function of the specified TMRB channel.
- **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_FF_RISING_FALLING**: Takes count values into capture register 0 (TBxCP0) upon rising of TBxFF0 pin input and takes count values into capture register 1 (TBxCP1) upon falling of TBxFF0 pin input.

Description:

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_FF_RISING_FALLING**, then at the time of the rising edge of port TBxFF0 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 TBxFF0 pin input.

The flip-flop output of TMRB6, TMRB7 can be used as the capture trigger of other channels.

TMRB0~2: TB6OUT

TMRB3~5: TB7OUT

Return:

None

14.2.3.6 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

14.2.3.7 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 leadingtiming value; bit of **MatchTrailingTiming** Indicates if the up-counter matches with trailingtiming 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 leadingtiming value is detected

MatchTrailingTiming(Bit1): a match with the trailingtiming 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
}
```

14.2.3.8 TMRB_GetINTMask

Indicate what causes the interrupt be masked.

Prototype:
TMRB_INTMask
TMRB_GetINTMask (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 be masked. Bit of **MatchLeadingTimingMask** indicates if the up-counter matches with leadingtiming value interrupt is masked; bit of **MatchTrailingTimingMask** Indicates if the up-counter matches with trailingtiming value interrupt is masked, and bit of **OverflowMask** indicates if overflow had occurred interrupt is masked.

Return:

TMRB Interrupt factor be masked. Each bit has the following meaning:
MatchLeadingTimingMask(Bit0): a match with the leadingtiming value interrupt is masked. **MatchTrailingTimingMask**(Bit1): a match with the trailingtiming value interrupt is masked.
OverFlowMask(Bit2): an up-counter is overflow interrupt is masked.

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

```
TMRB_INTMask factor = TMRB_GetINTMask(TSB_TB0);
if (factor.Bit.MatchLeadingTimingMask) {
    // Do A
}

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

if (factor.Bit.OverFlowMask) {
    // Do C
}
```

14.2.3.9 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 trailingtiming are match.
- **TMRB_MASK_MATCH_LEADINGTIMING_INT**: Mask the interrupt the factor of which is that the value in up-counter and leadingtiming 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 trailingtiming 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 leadingtiming 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

14.2.3.10 TMRB_ChangeLeadingTiming

Change the value of leadingtiming 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 leadingtiming, max is 0xFFFF.

Description:

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

Return:

None

***Note:**

LeadingTiming cannot exceed ***TrailingTiming***.

14.2.3.11 TMRB_ChangeTrailingTiming

Change the value of trailingtiming 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 trailingtiming, max is 0xFFFF.

Description:

This function will specify the absolute value of trailingtiming for the specified TMRB. The actual interval of trailingtiming 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.

14.2.3.12 TMRB_GetRegisterValue

Get register value of the specified TMRB channel.

Prototype:

uint16_t

TMRB_GetRegisterValue(TSB_TB_TypeDef* ***TBx,***

uint8_t ***Reg***)

Parameters:

TBx is the specified TMRB channel.

Reg is used to choose to return the value of register0 or to return the value of

register1, which can be one of the following,

- **TMRB_Reg_0:** specifying register0.
- **TMRB_Reg_1:** specifying register1.

Description:

This function will return the value in register of the specified TMRB channel.

Return:

The value of register

14.2.3.13 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

14.2.3.14 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.

TSB_TB0, TSB_TB1, TSB_TB2,

TSB_TB3, TSB_TB4, TSB_TB5.

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

14.2.3.15 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.

TSB_TB0, TSB_TB1, TSB_TB2,

TSB_TB3, TSB_TB4, TSB_TB5.

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

14.2.3.16 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_TB1, TSB_TB2, TSB_TB3,

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 TMRB1 through TMRB3, their start timing is synchronized with TMRB0. If the synchronous mode is enabled for TMRB5 through TMRB7, their start timing is synchronized with TMRB4.

Return:

None

***Note:**

TMRB1 through TMRB3, TMRB5 through TMRB7 must start counting by calling **TMRB_SetRunState()** before TMRB0, TMRB4 start counting, so that start timing can be synchronized.

14.2.3.17 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

14.2.3.18 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

14.2.3.19 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

14.2.3.20 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.

TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3,

TSB_TB4, TSB_TB5, TSB_TB6, TSB_TB7.

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.

14.2.4 Data Structure Description

14.2.4.1 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 trailingtiming value to be written into TBnRG1, max. is 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 trailingtiming, 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 leadingtiming value to be written into TBnRG0, max. is 0xFFFF, and it cannot be set larger than **TrailingTiming**.

14.2.4.2 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 trailingtiming,
- **TMRB_FLIPFLOP_MATCH_LEADINGTIMING**, which means that the reversing flip-flop output will be triggered when the up-counter matches the leadingtiming.

14.2.4.3 TMRB_INTFactor

Data Fields:

uint32_t

All: TMRB interrupt factor.

Bit

uint32_t

MatchLeadingTiming: 1 a match with the leadingtiming value is detected

uint32_t

MatchTrailingTiming: 1 a match with the trailingtiming value is detected

uint32_t

OverFlow: 1 an up-counter is overflow

uint32_t

Reserverd: 29 -

14.2.4.4 TMRB_INTMask

Data Fields:

uint32_t

All: TMRB interrupt factor be masked.

Bit

uint32_t

MatchLeadingTimingMask: 1 a match with the leadingtiming value interrupt is masked.

uint32_t

MatchTrailingTimingMask: 1 a match with the trailingtiming value interrupt is masked.

uint32_t

OverFlowMask: 1 an up-counter is overflow interrupt is masked.

uint32_t

Reserverd: 29 -

15. TSPI

15.1 Overview

TSPI is a serial interface that can communicate in synchronous full-duplex communication. It has SPI and SIO

modes, so that it can perform high-speed serial transfer between various peripheral devices.

TSPI has chip select signal pins (TSPIxCS0, TSPIxCS1 and TSPIxCSIN), a serial clock signal pin (TSPIxSCK)

and serial transmit and receive data signal pins (TSPIxTXD and TSPIxRXD).

Data length can be arranged by 1 bit from 7 bits (with parity) to 32 bits (without parity).

15.2 API Functions

15.2.1 Function List

- void TSPI_SWReset(TSB_TSPI_TypeDef * TSPIx);
- void TSPI_Enable(TSB_TSPI_TypeDef * TSPIx);
- void TSPI_Disable(TSB_TSPI_TypeDef * TSPIx);
- void TSPI_SetTxRxCtrl(TSB_TSPI_TypeDef * TSPIx, FunctionalState NewState);
- FunctionalState TSPI_GetTxRxCtrl(TSB_TSPI_TypeDef * TSPIx);
- void TSPI_SelectMode(TSB_TSPI_TypeDef * TSPIx, TSPI_Mode Mode);
- void TSPI_SelectMstr(TSB_TSPI_TypeDef * TSPIx, TSPI_Mstr Mstr);
- void TSPI_SelectTransferMode(TSB_TSPI_TypeDef * TSPIx, TSPI_TransferMode TrMode);
- void TSPI_SetCS(TSB_TSPI_TypeDef * TSPIx, TSPI_CSx CSx);
- void TSPI_SetTransferNum(TSB_TSPI_TypeDef * TSPIx, uint8_t Num);

- void TSPI_SetTxOutValueInIdle(TSB_TSPI_TypeDef * TSPIx, TSPI_TxOutValueInIdle OutputValue);
- void TSPI_SetTxOutValueInAndorran(TSB_TSPI_TypeDef * TSPIx, TSPI_TxOutValueInAndorran OutputValue);
- void TSPI_SetFIFOLevelINT(TSB_TSPI_TypeDef * TSPIx, uint8_t TxRx, uint8_t Level);
- void TSPI_SetDMA(TSB_TSPI_TypeDef * TSPIx, uint8_t TxRx, FunctionalState NewState);
- void TSPI_SetINT(TSB_TSPI_TypeDef * TSPIx, uint32_t IntSrc, FunctionalState NewState);
- void TSPI_InitFIFO(TSB_TSPI_TypeDef * TSPIx, uint8_t TxRx);
- void TSPI_SetBaudRate(TSB_TSPI_TypeDef * TSPIx, TSPI_BaudClock Clk, uint8_t Divider);
- void TSPI_Init(TSB_TSPI_TypeDef * TSPIx, TSPI_InitTypeDef * Init);
- void TSPI_SetTxData(TSB_TSPI_TypeDef * TSPIx, uint32_t dat);
- uint32_t TSPI_GetRxData(TSB_TSPI_TypeDef * TSPIx);
- FunctionalState TSPI_IsRegisterModifiable(TSB_TSPI_TypeDef * TSPIx);
- TSPI_StatusFlag TSPI_GetStatus(TSB_TSPI_TypeDef * TSPIx);
- TSPI_Err TSPI_GetErrorState(TSB_TSPI_TypeDef * TSPIx);
- void TSPI_ClrAllErrFlag(TSB_TSPI_TypeDef * TSPIx);

15.2.2 Detailed Description

Functions listed above can be divided into five parts:

1) Configure and control the common functions of each TSPI channel:
   TSPI_SWReset(), TSPI_Enable(),TSPI_Disable(),

   TSPI_SelectMode(), TSPI_SelectMstr(),TSPI_SelectTransferMode(),

   TSPI_ SetTxOutValueInIdle (),TSPI_SetTxOutValueInAndorran (), TSPI_Init(),

   TSPI_SetTransferNum(),TSPI_SetBaudRate().
2) Configure FIFO and DMA:
   TSPI_SetFIFOLevelINT(),TSPI_SetDMA(),TSPI_InitFIFO().
3) Transfer control:
   TSPI_SetTxData(),TSPI_GetRxData(),TSPI_SetTxRxCtr(),TSPI_SetCS().
4) The status indication of each TSPI channel :
   TSPI_GetStatus(),TSPI_GetErrorState(),

   TSPI_IsRegisterModifiable(),TSPI_GetTxRxCtrl(),TSPI_ClrAllErrFlag()
5) Configure interrupt,
   TSPI_SetINT().

15.2.3 Function Documentation

Note: In all of the following APIs, the parameter “TSB_TSPI_TypeDef * TSPIx” can be one of the following values:

TSB_TSPI0

15.2.3.1 TSPI_SWReset

Software reset TSPI module.

Prototype:

void

TSPI_SWReset(TSB_TSPI_TypeDef * ***TSPIx***)

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function software reset the specified TSPI channel selected by ***TSPIx***.

Return:

None.

15.2.3.2 TSPI_Enable

Enable the specified TSPI channel.

Prototype:

void

TSPI_Enable(TSB_TSPI_TypeDef * ***TSPIx***)

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function enables the specified TSPI channel selected by ***TSPIx***.

Return:

None.

Note:

This function must be called at first.

15.2.3.3 TSPI_Disable

Disable the specified TSPI channel

Prototype:

void

TSPI_Disable(TSB_TSPI_TypeDef * ***TSPIx***)

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function disables the specified TSPI channel selected by ***TSPIx***.

Return:

None.

15.2.3.4 TSPI_SetTxRxCtrl

Enable/Disable TSPI Transfer.

Prototype:

void

TSPI_SetTxRxCtrl (TSB_TSPI_TypeDef * ***TSPIx***,

FunctionalState ***NewState***)

Parameters:

TSPIx is the specified TSPI channel.

NewState specify the state for transfer, which can be:

ENABLE for enable transfer
DISABLE for disable transfer

Description:

This function enables/disables TSPI transfer of the specified TSPI channel selected by ***TSPIx***.

Return:

None.

15.2.3.5 TSPI_GetTxRxCtrl

Get the TSPI transfer control status.

Prototype:

FunctionalState

TSPI_GetTxRxCtrl(TSB_TSPI_TypeDef * TSPIx)

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function gets the TSPI transfer control status

Return:

The transfer control status of specify TSPI unit:

ENABLE: Specified TSPI channel is enable to transfer or receive.

DISABLE: Specified TSPI channel is disable to transfer or receive.

15.2.3.6 TSPI_SelectMode

Select TSPI mode: as SIO or SPI.

Prototype:

void

TSPI_SelectMode(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_Mode ***Mode***)

Parameters:

TSPIx is the specified TSPI channel.

Mode Select TSPI mode: as SIO or SPI. This parameter can be:

- **TSPI_MODE_SPI**: The TSPI channel work as SPI mode (Using TSPIxCS pins) .
- **TSPI_MODE_SIO**: The TSPI channel work as SIO mode (Not using TSPIxCS pins) .

Description:

This function is used to select TSPI mode: as SIO or SPI.

Return:

None.

15.2.3.7 TSPI_SelectMstr

Select master/slave.

Prototype:

void

TSPI_SelectMstr(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_Mstr ***Mstr***)

Parameters:

TSPIx is the specified TSPI channel.

Mstr Specify the master/slave. This parameter can be:

- **TSPI_MSTR_SLAVE**: Slave operation.
- **TSPI_MSTR_MASTER**: Master operation

Description:

This function is used to select master/slave.

Return:

None.

15.2.3.8 TSPI_SelectTransferMode

Select TSPI transfer mode

Prototype:

void

TSPI_SelectTransferMode(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_TransferMode ***TrMode***)

Parameters:

TSPIx is the specified TSPI channel.

TrMode Specify the transfer mode for TSPI. This parameter can be:

- **TSPI_TRMODE_TX**: The specified TSPI channel work as Half duplex (Transmit) mode.
- **TSPI_TRMODE_RX**: The specified TSPI channel work as Half duplex (Receive) mode.
- **TSPI_TRMODE_TXRX**: The specified TSPI channel work as Full duplex mode.

Description:

This function can make the specified TSPI work as Half duplex (Transmit) mode, or Half duplex (Receive) mode, or Full duplex mode.

Return:

None.

15.2.3.9 TSPI_SetCS

Select chip selection pin.

Prototype:

void

TSPI_SetCS(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_CSx ***CSx***)

Parameters:

TSPIx is the specified TSPI channel.

CSx Specify the chip selection pin. This parameter can be:

- **TSPI_CS0**: enable chip connect to CS0.
- **TSPI_CS1**: enable chip connect to CS1.

Description:

This function helps to select chip by change the voltage level of the selection pin.

Return:

None.

15.2.3.10 TSPI_SetTransferNum

Set the number of transfer frame.

Prototype:

void

TSPI_SetTransferNum(TSB_TSPI_TypeDef * ***TSPIx***,

uint8_t ***Num***)

Parameters:

TSPIx is the specified TSPI channel.

Num Specify the number of transferring, This parameter can be:

- **1**: One frame(Single transfer)
- **2-255**: 2 to 255 frames(Burst transfer)

Description:

this function will set the number of transfer frame.

Return:

None.

15.2.3.11 TSPI_ SetOutValueInIdle

Fixed output value function control when TSPIxTXD idles

Prototype:

void

TSPI_SetTxOutValueInIdle(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_TxOutValueInIdle ***OutputValue***);

Parameters:

TSPIx is the specified TSPI channel.

OutputValue Specify output value when TSPIxTXD idles.This parameter can be:

- **TSPI_TXPOUTVALUE_HIZ**: Keep Hi-z.
- **TSPI_TXPOUTVALUE_LASTDATA**:Last data in previous transfer.
- **TSPI_TXPOUTVALUE_KEEPLOW**: Fixed to low.
- **TSPI_TXPOUTVALUE_KEEPHIGH**: Fixed to high.

Description:

This function fixed output value function control when TSPIxTXD idles.

Return:

None.

15.2.3.12 TSPI_ SetOutValueInAndorran

Fixed output value function control when TSPIxTXD andorrans

Prototype:

void

TSPI_SetTxOutValueInAndorran(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_TxOutValueInAndorran ***OutputValue***);

Parameters:

TSPIx is the specified TSPI channel.

OutputValue Specify output value when TSPIxTXD andorrans.This parameter can be:

- **TSPI_TXPOUTVALUE_FIXEDTOLOW**: Fixed to low.
- **TSPI_TXPOUTVALUE_FIXEDTOHIGH**: Fixed to high.

Description:

This function fixed output value function control when TSPIxTXD andorrans.

Return:

None.

15.2.3.13 TSPI_SetFIFOLevelINT

Set the FIFO fill level to generate transfer (Tx or Rx) interrupt

Prototype:

void

TSPI_SetFIFOLevelINT(TSB_TSPI_TypeDef * ***TSPIx***,
uint8_t ***TxRx***,
uint8_t ***Level***)

Parameters:

TSPIx is the specified TSPI channel.

TxRx specify transmit or receive. This parameter can be:

- **TSPI_TX**: specify transmit mode.
- **TSPI_RX**: specify receive mode.

Level specify the FIFO fill level to generate interrupt. This parameter can be: 0 to 8 (depended on theData format)

Description:

This function sets the FIFO fill level to generate transfer (Tx or Rx) interrupt

Return:

None.

15.2.3.14 TSPI_SetDMA

Set Transfer DMA request control.

Prototype:

void

TSPI_SetDMA(TSB_TSPI_TypeDef * ***TSPIx***,
uint8_t ***TxRx***,
FunctionalState ***NewState***)

Parameters:

TSPIx is the specified TSPI channel.

TxRx specify transmit or receive. This parameter can be:

- **TSPI_TX**: specify transmit mode.
- **TSPI_RX**: specify receive mode.

NewState specify the state of transfer DMA request, which can be one of the following,

- **ENABLE:** enable DMA of the specified TSPI channel.
- **DISABLE:** disable DMA of the specified TSPI channel.

Description:

This function can enable or disable the DMA control of specified TSPI channel.0

Return:

None.

15.2.3.15 TSPI_SetINT

Enable/Disable the interrupt source.

Prototype:

void

TSPI_SetINT(TSB_TSPI_TypeDef * ***TSPIx***,

uint32_t ***IntSrc***,

FunctionalState ***NewState***)

Parameters:

TSPIx is the specified TSPI channel.

IntSrc specify the interrupt source to be set, which can be one of the following,

- **TSPI_INT_PERR: Parity error interrupt control.**
- **TSPI_INT_RXEND: Receive completion interrupt control.**
- **TSPI_INT_RXFIFO: Receive FIFO interrupt control.**
- **TSPI_INT_TXEND: Transmit completion interrupt control.**
- **TSPI_INT_TXFIFO: Transmit FIFO interrupt control.**
- **TSPI_INT_ALL (all above interrupt source)**

NewState specify the state of transfer DMA request, which can be one of the following,

- **ENABLE:** enable the Interrupt of the specified TSPI channel.
- **DISABLE:** disable Interrupt of the specified TSPI channel.

Description:

This function enables/disables the interrupt source of specified TSPI channel.

Return:

None.

15.2.3.16 TSPI_InitFIFO

Initialize transfer FIFO pointer.

Prototype:

void

TSPI_InitFIFO(TSB_TSPI_TypeDef * ***TSPIx***,

uint8_t ***TxRx***);

Parameters:

TSPIx is the specified TSPI channel.

TxRx specify transmit or receive. This parameter can be:

- **TSPI_TX**: specify transmit mode.
- **TSPI_RX**: specify receive mode.

Description:

This function initializes transfer FIFO pointer for the specified TSPI channel.

Return:

None.

15.2.3.17 TSPI_SetBaudRate

Set the clock and divider for baud rate generator

Prototype:

void

TSPI_SetBaudRate(TSB_TSPI_TypeDef * ***TSPIx***,

TSPI_BaudClock ***Clk***,

uint8_t ***Divider***);

Parameters:

TSPIx is the specified TSPI channel.

Clk Specify the input clock to the baud rate generator. This parameter can be:

- **TSPI_PHIT0_DIVIDE_2 :** Input clock to the baud rate generator T0/2
- **TSPI_PHIT0_DIVIDE_4 :** Input clock to the baud rate generator T0/4
- **TSPI_PHIT0_DIVIDE_8 :** Input clock to the baud rate generator T0/8
- **TSPI_PHIT0_DIVIDE_16 :** Input clock to the baud rate generator T0/16
- **TSPI_PHIT0_DIVIDE_32 :** Input clock to the baud rate generator T0/32
- **TSPI_PHIT0_DIVIDE_64 :** Input clock to the baud rate generator T0/64

- **TSPI_PHIT0_DIVIDE_128 :** Input clock to the baud rate generator T0/128
- **TSPI_PHIT0_DIVIDE_256 :** Input clock to the baud rate generator T0/256
- **TSPI_PHIT0_DIVIDE_512 :** Input clock to the baud rate generator T0/512
- **TSPI_PHIT0_DIVIDE_1024:** Input clock to the baud rate generator T0/1024

Divider Specify the division ratio "N". This parameter can be;

- **1 to 16**

Description:

This function sets the clock and divider for baud rate generator of the specified TSPI channel.

Return:

None.

15.2.3.18 TSPI_Init

Initialize the specified TSPI channel.

Prototype:

```
void
TSPI_Init(TSB_TSPI_TypeDef * TSPIx,
          TSPI_InitTypeDef * Init);
```

Parameters:

TSPIx is the specified TSPI channel.

Init is the structure containing basic TSPI configuration.

Description:

This function initializes the specified TSPI channel with the structure TSPI_InitTypeDef .

Return:

None.

Note:

Call TSPI_SetTransferNum() function at first

15.2.3.19 TSPI_SetTxData

Access to transmit FIFO.

Prototype:

void

TSPI_SetTxData(TSB_TSPI_TypeDef * ***TSPIx***,

uint32_t ***Dat***);

Parameters:

TSPIx is the specified TSPI channel.

Dat: The data which will be sent

Description:

This function writes data to transmit FIFO.

Return:

None.

15.2.3.20 TSPI_GetRxData

Access to receive FIFO.

Prototype:

uint32_t

TSPI_GetRxData(TSB_TSPI_TypeDef * ***TSPIx***);

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function initializes transfer FIFO pointer for the specified TSPI channel.

Return:

The received data.

15.2.3.21 TSPI_IsRegisterModifiable

Check if TSPI registers is modifiable.

Prototype:

FunctionalState

TSPI_IsRegisterModifiable(TSB_TSPI_TypeDef * ***TSPIx***);

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function checks if TSPI registers is modifiable.

Return:

The modifiable status of TSPI register.it can be one of the following values:

- **ENABLE** :The state when TSPI registers can be modified.
- **DISABLE**:The state when TSPI registers can be modified.

15.2.3.22 TSPI_GetStatus

Get the TSPI status.

Prototype:

TSPI_StatusFlag

TSPI_GetStatus(TSB_TSPI_TypeDef * ***TSPIx***);

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function returns the TSPI status of the specified TSPI channel.

Return:

The structure of the Tx/Rx FIFO, Run, Interrupt status of TSPI unit

15.2.3.23 TSPI_GetError

Get the error flag of TSPI transfer.

Prototype:

TSPI_Err

TSPI_GetErrorState (TSB_TSPI_TypeDef * ***TSPIx***);

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function gets the error flag of TSPI transfer.

Return:

The structure of the error state of transfer buffer.

15.2.3.24 TSPI_ ClrAllErrFlag

Clear all TSPI error flag.

Prototype:

void

TSPI_ ClrAllErrFlag (TSB_TSPI_TypeDef * ***TSPIx***);

Parameters:

TSPIx is the specified TSPI channel.

Description:

This function clear all TSPI error flag.

Return:

None.

15.2.4 Data Structure Description

15.2.4.1 TSPI_InitTypeDef

Data Fields:

uint8_t

DataDirection Transfer direction selection, which can be set as:

- **TSPI_LSB_FIRST:** transfer begin from the LSB
- **TSPI_MSB_FIRST:** transfer begin from the MSB

Uint8_t

FrameLength The length of frame, which can be set as:8 to 32.

Uint8_t

CycleBetweenFrames Interval cycle between frames in burst mode,whitch can be : 0 to 15.

Uint8_t

ClkPolority Level of serial clock during idle time,which can be set as:

- **TSPI_CS_ACTIVE_LOW:** Low level
- **TSPI_CS_ACTIVE_HIGH:** High level

uint8_t

ShortestIdleCycle Shortest idle cycle setting,which can be 1 to 15.

uint8_t

CS1Polarity Negative logic or Positive logic for pin CS1,which can be set as :

- **TSPI_CS_POLARITY_NEGATIVE:** Negative logic
- **TSPI_CS_POLARITY_POSITIVE:** Positive logic

uint8_t

CS0Polarity Negative logic or Positive logic for pin CS0,which can be set as :

- **TSPI_CS_POLARITY_NEGATIVE:** Negative logic
- **TSPI_CS_POLARITY_POSITIVE:** Positive logic

uint8_t

DelayCycleNum_CS_SCLK Delay cycle from asserting TSPIxCS to output serial clock, which can be 1 to 16.

uint8_t

DelayCycleNum_Negate_CS Delay cycle of negating TSPIxCS,which can be 1 to 16

uint8_t

LastBit_Hold_Time bit holding time of TSPIxTXD pin in SIO slave mode, which can be set as :

- **TSPI_HOLDTIME_FC_DIVIDE_2:** fc/2
- **TSPI_HOLDTIME_FC_DIVIDE_4:** fc/4
- **TSPI_HOLDTIME_FC_DIVIDE_8:** fc/8
- **TSPI_HOLDTIME_FC_DIVIDE_16:** fc/16
- **TSPI_HOLDTIME_FC_DIVIDE_32:** fc/32
- **TSPI_HOLDTIME_FC_DIVIDE_64:** fc/64
- **TSPI_HOLDTIME_FC_DIVIDE_128:** fc/128

FunctionalState

ParityCheck parity can be added in transfer. it can be set as :

- **ENABLE:** Enable parity check.
- **DISABLE:** Disable parity check.

uint8_t

Parity Select the kind of parity check, which can be set as :

- **TSPI_PARITY_ EVEN:** Enable even parity check.
- **TSPI_PARITY_ODD:** Enable odd parity check

15.2.4.2 TSPI_StatusFlag

Data Fields:

uint32_t

All: TSPI status flag.

Bit

uint32_t

RxFIFOFillLevel : 4 Receive FIFO fill level

uint32_t

RxFIFOFull : 1 Receive FIFO is full

uint32_t

RxFIFO_INT : 1 Receive FIFO interrupt generated

uint32_t

RxCompleted : 1 Reception completed

uint32_t

RxRUN : 1 Receive shift is operating

uint32_t

Reserved1 : 8 Reserved

uint32_t

TxFIFOFillLevel : 4 Transmit FIFO fill level

uint32_t

TxFIFOEmpty : 1 Transmit FIFO is empty

uint32_t

TxFIFO_INT : 1 Transmit FIFO interrupt generated

uint32_t

TxCompleted : 1 Transmission completed

uint32_t

TxRUN : 1 Transmit shift is operating

uint32_t

Reserved2 : 7 Reserved

uint32_t

TSPI_RegUsing : 1 TSPI registers must not be modified.

15.2.4.3 TSPI_Err

Data Fields:

uint32_t

All: TSPI Error flag register.

Bit

uint32_t

Reserved1: 9 Reserved

uint32_t

ParityErr: 1 Parity error flag

uint32_t

OverrunErr: 1 Overrun error flag

uint32_t

UnderrunErr: 1 Underrun error flag

uint32_t

Reserved2: 20 Reserved

16. UART

16.1 Overview

TMPM06x has two serial I/O channels. Each channel can operate in both UART mode (asynchronous communication) and I/O Interface mode(synchronous 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/TX00_Periph_Driver/src/tmpm06x_uart.c, with /Libraries/TX00_Periph_Driver/inc/tmpm06x_uart.h containing the macros, data types, structures and API definitions for use by applications.

16.2 API Functions

16.2.1 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_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_TRxAutoDisable ***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***);
- void UART_TxBufferClear(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 UART_SetRxDMAReq (TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetTxDMAReq (TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetInputClock(TSB_SC_TypeDef * ***UARTx***, uint32_t ***clock***)
- void SIO_SetInputClock(TSB_SC_TypeDef * ***SIOx***, uint32_t ***Clock***)
- void SIO_Enable(TSB_SC_TypeDef* ***SIOx***)
- void SIO_Disable(TSB_SC_TypeDef* ***SIOx***)
- void SIO_Init(TSB_SC_TypeDef* ***SIOx***,
  uint32_t ***IOClkSel***,
  UART_InitTypeDef* ***InitStruct***)
- uint8_t SIO_GetRxData(TSB_SC_TypeDef* ***SIOx***)
- void SIO_SetTxData(TSB_SC_TypeDef* ***SIOx***, uint8_t ***Data***)

16.2.2 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_SetInputClock(),UART_Init() and UART_DefaultConfig(),SIO_Enable(), SIO_Disable(),SIO_SetInputClock(),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_TxBufferClear (),UART_GetRxFIFOFillLevelStatus(),

   UART_GetRxFIFOOverRunStatus(),UART_GetTxFIFOFillLevelStatus(),and

   UART_GetTxFIFOUnderRunStatus(),

16.2.3 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.

16.2.3.1 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

16.2.3.2 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

16.2.3.3 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.

16.2.3.4 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

16.2.3.5 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, and 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, and transfer mode and flow control for the specified UART channel selected by ***UARTx***.

Return:

None

16.2.3.6 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

16.2.3.7 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

16.2.3.8 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

16.2.3.9 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.

16.2.3.10 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

16.2.3.11 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

16.2.3.12 UART_FIFOConfig

Enable or disable 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 specified UART channel selected by ***UARTx*** in UART FIFO when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:

None

16.2.3.13 UART_SetFIFOTransferMode

Transfer mode setting.

Prototype:

void

UART_SetFIFOTransferMode (TSB_SC_TypeDef* ***UARTx***,

uint32_t ***TransferMode***);

Parameters:

UARTx is the specified UART channel.

TransferMode Transfer mode.

This parameter can be one of the following values:

UART_TRANSFER_PROHIBIT, UART_TRANSFER_HALFDPX_RX, UART_TRANSFER_HALFDPX_TX or UART_TRANSFER_FULLDPX.

Description:

Transfer mode setting.

Return:

None

16.2.3.14 UART_TRxAutoDisable

Controls automatic disabling of transmission and reception.

Prototype:

void

UART_TRxAutoDisable (TSB_SC_TypeDef* ***UARTx***,

UART_TRxAutoDisable ***TRxAutoDisable***);

Parameters:

UARTx is the specified UART channel.

TRxAutoDisable Disabling transmission and reception or not

This parameter can be one of the following values:

UART_RXTXCNT_NONE or UART_RXTXCNT_AUTODISABLE .

Description:

Controls automatic disabling of transmission and reception.

Return:

None

16.2.3.15 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 new state of receive interrupt for receive FIFO.

This parameter can be one of the following values:

ENABLE or DISABLE

Description:

Enable or disable receive interrupt for receive FIFO.

Return:

None

16.2.3.16 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 new state of transmit interrupt for transmit FIFO.

This parameter can be one of the following values:

ENABLE or DISABLE

Description:

Enable or disable transmit interrupt for transmit FIFO.

Return:

None

16.2.3.17 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 bytes used in receive FIFO.

This parameter can be one of the following values:

UART_RXFIFO_MAX or UART_RXFIFO_RXFLEVEL

Description:

Bytes used in receive FIFO.

Return:

None

16.2.3.18 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 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:

Receive FIFO fill level to generate receive interrupts.

Return:

None

16.2.3.19 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 RX interrupt generation condition.

This parameter can be one of the following values:

UART_RFIS_REACH_FLEVEL or UART_RFIS_REACH_EXCEED_FLEVEL

Description:

Select RX interrupt generation condition.

Return:

None

16.2.3.20 UART_RxFIFOClear

Receive FIFO clear.

Prototype:

void

UART_RxFIFOClear (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Receive FIFO clear.

Return:

None

16.2.3.21 UART_TxFIFOFillLevel

Transmit FIFO fill level to generate transmit interrupts.

Prototype:

void
UART_TxFIFOFillLevel (TSB_SC_TypeDef* ***UARTx***,
uint32_t ***TxFIFOLevel***);

Parameters:

UARTx is the specified UART channel.

TxFIFOLevel is transmit 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:

Transmit FIFO fill level to generate transmit interrupts.

Return:

None

16.2.3.22 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 TX interrupt generation condition.

This parameter can be one of the following values:

UART_TFIS_REACH_FLEVEL or UART_TFIS_REACH_NOREACH_FLEVEL.

Description:

Select TX interrupt generation condition.

Return:

None

16.2.3.23 UART_TxFIFOClear

Transmit FIFO clear.

Prototype:

void

UART_TxFIFOClear (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Transmit FIFO clear.

Return:

None

16.2.3.24 UART_TxBufferClear

Transmit buffer clear.

Prototype:

void

UART_TxBufferClear (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Transmit buffer clear.

Return:

None

16.2.3.25 UART_GetRxFIFOFillLevelStatus

Status of receive FIFO fill level.

Prototype:

uint32_t

UART_GetRxFIFOFillLevelStatus (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Status of receive 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_4B: TX FIFO fill level is 4 bytes.

16.2.3.26 UART_GetRxFIFOOverRunStatus

Receive FIFO overrun.

Prototype:

uint32_t

UART_ GetRxFIFOOverRunStatus (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Receive FIFO overrun.

Return:

UART_RXFIFO_OVERRUN: Flags for RX FIFO overrun.

16.2.3.27 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_4B: TX FIFO fill level is 4 bytes.

16.2.3.28 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.

16.2.3.29 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

16.2.3.30 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

16.2.3.31 UART_SetInputClock

Selects input clock for prescaler.

Prototype:

void

UART_SetInputClock (TSB_SC_TypeDef * UARTx,

uint32_t clock)

Parameters:

UARTx is the specified UART channel.

Clock is Selects input clock for prescaler as PhiT0/2 or PhiT0.

This parameter can be one of the following values:

0: PhiT0/2

1: PhiT0

Description:

This function will select the specified UART channel by ***UARTx*** and specified the input clock for prescaler by ***clock***

Return:

None

16.2.3.32 SIO_SetInputClock

Selects input clock for prescaler.

Prototype:

void

SIO_SetInputClock (TSB_SC_TypeDef * SIOx,

uint32_t Clock)

Parameters:

SIOx is the specified SIO channel.

Clock is Selects input clock for prescaler as PhiT0/2 or PhiT0.

This parameter can be one of the following values:

SIO_CLOCK_T0_HALF: PhiT0/2

SIO_CLOCK_T0: PhiT0

Description:

This function will select the specified SIO channel by ***SIOx*** and specified the

input clock for prescaler by ***clock***

Return:

None

16.2.3.33 SIO_Enable

Enable the specified SIO channel.

Prototype:

void

SIO_Enable(TSB_SC_TypeDef* ***SIOx***)

Parameters:

SIOx is the specified SIO channel.

Description:

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

Return:

None

16.2.3.34 SIO_Disable

Disable the specified SIO channel.

Prototype:

void

SIO_Disable(TSB_SC_TypeDef* ***SIOx***)

Parameters:

SIOx is the specified SIO channel.

Description:

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

Return:

None

16.2.3.35 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.

InitStruct is the structure containing basic SIO configuration. (Refer to “Data Structure Description” for details).

Description:

This function will initialize and configure the specified SIO channel selected by ***SIOx***.

Return:

None

16.2.3.36 SIO_GetRxData

Get data received from the specified SIO channel.

Prototype:

Uint8_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

16.2.3.37 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

16.2.4 Data Structure Description

16.2.4.1 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.

16.2.4.2 SIO_InitTypeDef

Data Fields:

uint32_t

InputClkEdge Select the input clock edge, which can be set as:

- **SIO_SCLKS_TXDF_RXDR** Data in the transfer buffer is sent to TXDx pin one bit at a time on the falling edge of SCLKx, data from RXDx pin is received in the receive buffer one bit at a time on the rising edge of SCLKx.
- **SIO_SCLKS_TXDR_RXDF** Data in the transfer buffer is sent to TXDx pin one bit at a time on the rising edge of SCLKx, data from RXDx pin is received in the receive buffer one bit at a time on the falling edge of SCLKx.

uint32_t

TIDLE The status of TXDx pin after output of the last bit, which can be set as:

- **SIO_TIDLE_LOW** Set the status of TXDx pin keep a low level output.
- **SIO_TIDLE_HIGH** Set the status of TXDx pin keep a high level output.
- **SIO_TIDLE_LAST** Set the status of TXDx pin keep a last bit.

uint32_t

TXDEMP The status of TXDx pin when an under run error is occured in SCLK input mode, which can be set as:

- **SIO_TXDEMP_LOW** Set the status of TXDx pin is low level output.
- **SIO_TXDEMP_HIGH** Set the status of TXDx pin is high level output.

uint32_t

EHOLDTime The last bit hold time of TXDx pin in SCLK input mode, which can be set as:

- **SIO_EHOLD_FC_2** Set a last bit hold time is 2/fc.
- **SIO_EHOLD_FC_4** Set a last bit hold time is 4/fc.
- **SIO_EHOLD_FC_8** Set a last bit hold time is 8/fc.

- **SIO_EHOLD_FC_16** Set a last bit hold time is 16/fc.
- **SIO_EHOLD_FC_32** Set a last bit hold time is 32/fc.
- **SIO_EHOLD_FC_64** Set a last bit hold time is 64/fc.
- **SIO_EHOLD_FC_128** Set a last bit hold time is 128/fc.

uint32_t

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

- **SIO_SINT_TIME_NONE** Interval time is None.
- **SIO_SINT_TIME_SCLK_1** Interval time is 1xSCLK.
- **SIO_SINT_TIME_SCLK_2** Interval time is 2xSCLK.
- **SIO_SINT_TIME_SCLK_4** Interval time is 4xSCLK.
- **SIO_SINT_TIME_SCLK_8** Interval time is 8xSCLK.
- **SIO_SINT_TIME_SCLK_16** Interval time is 16xSCLK.
- **SIO_SINT_TIME_SCLK_32** Interval time is 32xSCLK.
- **SIO_SINT_TIME_SCLK_64** Interval time is 64xSCLK.

uint32_t

TransferMode Setting transfer mode, which can be set as:

- **SIO_TRANSFER_PROHIBIT** Transfer prohibit.
- **SIO_TRANSFER_HALFDPX_RX** Half duplex (Receive).
- **SIO_TRANSFER_HALFDPX_TX** Half duplex (Transmit).
- **SIO_TRANSFER_FULLDPX** Full duplex.

uint32_t

TransferDir Setting transfer mode, which can be set as:

- **SIO_LSB_FRIST** LSB first.
- **SIO_MSB_FRIST** MSB first.

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

DoubleBuffer Double Buffer mode, which can be set as:

- **SIO_WBUF_DISABLE** Double buffer disable.
- **SIO_WBUF_ENABLE** Double buffer enable.

uint32_t

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

- **SIO_BR_CLOCK_TS0** Select the input clock to baud rate generator is TS0.
- **SIO_BR_CLOCK_TS2** Select the input clock to baud rate generator is TS2.
- **SIO_BR_CLOCK_TS8** Select the input clock to baud rate generator is TS8.
- **SIO_BR_CLOCK_TS32** Select the input clock to baud rate generator is TS32.

uint32_t

Divider Division ratio "N", which can be set as:

- **SIO_BR_DIVIDER_16** Division ratio is 16.
- **SIO_BR_DIVIDER_1** Division ratio is 1.
- **SIO_BR_DIVIDER_2** Division ratio is 2.
- **SIO_BR_DIVIDER_3** Division ratio is 3.

- **SIO_BR_DIVIDER_4** Division ratio is 4.
- **SIO_BR_DIVIDER_5** Division ratio is 5.
- **SIO_BR_DIVIDER_6** Division ratio is 6.
- **SIO_BR_DIVIDER_7** Division ratio is 7.
- **SIO_BR_DIVIDER_8** Division ratio is 8.
- **SIO_BR_DIVIDER_9** Division ratio is 9.
- **SIO_BR_DIVIDER_10** Division ratio is 10.
- **SIO_BR_DIVIDER_11** Division ratio is 11.
- **SIO_BR_DIVIDER_12** Division ratio is 12.
- **SIO_BR_DIVIDER_13** Division ratio is 13.
- **SIO_BR_DIVIDER_14** Division ratio is 14.
- **SIO_BR_DIVIDER_15** Division ratio is 15.

17. USBD

17.1 Overview

TOSHIBA TMPM067 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 5 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

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\TX00_Periph_Driver\src\usbd_hw.c, with \Libraries\TX00_Periph_Driver\inc\usbd_hw.h containing the macros, data types, structures and API definitions for use by applications.

17.2 API Functions

17.2.1 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***) ;

17.2.2 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()

17.2.3 Function Documentation

17.2.3.1 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.

17.2.3.2 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

17.2.3.3 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_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

17.2.3.4 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.

17.2.3.5 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

17.2.3.6 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.

17.2.3.7 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 4) :** EPx_MaxPacketSize register.

- **UDC2_EPxSTATUS(x = 0 to 4) :** EPx status register.

- **UDC2_EPxDATASIZE(x = 0 to 4) :** EPx datasize register.

- **UDC2_EPxFIFO(x = 0 to 4) :** 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 gotten by calling this function with parameter below one by one:

UDC2_BREQ_BMREQTYPE,

UDC2_WVALUE,

UDC2_WINDEX,

UDC2_WLENGTH

Return:

None

17.2.3.8 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 4) :** EPx Max Packet Size register.
- **UDC2_EPxSTATUS (x = 0 to 4) :** EPx status register.
- **UDC2_EPxFIFO (x = 1 to 4) :** 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

17.2.3.9 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.

17.2.3.10 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

17.2.3.11 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.

17.2.3.12 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

17.2.3.13 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

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 4) :**

- **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

17.2.3.14 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.

17.2.3.15 USBD_GetEPxStatus

Get current EPx status (x = 1 to 4)

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

Description:

This function will get status for current endpoint x (x = 1 to 4).

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 4).

Refer to data structure of USBD_EPxStatus for details.

17.2.3.16 USBD_ConfigEPx

Configure EPx (x = 1 to 4).

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

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 4).

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

17.2.4 Data Structure Description

17.2.4.1 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

17.2.4.2 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 : 3

Place a 3bits gap to none used area. Read as undefined. Write as zero

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

17.2.4.3 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

17.2.4.4 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

17.2.4.5 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

Always be 0 because it isn't supported in M067 (Read Only).

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

It isn't supported in M067

uint32_t

Reserved*: *24

Place a 24bits gap to none used area. Read as undefined. Write as zero

17.2.4.6 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

uint32_t

Reserved3*: *16

Place a 16bits gap to none used area. Read as undefined. Write as zero

17.2.4.7 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, must be set to ‘1’ as IN only.

for EP2,4 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

uint32_t

Reserved2*:** ***16

Place a 16bits gap to none used area. Read as undefined. Write as zero

18. WDT

18.1 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\TX00_Periph_Driver\src\tmpm06x_wdt.c, with \Libraries/TX00_Periph_Driver\inc\tmpm06x_wdt.h containing the API definitions for use by applications.

18.2 API Functions

18.2.1 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)

18.2.2 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().

18.2.3 Function Documentation

18.2.3.1 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

18.2.3.2 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**.

***Note:**

If CPU needs to enter the IDLE mode, this function must be called with appropriate parameter.

Return:

None

18.2.3.3 WDT_SetOverflowOutput

Set WDT to generate NMI interrupt or 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 overflows.
- **WDT_WDOUT**: Set WDT to generate reset when counter overflows.

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

18.2.3.4 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 “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

18.2.3.5 WDT_Enable

Enable the WDT function.

Prototype:

void

WDT_Enable(void)

Parameters:

None

Description:

This function will enable WDT.

Return:

None

18.2.3.6 WDT_Disable

Disable the WDT function.

Prototype:

void

WDT_Disable(void)

Parameters:

None

Description:

This function will disable WDT.

Return:

None

18.2.3.7 WDT_WriteClearCode

Write the clear code.

Prototype:

void

WDT_WriteClearCode (void)

Parameters:

None

Description:

This function will clear the WDT counter.

Return:

None

18.2.4 Data Structure Description

18.2.4.1 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.