

TOSHIBA

TOSHIBA TX04 Peripheral Driver User Guide (TMPM440)

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M440UG-01E

RESTRICTIONS ON PRODUCT USE

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

Index

1. Introduction	1
2. Organization of TOSHIBA TX04 Peripheral Driver	1
3. ADC	3
3.1 Overview	3
3.2 API Functions.....	3
3.2.1 Function List	3
3.2.2 Detailed Description.....	4
3.2.3 Function Documentation	4
3.2.4 Data Structure Description.....	21
4. CG	24
4.1 Overview	24
4.2 API Functions.....	25
4.2.1 Function List	25
4.2.2 Detailed Description.....	26
4.2.3 Function Documentation	26
4.2.4 Data Structure Description.....	54
5. DAC	56
5.1 Overview	56
5.2 API Functions.....	56
5.2.1 Function List	56
5.2.2 Detailed Description.....	56
5.2.3 Function Documentation	56
5.2.4 Data Structure Description.....	58
6. DMAC	59
6.1 Overview	59
6.2 API Functions.....	59
6.2.1 Function List	59
6.2.2 Detailed Description.....	60
6.2.3 Function Documentation	60
6.2.4 Data Structure Description.....	75
7. EPHC.....	81
7.1 Overview	81
7.2 API Functions.....	81
7.2.1 Function List	81
7.2.2 Detailed Description.....	82
7.2.3 Function Documentation	82
7.2.4 Data Structure Description.....	93
8. ESIO	97
8.1 Overview	97
8.2 API Functions.....	97
8.2.1 Function List	97

8.2.2 Detailed Description.....	98
8.2.3 Function Documentation.....	98
8.2.4 Data Structure Description.....	113
9. EXB	116
9.1 Overview	116
9.2 API Functions.....	116
9.2.1 Function List	116
9.2.2 Detailed Description.....	117
9.2.3 Function Documentation.....	117
9.2.4 Data Structure Description.....	120
10. FC	123
10.1 Overview	123
10.2 API Functions.....	123
10.2.1 Function List	123
10.2.2 Detailed Description.....	123
10.2.3 Function Documentation.....	124
10.2.4 Data Structure Description.....	129
11. FUART	130
11.1 Overview	130
11.2 API Functions.....	130
11.2.1 Function List	130
11.2.2 Detailed Description.....	131
11.2.3 Function Documentation.....	131
11.2.4 Data Structure Description.....	143
12. GPIO	145
12.1 Overview	145
12.2 API Functions.....	145
12.2.1 Function List	145
12.2.2 Detailed Description.....	146
12.2.3 Function Documentation.....	146
12.2.4 Data Structure Description.....	168
13. KSCAN	170
13.1 Overview	170
13.2 API Functions.....	170
13.2.1 Function List	170
13.2.2 Detailed Description.....	170
13.2.3 Function Documentation.....	171
13.2.4 Data Structure Description.....	179
14. KWUP.....	180
14.1 Overview	180
14.2 API Functions.....	180
14.2.1 Function List	180
14.2.2 Detailed Description.....	180

14.2.3	Function Documentation	181
14.2.4	Data Structure Description	182
15.	PHC	185
15.1	Overview	185
15.2	API Functions	185
15.2.1	Function List	185
15.2.2	Detailed Description	186
15.2.3	Function Documentation	186
15.2.4	Data Structure Description	194
16.	RTC	196
16.1	Overview	196
16.2	API Functions	196
16.2.1	Function List	196
16.2.2	Detailed Description	197
16.2.3	Function Documentation	198
16.2.4	Data Structure Description	218
17.	SBI	220
17.1	Overview	220
17.2	API Functions	220
17.2.1	Function List	220
17.2.2	Detailed Description	221
17.2.3	Function Documentation	221
17.2.4	Data Structure Description	228
18.	TMRB	231
18.1	Overview	231
18.2	API Functions	231
18.2.1	Function List	231
18.2.2	Detailed Description	232
18.2.3	Function Documentation	232
18.2.4	Data Structure Description	247
19.	TMRC	250
19.1	Overview	250
19.2	API Functions	250
19.2.1	Function List	250
19.2.2	Detailed Description	250
19.2.3	Function Documentation	251
19.2.4	Data Structure Description	258
20.	TMRD	260
20.1	Overview	260
20.2	API Functions	261
20.2.1	Function List	261
20.2.2	Detailed Description	261
20.2.3	Function Documentation	262

	20.2.4 Data Structure Description.....	276
21.	SIO/UART	279
21.1	Overview	279
21.2	API Functions.....	279
21.2.1	Function List	279
21.2.2	Detailed Description.....	280
21.2.3	Function Documentation	281
21.2.4	Data Structure Description.....	300
22.	WDT	304
22.1	Overview	304
22.2	API Functions.....	304
22.2.1	Function List	304
22.2.2	Detailed Description.....	304
22.2.3	Function Documentation	305
22.2.4	Data Structure Description.....	308
23.	PSC.....	309
23.1	Overview	309
23.2	API Functions.....	309
23.2.1	Function List	309
23.2.2	Detailed Description.....	310
23.2.3	Function Documentation	311
23.2.4	Data Structure Description.....	335

1. Introduction

TOSHIBA TX04 Peripheral Driver is a set of drivers for all peripherals found on the TOSHIBA TX04 series micro controllers. TPM440 Peripheral Driver is an important part of TOSHIBA TX04 Peripheral Driver, which are designed for TPM440 MCU.

TOSHIBA TX04 Peripheral Driver contains a collection of macros, data types, and structures for each peripheral.

The design goals of TOSHIBA TPM440 Peripheral Driver:

- Completely written in C except the start-up routine and where not possible
- Cover all the peripherals on MCU

2. Organization of TOSHIBA TX04 Peripheral Driver

/Libraries

This folder contains all CMSIS files and TPM440 Peripheral Drivers.

/Libraries/ TX04_CMSIS

This folder contains the TPM440 CMSIS files: device peripheral access layer and core peripheral access layer.

/Libraries/TX04_Periph_Driver

This folder contains all the source code of the drivers, the core of TOSHIBA TPM440 Peripheral Driver.

/Libraries/TX04_Periph_Driver/inc

This folder contains all the header files of TPM440 Peripheral Drivers for each peripheral.

/Libraries/TX04_Periph_Driver/src

This folder contains all the source files of TPM440 Peripheral Drivers for each peripheral.

/Project

This folder contains template project and examples for using TPM440 Peripheral Driver.

/Project/Template

This folder contains template project of TOSHIBA TPM440 Peripheral Driver.

/Project/Examples

This folder contains a set of examples for using TPM440 Peripheral Driver

/Utilities/TMPM440-EVAL

This folder contains the configuration and driver files for hardware resources (e.g. led, key) on Toshiba TMPM440-EVAL board.

3. ADC

3.1 Overview

A 12-bit, sequential-conversion analog/digital converter (A/D converter) is built into TMPM440. This A/D converter is equipped with 8 analog input channels in unit A and unit B, and 4 analog input channels in unit C.

These analog input channels are also used as input ports.

The 12-bit AD converter has the following features:

1. Start normal AD conversion and top-priority AD conversion by software activation, 16-bit time (TMRB) activation or hardware activation with an external trigger input.
2. AD conversion in 4 different modes:
 - Fixed-channel single conversion mode
 - Channel scan single conversion mode
 - Fixed-channel repeat conversion mode
 - Channel scan repeat conversion mode
3. Normal / Top-priority AD conversion completion interrupt
4. Normal / Top-priority AD conversion completion / busy flag
5. AD monitor function

When the AD monitor function is enabled, an interrupt is generated if any comparison result is matched.

The ADC API provides a set of functions for using the TMPM440 ADC modules. It includes ADC channel set, mode set, monitor function set, interrupt set, ADC status read, ADC result value read and so on.

This driver is contained in TX04_Periph_Driver\src\tmpm440_adc.c (*), with TX04_Periph_Driver\incl\tmpm440_adc.h (*) containing the API definitions for use by applications.

3.2 API Functions

3.2.1 Function List

- ◆ void ADC_SWReset(TSB_AD_TypeDef * ADx);
- ◆ void ADC_SetClk(TSB_AD_TypeDef * ADx, uint32_t Sample_HoldTime, uint32_t Prescaler_Output);

- ◆ void ADC_Start(TSB_AD_TypeDef * ADx);
- ◆ void ADC_SetScanMode(TSB_AD_TypeDef * ADx, FunctionalState NewState);
- ◆ void ADC_SetRepeatMode(TSB_AD_TypeDef * ADx, FunctionalState NewState);
- ◆ void ADC_SetINTMode(TSB_AD_TypeDef * ADx, uint8_t INTMode);
- ◆ void ADC_SetInputChannel(TSB_AD_TypeDef * ADx, uint8_t InputChannel);
- ◆ void ADC_SetScanChannel(TSB_AD_TypeDef * ADx, uint8_t StartChannel, uint8_t Range);
- ◆ void ADC_SetVrefCut(TSB_AD_TypeDef * ADx, uint8_t VrefCtrl);
- ◆ void ADC_SetIdleMode(TSB_AD_TypeDef * ADx, FunctionalState NewState);
- ◆ void ADC_SetVref(TSB_AD_TypeDef * ADx, FunctionalState NewState);
- ◆ void ADC_SetInputChannelTop(TSB_AD_TypeDef * ADx, uint8_t TopInputChannel);
- ◆ void ADC_StartTopConvert(TSB_AD_TypeDef * ADx);
- ◆ void ADC_SetMonitor(TSB_AD_TypeDef * ADx, ADC_CMPCRx ADCMPx, FunctionalState NewState);
- ◆ void ADC_ConfigMonitor(TSB_AD_TypeDef * ADx, ADC_CMPCRx ADCMPx, ADC_MonitorTypeDef * Monitor);
- ◆ void ADC_SetHWTrg(TSB_AD_TypeDef * ADx, uint8_t HwSource, FunctionalState NewState);
- ◆ void ADC_SetHWTrgTop(TSB_AD_TypeDef * ADx, uint8_t HwSource, FunctionalState NewState);
- ◆ ADC_State ADC_GetConvertState(TSB_AD_TypeDef * ADx);
- ◆ ADC_Result ADC_GetConvertResult(TSB_AD_TypeDef * ADx, uint8_t ADREGx);
- ◆ void ADC_SetDMAReq(TSB_AD_TypeDef * ADx, uint8_t **DMAReq**, FunctionalState **NewState**);

3.2.2 Detailed Description

Functions listed above can be divided into four parts:

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

3.2.3 Function Documentation

3.2.3.1 ADC_SWReset

Software reset ADC.

Prototype:

void

ADC_SWReset(TSB_AD_TypeDef * **ADx**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

Description:

This function will software reset ADC.

Notes:

A software reset initializes all the registers except for ADxCLK<ADCLK>.

Initialization takes 3μs in case of the software reset.

Return:

None

3.2.3.2 ADC_SetClk

Set ADC sample hold time and prescaler output.

Prototype:

void

ADC_SetClk(TSB_AD_TypeDef * **ADx**,
 uint32_t **Sample_HoldTime**,
 uint32_t **Prescaler_Output**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

Sample_HoldTime: Select ADC sample hold time.

This parameter can be one of the following values:

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

- **ADC_CONVERSION_CLK_160:** 160x <ADCLK>
- **ADC_CONVERSION_CLK_320:** 320x <ADCLK>

Prescaler_Output: Select ADC prescaler output(ADCLK).

This parameter can be one of the following values:

- **ADC_FC_DIVIDE_LEVEL_2:** $f_c / 2$
- **ADC_FC_DIVIDE_LEVEL_4:** $f_c / 4$
- **ADC_FC_DIVIDE_LEVEL_8:** $f_c / 8$
- **ADC_FC_DIVIDE_LEVEL_16:** $f_c / 16$

Description:

This function will set ADC unit by **ADx**, ADC sample hold time by **Sample_HoldTime** and prescaler output by **Prescaler_Output**.

Notes:

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

Return:

None

3.2.3.3 ADC_Start

Start AD conversion.

Prototype:

void

ADC_Start(TSB_AD_TypeDef * **ADx**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

Description:

This function will start normal AD conversion.

Notes:

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

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

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

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

Return:

None

3.2.3.4 ADC_SetScanMode

Enable or disable ADC scan mode.

Prototype:

void

ADC_SetScanMode(TSB_AD_TypeDef * **ADx**, FunctionalState **NewState**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

NewState: Specify ADC scan mode state

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable or disable ADC scan mode.

Return:

None

3.2.3.5 ADC_SetRepeatMode

Enable or disable ADC repeat mode.

Prototype:

void

ADC_SetRepeatMode(TSB_AD_TypeDef * **ADx**, FunctionalState **NewState**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA**: unit A
- **TSB_ADB**: unit B
- **TSB_ADC**: unit C

NewState: Specify ADC repeat mode state

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will enable or disable ADC repeat mode.

Return:

None

3.2.3.6 ADC_SetINTMode

Set ADC interrupt mode in fixed channel repeat conversion mode.

Prototype:

void

ADC_SetINTMode(TSB_AD_TypeDef * **ADx**, uint8_t **INTMode**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA**: unit A
- **TSB_ADB**: unit B
- **TSB_ADC**: unit C

INTMode: Specify AD conversion interrupt mode.

The parameter can be one of the following values:

- **ADC_INT_SINGLE**: Generate in interrupt once every single conversion.
- **ADC_INT_CONVERSION_2**: Generate interrupt once every 2 conversions.
- **ADC_INT_CONVERSION_3**: Generate interrupt once every 3 conversions.
- **ADC_INT_CONVERSION_4**: Generate interrupt once every 4 conversions.

- **ADC_INT_CONVERSION_5:** Generate interrupt once every 5 conversions.
- **ADC_INT_CONVERSION_6:** Generate interrupt once every 6 conversions.
- **ADC_INT_CONVERSION_7:** Generate interrupt once every 7 conversions.
- **ADC_INT_CONVERSION_8:** Generate interrupt once every 8 conversions.

Description:

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

Notes:

This function is valid only in fixed channel repeat conversion mode.

Examples for setting fixed channel repeat conversion mode:

1. **ADC_SetScanMode(DISABLE).**
2. **ADC_SetRepeatMode(ENABLE).**

Return:

None

3.2.3.7 ADC_SetInputChannel

Set ADC input channel.

Prototype:

void

ADC_SetInputChannel(TSB_AD_TypeDef * **ADx**, uint8_t **InputChannel**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

InputChannel: Analog input channel.

This parameter can be one of the following values in unit A or unit B:

**ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07.**

This parameter can be one of the following values in unit C:

ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03.

Description:

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

Notes:

Only one channel can be selected as normal conversion input each time.

Return:

None

3.2.3.8 ADC_SetScanChannel

Set ADC scan channel.

Prototype:

void

```
ADC_SetScanChannel(TSB_AD_TypeDef * ADx,  
                  uint8_t StartChannel,  
                  uint8_t Range)
```

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

StartChannel: Specify the start channel to be scanned.

This parameter can be one of the following values in unit A or unit B:

ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07.

This parameter can be one of the following values in unit C:

ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03.

Range: Specify the range of assignable channel scan value.

This parameter can be **1** to **8** in unit A or unit B and be **1** to **4** in unit C.

Description:

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

Notes:

Valid channel scan setting values are shown as follows:

StartChannel	Range
ADC_AN_00	1 ~ 8
ADC_AN_01	1 ~ 7
ADC_AN_02	1 ~ 6
ADC_AN_03	1 ~ 5
ADC_AN_04	1 ~ 4

ADC_AN_05	1 ~ 3
ADC_AN_06	1 ~ 2
ADC_AN_07	1 ~ 1

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

Return:

None

3.2.3.9 ADC_SetVrefCut

Control AVREFH-AVREFL current.

Prototype:

void

ADC_SetVrefCut(TSB_AD_TypeDef * **ADx**, uint8_t **VrefCtrl**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

VrefCtrl: Specify how to apply AVREFH-AVREFL current.

This parameter can be one of the following values:

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

Description:

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

Return:

None

3.2.3.10 ADC_SetIdleMode

Set ADC operation in IDLE mode.

Prototype:

void

ADC_SetIdleMode(TSB_AD_TypeDef * **ADx**, FunctionalState **NewState**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

NewState: Specify ADC operation state in IDLE mode.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will enable or disable ADC operation state in system IDLE mode.

This function is necessary to be called before system enter IDLE mode.

Return:

None

3.2.3.11 ADC_SetVref

Set ADC Vref application control on or off.

Prototype:

void

ADC_SetVref(TSB_AD_TypeDef * **ADx**, FunctionalState **NewState**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

NewState: Specify AD conversion Vref application control.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

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

Notes:

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

Return:

None

3.2.3.12 ADC_SetInputChannelTop

Select ADC top-priority conversion analog input channel.

Prototype:

void

ADC_SetInputChannelTop(TSB_AD_TypeDef * **ADx**, uint8_t **TopInputChannel**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA**: unit A
- **TSB_ADB**: unit B
- **TSB_ADC**: unit C

TopInputChannel: Analog input channel for top-priority conversion.

This parameter can be one of the following values in unit A or unit B:

**ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07.**

This parameter can be one of the following values in unit C:

ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03.

Description:

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

Notes:

Only one channel can be selected as Top-priority conversion input each time.

Return:

None

3.2.3.13 ADC_StartTopConvert

Start top-priority AD conversion.

Prototype:

void

ADC_StartTopConvert(TSB_AD_TypeDef * **ADx**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

Description:

This function will start top-priority AD conversion.

Notes:

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

Return:

None

3.2.3.14 ADC_SetMonitor

Enable or disable the specified ADC monitor module.

Prototype:

void

ADC_SetMonitor(TSB_AD_TypeDef * **ADx**,
ADC_CMPCRx **ADCMPx**,
FunctionalState **NewState**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

ADCMPx: Select which compare control register will be used.

The parameter can be one of the following values:

- **ADC_CMPCR_0:** ADCMPCR0
- **ADC_CMPCR_1:** ADCMPCR1

NewState: Specify ADC monitor function state.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This device has 2 AD monitor modules which are controlled by 2 compare control registers.

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

Return:

None

3.2.3.15 ADC_ConfigMonitor

Configure the specified ADC monitor module.

Prototype:

```
void  
ADC_ConfigMonitor(TSB_AD_TypeDef * ADx,  
                  ADC_CMPCRx ADCMPx,  
                  ADC_MonitorTypeDef * Monitor)
```

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

ADCMPx: Select which compare control register will be used.

The parameter can be one of the following values:

- **ADC_CMPCR_0:** ADCMPCR0
- **ADC_CMPCR_1:** ADCMPCR1

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

Description:

This device has 2 AD monitor modules which are controlled by 2 compare control registers.

This function will specify compare control register by **ADCMPx** setting and specify ADC monitor configuration **Monitor** setting.

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

Return:

None

3.2.3.16 ADC_SetHWTrg

Set hardware trigger for normal AD conversion.

Prototype:

void

```
ADC_SetHWTrg(TSB_AD_TypeDef * ADx,  
             uint8_t HwSource,  
             FunctionalState NewState)
```

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

HwSource: Hardware source for activating normal AD conversion.

This parameter can be one of the following values:

- **ADC_EXT_TRG:**

ADC_EXT_TRG (ADTRGx and ADTRGSNC).

It can be started AD conversion of Unit A and Unit B at same time by ADGGSNC (PH3) pin.

ADGGSNC (PH3) pin is only used in Unit A and Unit B.

- **ADC_MATCH_TMRB_NORMAL:**

The Timer is TB17RG0 for normal AD conversion while in Unit A.

The Timer is TB18RG0 for normal AD conversion while in Unit B.

The Timer is TB19RG0 for normal AD conversion while in Unit C.

NewState: Specify state of hardware source for activating normal AD conversion.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

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

This function also has relation with TB5 setting.

Notes:

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

Return:

None

3.2.3.17 ADC_SetHWTrgTop

Set hardware trigger for top-priority AD conversion.

Prototype:

void

```
ADC_SetHWTrgTop(TSB_AD_TypeDef * ADx,  
                uint8_t HwSource,  
                FunctionalState NewState)
```

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

HwSource: Hardware source for activating top-priority AD conversion.

This parameter can be one of the following values:

- **ADC_EXT_TRG:**

ADC_EXT_TRG include 2 pins: ADTRGx and ADTRGSNC.

It can be started AD conversion of Unit A and UnitB at same time by ADGGSNC (PH3) pin.

ADGGSNC (PH3) pin is only used in Unit A and Unit B.

- **ADC_MATCH_TMRB_TOP:**

The Timer is TB14RG0 for top-priority AD conversion while in Unit A.

The Timer is TB15RG0 for top-priority AD conversion while in Unit B.

The Timer is TB16RG0 for top-priority AD conversion while in Unit C.

NewState: Specify state of hardware source for activating top-priority AD conversion.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

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

Notes:

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

Return:

None

3.2.3.18 ADC_GetConvertState

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

Prototype:

WorkState

ADC_GetConvertState(TSB_AD_TypeDef * **ADx**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA:** unit A
- **TSB_ADB:** unit B
- **TSB_ADC:** unit C

Description:

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

Return:

AD conversion state:

NormalComplete (Bit 1) means normal AD conversion is complete.

TopComplete (Bit 3) means top-priority AD conversion is complete.

3.2.3.19 ADC_GetConvertResult

Read AD conversion result.

Prototype:

ADC_Result

ADC_GetConvertResult(TSB_AD_TypeDef * **ADx**,

uint8_t **ADREGx**)

Parameters:

ADx: Select ADC unit.

This parameter can be one of the following values:

- **TSB_ADA**: unit A
- **TSB_ADB**: unit B
- **TSB_ADC**: unit C

ADREGx: Select ADC result register.

The parameter can be one of the following values:

ADC_REG_00, **ADC_REG_01**, **ADC_REG_02**, **ADC_REG_03**,
ADC_REG_04, **ADC_REG_05**, **ADC_REG_06**, **ADC_REG_07**,
ADC_REG_SP.

Description:

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

Notes:

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

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

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

Fixed-channel single mode	
Channel	Storage register
ADC_AN_00	ADC_REG_00
ADC_AN_01	ADC_REG_01
ADC_AN_02	ADC_REG_02
ADC_AN_03	ADC_REG_03
ADC_AN_04	ADC_REG_04
ADC_AN_05	ADC_REG_05
ADC_AN_06	ADC_REG_06
ADC_AN_07	ADC_REG_07

Fixed-channel repeat mode	
Interrupt mode	Storage register

Interrupt by each time ADC	ADC_REG_00
Interrupt by each time 2 ADC	ADC_REG_00 to ADC_REG_01
Interrupt by each time 3 ADC	ADC_REG_00 to ADC_REG_02
Interrupt by each time 4 ADC	ADC_REG_00 to ADC_REG_03
Interrupt by each time 5 ADC	ADC_REG_00 to ADC_REG_04
Interrupt by each time 6 ADC	ADC_REG_00 to ADC_REG_05
Interrupt by each time 7 ADC	ADC_REG_00 to ADC_REG_06
Interrupt by each time 8 ADC	ADC_REG_00 to ADC_REG_07

Channel scan single mode / repeat mode		
Start channel	Scan channel range	Storage register
ADC_AN_00	15 channels	ADC_REG_00 to ADC_REG_07
ADC_AN_01	14 channels	ADC_REG_01 to ADC_REG_07
ADC_AN_02	13 channels	ADC_REG_02 to ADC_REG_07
ADC_AN_03	12 channels	ADC_REG_03 to ADC_REG_07
ADC_AN_04	11 channels	ADC_REG_04 to ADC_REG_07
ADC_AN_05	10 channels	ADC_REG_05 to ADC_REG_07
ADC_AN_06	9 channels	ADC_REG_06 to ADC_REG_07
ADC_AN_07	8 channels	ADC_REG_07 to ADC_REG_07

The ADC mode setting, please refer to relate APIs.

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

Return:

AD conversion result:

ADR (Bit 0 to Bit 11) means AD result value.

ADRF (Bit 12) means AD result has been stored.

ADOVRF (Bit 13) means new AD result is stored before the old one is read.

ADRF_MR (Bit 16) the mirror register of **ADRF**.

ADOVRF_MR (Bit 17) the mirror register of **ADOVRF**.

ADR_MR (Bit 20 to Bit 31) the mirror register of **ADR**.

3.2.3.20 ADC_SetDMAReq

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

Prototype:

void

ADC_SetDMAReq(TSB_AD_TypeDef * **ADx**,uint8_t **DMAReq**,
FunctionalState **NewState**)

Parameters:

DMAReq: Specify AD conversion DMA request type.

The parameter can be one of the following values:

- **ADC_DMA_REQ_NORMAL:** normal AD conversion.
- **ADC_DMA_REQ_TOP:** top-priority AD conversion.

NewState: Specify AD conversion DMA activation factor.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

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

Return:

None

3.2.4 Data Structure Description

3.2.4.1 ADC_MonitorTypeDef

Data Fields for this structure:

uint8_t

CmpChannel Select which ADC Result Register to be used,
which can be:

ADC_AN_00, ADC_AN_01, ADC_AN_02, ADC_AN_03,
ADC_AN_04, ADC_AN_05, ADC_AN_06, ADC_AN_07,

uint32_t

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

ADC_CmpCondition

Condition Condition to compare AINx with ADCMPn (x= 0 to 7, n = 0 to 1),
which can be:

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

ADC_CmpCntMode

CntMode Mode to compare AINx with ADCMPn (x = 0 to 07, n = 0 to 1),
which can be:

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

uint32_t

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

3.2.4.2 ADC_State

Data Fields for this structure:

uint32_t

All specifies AD conversion state.

Bit Fields:

uint32_t

Reserved0 (Bit 0) reserved.

uint32_t

NormalComplete (Bit 1) means normal AD conversion is complete.

uint32_t

Reserved1 (Bit 2) reserved.

uint32_t

TopComplete (Bit 3) means top-priority AD conversion is complete.

uint32_t

Reserved2 (Bit 4 to Bit 31) reserved.

3.2.4.3 ADC_Result

Data Fields for this structure:

uint32_t

All specifies AD conversion result.

Bit Fields:

uint32_t

ADResult (Bit 0 to Bit 11) means AD result value.

uint32_t

Stored (Bit 12) means AD result has been stored.

uint32_t

OverRun (Bit 13) means new AD result is stored
before the old one is read.

uint32_t

Reserved (Bit 14 to Bit 15) reserved.

uint32_t

Stored_MR (Bit 16) uint32_t	means the mirror of . Stored .
OverRun_MR (Bit 17) uint32_t	means the mirror of . OverRun .
Reserved (Bit 18 to Bit 19) uint32_t	reserved.
ADResult_MR (Bit 20 to Bit 31)	means the mirror of ADResult .

4. CG

4.1 Overview

The CG API provides a set of functions for using the TPM440 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 TX04_Periph_Driver\src\tpm440_cg.c, with

TX04_Periph_Driver\inc\tpm440_cg.h containing the API definitions for use by applications.

The following symbols *fosc*, *fppll*, *fc*, *fgear*, *fsys*, *fperiph*, $\Phi T0$ are used for kinds of clock in CG. Please refer to the clock system diagram in section “Clock Block Diagram” of the datasheet for their meaning.

fosc : A clock generated in the internal oscillation circuit and input from X1 and X2 pins

fPLL : A clock multiplied by PLL

fc : A clock selected by CGPLLSEL<PLL0SEL> (high-speed clock)

fgear : A clock selected by CGSYSCR<GEAR[2:0]>

fsys : A clock identical with fgear

fperiph : A clock selected by CGSYSCR<FPSEL>

$\Phi T0$: A clock selected by CGSYSCR<PRCK[2:0]> (prescaler clock)

4.2 API Functions

4.2.1 Function List

- ◆ void CG_SetFgearLevel(CG_DivideLevel **DivideFgearFromFc**)
- ◆ CG_DivideLevel CG_GetFgearLevel(void)
- ◆ void CG_SetPhiT0Src(CG_PhiT0Src **PhiT0Src**)
- ◆ CG_PhiT0Src CG_GetPhiT0Src(void)
- ◆ Result CG_SetPhiT0Level(CG_DivideLevel **DividePhiT0FromFc**)
- ◆ CG_DivideLevel CG_GetPhiT0Level(void)
- ◆ void CG_SetSCOUTSrc(CG_SCOUTSrc **Source**)
- ◆ CG_SCOUTSrc CG_GetSCOUTSrc(void)
- ◆ void CG_SetWarmUpTime(CG_WarmUpSrc **Source**, uint16_t **Time**)
- ◆ void CG_StartWarmUp(void)
- ◆ WorkState CG_GetWarmUpState(void)
- ◆ Result CG_SetFPLLValue(CG_FpllValue **NewValue**)
- ◆ Result CG_SetPLL(FunctionalState **NewState**)
- ◆ FunctionalState CG_GetPLLState(void)
- ◆ Result CG_SetFPLLForADCValue(uint32_t **NewValue**)
- ◆ Result CG_SetFosc(CG_FoscSrc **Source**, FunctionalState **NewState**)
- ◆ void CG_SetFadcSrc(CG_FadcSrc **FadcSrc**)
- ◆ void CG_SetPLL1ForADC(FunctionalState **NewState**)
- ◆ void CG_SetFoscSrc(CG_FoscSrc **Source**)
- ◆ CG_FoscSrc CG_GetFoscSrc(void)
- ◆ FunctionalState CG_GetFoscState(CG_FoscSrc **Source**)
- ◆ Result CG_SetFs(FunctionalState **NewState**)
- ◆ FunctionalState CG_GetFsState(void)
- ◆ void CG_SetSTBYMode(CG_STBYMode **Mode**)
- ◆ CG_STBYMode CG_GetSTBYMode(void)
- ◆ void CG_SetPinStateInStop1Mode(FunctionalState **NewState**)
- ◆ FunctionalState CG_GetPinStateInStop1Mode(void)
- ◆ void CG_SetPortKeepInStop2Mode(FunctionalState **NewState**)
- ◆ FunctionalState CG_GetPortKeepInStop2Mode(void)
- ◆ Result CG_SetFcSrc(CG_FcSrc **Source**)
- ◆ CG_FcSrc CG_GetFcSrc(void)
- ◆ void CG_SetFtmrdSrc(CG_TmrdUnit **TmrdUnit**, CG_FtmrdSrc **FtmrdSrc**)
- ◆ CG_FtmrdSrc CG_GetFtmrdSrc(CG_TmrdUnit **TmrdUnit**)
- ◆ void CG_SetTMRDClk(CG_TmrdUnit **TmrdUnit**, FunctionalState **NewState**)
- ◆ FunctionalState CG_GetTMRDClkState(CG_TmrdUnit **TmrdUnit**)
- ◆ void CG_SetProtectCtrl(FunctionalState **NewState**)
- ◆ void CG_SetSTBYReleaseINTSrc(CG_INTSrc **INTSource**,

CG_INTActiveState **ActiveState**,

FunctionalState **NewState**)

- ◆ CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc **INTSource**)
- ◆ void CG_ClearINTRReq(CG_INTSrc **INTSource**)
- ◆ CG_ResetFlag CG_GetResetFlag(void)
- ◆ void CG_SetPeriphClkSupply(uint32_t **Periph**)
- ◆ void CG_SetFclkPeriphA(uint32_t Periph, FunctionalState **NewState**)
- ◆ void CG_SetFclkPeriphB(uint32_t Periph, FunctionalState **NewState**)
- ◆ void CG_SetFcPeriphA(uint32_t Periph, FunctionalState **NewState**)
- ◆ void CG_SetFcPeriphB(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_SetPhiT0Src(), CG_GetPhiT0Src(),
 CG_SetPhiT0Level(), CG_GetPhiT0Level(), CG_SetSCOUTSrc(), CG_GetSCOUTSrc(),
 CG_SetWarmUpTime(), CG_StartWarmUp(),
 CG_GetWarmUpState(), CG_SetFPLLValue(), CG_SetPLL(), CG_GetPLLState(),
 CG_SetFosc(), CG_SetFoscSrc(), CG_GetFoscSrc(),
 CG_GetFoscState(), CG_SetFcSrc(), CG_GetFcSrc(), CG_SetProtectCtrl(),
 CG_SetFclkPeriphA(), CG_SetFclkPeriphB(), CG_SetFcPeriphA(), CG_SetFcPeriphB(), CG_
 SetFPLLForADCValue(), CG_SetPLL1ForADC(), CG_SetFadcSrc(), CG_SetFs(), CG_GetFs
 State().
- 2) The 2nd group of APIs handle settings of standby modes:
 CG_SetSTBYMode(), CG_GetSTBYMode(),
 CG_SetPinStateInStop1Mode(), CG_GetPinStateInStop1Mode(),
 CG_SetPortKeepInStop2Mode(), CG_GetPortKeepInStop2Mode().
- 3) The 3rd group of APIs handle settings of interrupts:
 CG_SetSTBYReleaseINTSrc(), CG_GetSTBYReleaseINTState(), CG_ClearINTRReq(),
 CG_GetResetFlag().
- 4) The other APIs control clock supply for peripherals:
 CG_SetPeriphClkSupply(), CG_SetFtmrdSrc(), CG_GetFtmrdSrc(), CG_SetTMRDClk(),
 CG_GetTMRDClkState().

4.2.3 Function Documentation

4.2.3.1 CG_SetFgearLevel

Set the dividing level between clock fgear and fc.

Prototype:

void

CG_SetFgearLevel(CG_DivideLevel ***DivideFgearFromFc***)

Parameters:

DivideFgearFromFc: the divide level between fgear and fc

The value could be the following values:

- **CG_DIVIDE_1**: fgear = fc
- **CG_DIVIDE_2**: fgear = fc/2
- **CG_DIVIDE_4**: fgear = fc/4
- **CG_DIVIDE_8**: fgear = fc/8
- **CG_DIVIDE_16**: fgear = fc/16

Description :

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

Return:

None

4.2.3.2 CG_GetFgearLevel

Get the dividing level between fgear and fc.

Prototype:

CG_DivideLevel

CG_GetFgearLevel(void)

Parameters:

None

Description:

This function will get the dividing level between fgear and fc.

If the value "Reserved" is read from the register, the API will return

CG_DIVIDE_UNKNOWN.

Return:

The dividing level between clock fgear and fc.

The value returned can be one of the following values:

- CG_DIVIDE_1**: fgear = fc
- CG_DIVIDE_2**: fgear = fc/2
- CG_DIVIDE_4**: fgear = fc/4
- CG_DIVIDE_8**: fgear = fc/8
- CG_DIVIDE_16**: fgear = fc/16

CG_DIVIDE_UNKNOWN: invalid data is read

4.2.3.3 CG_SetPhiT0Src

Set fperiph for PhiT0.

Prototype:

void

CG_SetPhiT0Src(CG_PhiT0Src **PhiT0Src**)

Parameters:

PhiT0Src: Select PhiT0 source.

This parameter can be one of the following values:

- **CG_PHIT0_SRC_FGEAR** means PhiT0 source is fgear.
- **CG_PHIT0_SRC_FC** means PhiT0 source is fc.

Description:

This function selects the source for PhiT0.

Return:

None

4.2.3.4 CG_GetPhiT0Src

Get the PhiT0 source.

Prototype:

CG_PhiT0Src

CG_GetPhiT0Src(void)

Parameters:

None

Description:

This function will get the PhiT0 source.

Return:

CG_PHIT0_SRC_FGEAR means PhiT0 source is fgear.

CG_PHIT0_SRC_FC means PhiT0 source is fc.

4.2.3.5 CG_SetPhiT0Level

Set the dividing level between PhiT0 ($\Phi T0$) and fc.

Prototype:

Result

CG_SetPhiT0Level(CG_DivideLevel ***DividePhiT0FromFc***)

Parameters:

DividePhiT0FromFc: divide level between PhiT0($\Phi T0$) and fc.

This parameter can be one of the following values:

- **CG_DIVIDE_1**: $\Phi T0 = fc$
- **CG_DIVIDE_2**: $\Phi T0 = fc/2$
- **CG_DIVIDE_4**: $\Phi T0 = fc/4$
- **CG_DIVIDE_8**: $\Phi T0 = fc/8$
- **CG_DIVIDE_16**: $\Phi T0 = fc/16$
- **CG_DIVIDE_32**: $\Phi T0 = fc/32$
- **CG_DIVIDE_64**: $\Phi T0 = fc/64$
- **CG_DIVIDE_128**: $\Phi T0 = fc/128$
- **CG_DIVIDE_256**: $\Phi T0 = fc/256$
- **CG_DIVIDE_512**: $\Phi T0 = fc/512$

Description:

This function will set the dividing level of prescaler clock.

Return:

SUCCESS means the setting has been written to registers successfully.

ERROR means the setting has not been written to registers.

4.2.3.6 CG_GetPhiT0Level

Get the dividing level between clock $\Phi T0$ 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 $\Phi T0$ and fc, the value will be one of the following:

CG_DIVIDE_1: $\Phi T0 = fc$

CG_DIVIDE_2: $\Phi T0 = fc/2$

CG_DIVIDE_4: $\Phi T0 = fc/4$
CG_DIVIDE_8: $\Phi T0 = fc/8$
CG_DIVIDE_16: $\Phi T0 = fc/16$
CG_DIVIDE_32: $\Phi T0 = fc/32$
CG_DIVIDE_64: $\Phi T0 = fc/64$
CG_DIVIDE_128: $\Phi T0 = fc/128$
CG_DIVIDE_256: $\Phi T0 = fc/256$
CG_DIVIDE_512: $\Phi T0 = fc/512$
CG_DIVIDE_UNKNOWN: invalid data is read.

4.2.3.7 CG_SetSCOUTSrc

Set the clock source of SCOUT output.

Prototype:

void
CG_SetSCOUTSrc(CG_SCOUTSrc **Source**)

Parameters:

Source: select clock source of SCOUT.

This parameter can be one of the following values:

- **CG_SCOUT_FC_DIVIDE_4:** SCOUT source is set to $fc/4$.
- **CG_SCOUT_FC_DIVIDE_8:** SCOUT source is set to $fc/8$.
- **CG_SCOUT_FOSC:** SCOUT source is set to $fosc$.
- **CG_SCOUT_LOW:** "Low" is output from the pin.

Description:

This function will set the source for the clock output from SCOUT pin.

Return:

None

4.2.3.8 CG_GetSCOUTSrc

Get the clock source of SCOUT output.

Prototype:

SCOUTSrc
CG_GetSCOUTSrc(void)

Parameters:

None

Description:

This function will get the clock source of SCOUT output.

Return:

The clock source of SCOUT output:

- **CG_SCOUT_FC_DIVIDE_4**: SCOUT source is to fc/4.
- **CG_SCOUT_FC_DIVIDE_8**: SCOUT source is to fc/8.
- **CG_SCOUT_FOSC**: SCOUT source is to fosc.
- **CG_SCOUT_LOW**: SCOUT output is prohibited.

4.2.3.9 CG_SetWarmUpTime

Set the warm up time.

Prototype:

void

CG_SetWarmUpTime(CG_WarmUpSrc **Source**,
uint16_t **Time**)

Parameters:

Source: select source of warm-up counter.

- **CG_WARM_UP_SRC_OSC_INT**: internal high-speed oscillator is selected as timer source.
- **CG_WARM_UP_SRC_OSC_EXT**: external high-speed oscillator is selected as timer source.

Time: select count time of warm-up timer.

Max value is 0xFFFF.

Description:

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

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

Example of calculating register value for warm-up time:

/* When using high-speed oscillator 8MHz, and set warm-up time 5ms. */

So value = (warm-up time to set) / (input frequency cycle(s)) = 5ms / (1/8MHz) = 4000cycle = 0x9C40.

So set **Time** = 0x9C40,

Return:

None.

4.2.3.10 CG_StartWarmUp

Start operation of warm up timer for oscillator.

Prototype:

void
CG_StartWarmUp(void)

Parameters:

None

Description:

This function will start the warm up timer.

Return:

None

4.2.3.11 CG_GetWarmUpState

Check whether warm up is completed or not.

Prototype:

WorkState
CG_GetWarmUpState(void)

Parameters:

None

Description:

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

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

Return:

Warm up state:

DONE: means warm-up operation is finished.

BUSY: means warm-up operation is in progress.

4.2.3.12 CG_SetFPLLValue

Set multiplying value for PLL0 (for fsys).

Prototype:

Result

CG_SetFPLLValue(CG_FpllValue **NewValue**)

Parameters:

NewValue:

- **CG_FPLL0_IN8_OUT40:** Input clock 8MHz, 5 multiplying value is selected.
- **CG_FPLL0_IN8_OUT48:** Input clock 8MHz, 6 multiplying value is selected.
- **CG_FPLL0_IN8_OUT64:** Input clock 8MHz, 8 multiplying value is selected.
- **CG_FPLL0_IN8_OUT80:** Input clock 8MHz, 10 multiplying value is selected.
- **CG_FPLL0_IN10_OUT50:** Input clock 10MHz, 5 multiplying value is selected.
- **CG_FPLL0_IN10_OUT60:** Input clock 10MHz, 6 multiplying value is selected.
- **CG_FPLL0_IN10_OUT80:** Input clock 10MHz, 8 multiplying value is selected.
- **CG_FPLL0_IN10_OUT100:** Input clock 10MHz, 10 multiplying value is selected.

Description:

This function sets multiplying value for PLL0 (for fsys).

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

4.2.3.13 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.

4.2.3.14 CG_GetPLLState

Get the state of PLL0 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.

4.2.3.15 CG_SetFPLLForADCValue

Set multiplying value for PLL1 (for ADC).

Prototype:

Result

CG_SetFPLLForADCValue(uint32_t **NewValue**)

Parameters:

NewValue: select PLL multiplying value.

This parameter can be one of the following values:

- **CG_FPLL1_IN8_OUT64:** Input clock 8MHz, 6 multiplying value is selected.
- **CG_FPLL1_IN8_OUT80:** Input clock 8MHz, 10 multiplying value is selected.
- **CG_FPLL1_IN10_OUT80:** Input clock 10MHz, 8 multiplying value is selected.
- **CG_FPLL1_IN10_OUT100:** Input clock 10MHz, 10 multiplying value is selected.

Description:

This function sets multiplying value for PLL1 (for ADC).

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

4.2.3.16 CG_SetFadcSrc

Set the clock source of ADC.

Prototype:

void

CG_SetFadcSrc(CG_FadcSrc **FadcSrc**)

Parameters:

FadcSrc: Select clock source for ADC.

This parameter can be one of the following values:

- **CG_FADC_SRC_FC**: Use of fc
- **CG_FADC_SRC_FPLL**: Use of fplladc.

Description:

This function sets the clock source of ADC.

Return:

None.

4.2.3.17 CG_SetPLL1ForADC

Enable PLL1 for ADC or disable it.

Prototype:

void

CG_SetPLL1ForADC(FunctionalState **NewState**)

Parameters:

NewState: New state of PLL1.

This parameter can be one of the following values:

- **ENABLE**: to enable PLL1 for ADC.
- **DISABLE**: to disable PLL1 for ADC.

Description:

This function enables PLL1 for ADC or disables it.

Return:

None.

4.2.3.18 CG_SetFosc

Enable or disable high-speed oscillator (fosc).

Prototype:

Result

CG_SetFosc(CG_FoscSrc **Source**,
FunctionalState **NewState**)

Parameters:

Source: select clock source of fosc.

This parameter can be one of the following values:

- **CG_FOSC_OSC_EXT:** external high-speed oscillator is selected,
- **CG_FOSC_OSC_INT:** internal high-speed oscillator is selected.

NewState

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

Description:

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

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

4.2.3.19 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

4.2.3.20 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.

4.2.3.21 CG_GetFoscState

Get the state of the high-speed oscillator.

Prototype:

FunctionalState

CG_GetFoscState(CG_FoscSrc **Source**)

Parameters:

Source: select source for fosc.

➤ **CG_FOSC_OSC_EXT:** external high-speed oscillator is selected,

➤ **CG_FOSC_OSC_INT:** internal high-speed oscillator is selected.

Description:

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

Return:

The state of fosc

ENABLE: fosc is enabled.

DISABLE: fosc is disabled.

4.2.3.22 CG_SetFs

Enable or disable low-speed oscillation (fs).

Prototype:

Result

CG_SetFs(FunctionalState **NewState**)

Parameters:

NewState: oscillation is enabled or disabled.

- **ENABLE**: low-speed oscillation (fs) is enabled.
- **DISABLE**: low-speed oscillation (fs) is disabled.

Description:

This function will enable or disable low-speed oscillation (fs).

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

4.2.3.23 CG_GetFsState

Get the state of low-speed oscillation (fs).

Prototype:

FunctionalState

CG_GetFsState(void)

Parameters:

None

Description:

This function will get the state of low-speed oscillation (fs).

Return:

The state of fs

ENABLE: fs is enabled.

DISABLE: fs is disabled.

4.2.3.24 CG_SetSTBYMode

Set to the specified low-power 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_STOP2:** STOP2 mode. This mode halts main voltage supply, retaining some function operation.
- **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

4.2.3.25 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_STOP2: STOP2 mode

CG_STBY_MODE_IDLE: IDLE mode

CG_STBY_MODE_UNKNOWN: Invalid data is read.

4.2.3.26 CG_SetPinStateInStop1Mode

Set pin status in stop1 mode

Prototype:

void

CG_SetPinStateInStop1Mode(FunctionalState **NewState**)

Parameters:**NewState:**

➤ **DISABLE:** <DRVE>=0

➤ **ENABLE:** <DRVE>=1

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

Description:

This function sets pin status in stop1 mode.

Return:

None

4.2.3.27 CG_GetPinStateInStop1Mode

Get pin status in stop1 mode

Prototype:

FunctionalState

CG_GetPinStateInStop1Mode(void)

Parameters:

None

Description:

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

Return:

The pin state in stop1 mode

DISABLE: <DRVE>=0

ENABLE: <DRVE>=1

4.2.3.28 CG_SetPortKeepInStop2Mode

Enables or disables to keep IO control signal in stop2 mode

Prototype:

void

CG_SetPortKeepInStop2Mode(FunctionalState **NewState**)

Parameters:**NewState:**

- **DISABLE:** <PTKEEP>=0
- **ENABLE:** <PTKEEP>=1

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

Description:

This function enables or disables to keep IO control signal in stop2 mode.

Return:

None

4.2.3.29 CG_GetPortKeepInStop2Mode

Get the pin status in stop mode

Prototype:

FunctionalState

CG_GetPinStateInStopMode(void)

Parameters:

None

Description:

This function will get the status of IO control signal in stop2 mode.

Return:

The port keep in stop2 mode

DISABLE: <PTKEEP>=0

ENABLE: <PTKEEP>=1

4.2.3.30 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 source for fc successfully

ERROR: clock source of fc is not changed.

4.2.3.31 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.

4.2.3.32 CG_SetFtmrdSrc

Set the clock source of high-resolution timer TMRD.

Prototype:

void

CG_SetFtmrdSrc(CG_TmrdUnit *TmrdUnit*,
CG_FtmrdSrc *FtmrdSrc*)

Parameters:

TmrdUnit: Select unit for TIMER-D

This parameter can be one of the following values:

CG_TMRD_UNIT_A: Select unit A.

FtmrdSrc: Select clock source for ftmrd

This parameter can be one of the following values:

- **CG_FTMRD_SRC_FPLL:** ftmrd source will be set to fpll
- **CG_FTMRD_SRC_HALF_FPLL:** ftmrd source will be set to fpll/2
- **CG_FTMRD_SRC_QUARTER_FPLL:** ftmrd source will be set to fpll/4

Description:

This function sets the source of high-resolution PPG clock (ftmrd).

Return:

None

4.2.3.33 CG_GetFtmrdSrc

Get the clock source of high-resolution timer TMRD.

Prototype:

CG_FtmrdSrc

CG_GetFtmrdSrc(CG_TmrdUnit *TmrdUnit*)

Parameters:

TmrdUnit. Select unit for TIMER-D

This parameter can be one of the following values:

CG_TMRD_UNIT_A: Select unit A.

Description:

This function gets the source of high-resolution PPG clock (ftmrd).

Return:

Source of ftmrd:

CG_FTMRD_SRC_FPLL: ftmrd source is fpll

CG_FTMRD_SRC_HALF_FPLL: ftmrd source is fpll/2

CG_FTMRD_SRC_QUARTER_FPLL: ftmrd source is fpll/4

CG_FTMRD_SRC_UNKNOWN: Invalid data is read

4.2.3.34 CG_SetTMRDCIk

Enable or disable to provide TMRD clock.

Prototype:

void

CG_SetTMRDCIk(CG_TmrdUnit *TmrdUnit*,
FunctionalState *NewState*)

Parameters:

TmrdUnit: Select unit for TIMER-D

This parameter can be one of the following values:

CG_TMRD_UNIT_A: Select unit A.

NewState

DISABLE: Enable providing TMRD clock

ENABLE: Disable providing TMRD clock

Description:

This function enables or disables to provide TMRD clock.

Return:

None

4.2.3.35 CG_GetTMRDClkState

Get the state of TMRD clock.

Prototype:

FunctionalState

CG_GetTMRDClkState(CG_TmrdUnit ***TmrdUnit***)

Parameters:

TmrdUnit: Select unit for TIMER-D

This parameter can be one of the following values:

CG_TMRD_UNIT_A: Select unit A.

Description:

This function gets the state of TMRD clock.

Return:

State of TMRD clock:

ENABLE: Providing TMRD clock is enabled.

DISABLE: Providing TMRD clock is disabled.

4.2.3.36 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

4.2.3.37 CG_SetSTBYReleaseINTSrc

Set the INT source for releasing low power mode.

Prototype:

void

CG_SetSTBYReleaseINTSrc(CG_INTSrc **INTSource**,
CG_INTActiveState **ActiveState**,
FunctionalState **NewState**)

Parameters:

INTSource: select the INT source for releasing standby mode

This parameter can be one of the following values:

- **CG_INT_SRC_0:** INT0
- **CG_INT_SRC_1:** INT1
- **CG_INT_SRC_2:** INT2
- **CG_INT_SRC_3:** INT3
- **CG_INT_SRC_4:** INT4
- **CG_INT_SRC_5:** INT5
- **CG_INT_SRC_6:** INT6
- **CG_INT_SRC_7:** INT7
- **CG_INT_SRC_8:** INT8
- **CG_INT_SRC_9:** INT9
- **CG_INT_SRC_A:** INTA
- **CG_INT_SRC_B:** INTB
- **CG_INT_SRC_C:** INTC
- **CG_INT_SRC_D:** INTD
- **CG_INT_SRC_E:** INTE
- **CG_INT_SRC_F:** INTF
- **CG_INT_SRC_10:** INT10
- **CG_INT_SRC_11:** INT11
- **CG_INT_SRC_12:** INT12
- **CG_INT_SRC_13:** INT13
- **CG_INT_SRC_14:** INT14
- **CG_INT_SRC_15:** INT15

- **CG_INT_SRC_INTKWUP0**: INTKWUP0
- **CG_INT_SRC_INTKWUP1**: INTKWUP1
- **CG_INT_SRC_INTKSCAN**: INTKSCAN
- **CG_INT_SRC_INTRTC**: INTRTC
- **CG_INT_SRC_INTPHC00**: INTPHC00
- **CG_INT_SRC_INTPHC01**: INTPHC01
- **CG_INT_SRC_INTPHC0EVRY**: INTPHC0EVRY
- **CG_INT_SRC_INTPHC10**: INTPHC10
- **CG_INT_SRC_INTPHC11**: INTPHC11
- **CG_INT_SRC_INTPHC1EVRY**: INTPHC1EVRY

ActiveState: select the active state for release trigger.

When **INTSource** is equal to one of **CG_INT_SRC_0** to **CG_INT_SRC_15**, it can be one of the following values:

- **CG_INT_ACTIVE_STATE_L**: active on low level
- **CG_INT_ACTIVE_STATE_H**: active on high level
- **CG_INT_ACTIVE_STATE_FALLING**: active on falling edge
- **CG_INT_ACTIVE_STATE_RISING**: active on rising edge
- **CG_INT_ACTIVE_STATE_BOTH_EDGES**: active on both edges

When **INTSource** is equal to one of **CG_INT_SRC_INTKWUP0** to **CG_INT_SRC_INTKWUP1**, it can be one of the following values:

- **CG_INT_ACTIVE_STATE_H**: active on high level

When **INTSource** is equal to **CG_INT_SRC_INTRTC**, it can be one of the following values:

- **CG_INT_ACTIVE_STATE_FALLING**: active on falling edge

When **INTSource** is equal to one of the others, it can be one of the following values:

- **CG_INT_ACTIVE_STATE_RISING**: active on rising edge

NewState: enable or disable this release trigger

This parameter can be one of the following values:

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

Description:

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

Return:

None

4.2.3.38 CG_GetSTBYReleaseINTState

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

Prototype:

CG_INT_ActiveState

CG_GetSTBYReleaseINTSrc(CG_INTSrc **INTSource**)

Parameters:

INTSource: select the release INT source

This parameter can be one of the following values:

- **CG_INT_SRC_0:** INT0
- **CG_INT_SRC_1:** INT1
- **CG_INT_SRC_2:** INT2
- **CG_INT_SRC_3:** INT3
- **CG_INT_SRC_4:** INT4
- **CG_INT_SRC_5:** INT5
- **CG_INT_SRC_6:** INT6
- **CG_INT_SRC_7:** INT7
- **CG_INT_SRC_8:** INT8
- **CG_INT_SRC_9:** INT9
- **CG_INT_SRC_A:** INTA
- **CG_INT_SRC_B:** INTB
- **CG_INT_SRC_C:** INTC
- **CG_INT_SRC_D:** INTD
- **CG_INT_SRC_E:** INTE
- **CG_INT_SRC_F:** INTF
- **CG_INT_SRC_10:** INT10
- **CG_INT_SRC_11:** INT11
- **CG_INT_SRC_12:** INT12
- **CG_INT_SRC_13:** INT13
- **CG_INT_SRC_14:** INT14
- **CG_INT_SRC_15:** INT15
- **CG_INT_SRC_INTKWUP0:** INTKWUP0
- **CG_INT_SRC_INTKWUP1:** INTKWUP1
- **CG_INT_SRC_INTKSCAN:** INTKSCAN
- **CG_INT_SRC_INTRTC:** INTRTC
- **CG_INT_SRC_INTPHC00:** INTPHC00
- **CG_INT_SRC_INTPHC01:** INTPHC01
- **CG_INT_SRC_INTPHC0EVRY:** INTPHC0EVRY
- **CG_INT_SRC_INTPHC10:** INTPHC10
- **CG_INT_SRC_INTPHC11:** INTPHC11
- **CG_INT_SRC_INTPHC1EVRY:** INTPHC1EVRY

Description:

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

Return:

Active state of the input INT

The value returned can be one of the following values:

CG_INT_ACTIVE_STATE_L: active on low level

CG_INT_ACTIVE_STATE_H: active on high level

CG_INT_ACTIVE_STATE_FALLING: active on falling edge

CG_INT_ACTIVE_STATE_RISING: active on rising edge

CG_INT_ACTIVE_STATE_BOTH_EDGES: active on both edges

4.2.3.39 CG_ClearINTReq

Clears the input INT request.

Prototype:

void

CG_ClearINTReq(CG_INTSrc **INTSource**)

Parameters:

INTSource: select the release INT source.

This parameter can be one of the following values:

- **CG_INT_SRC_0**: INT0
- **CG_INT_SRC_1**: INT1
- **CG_INT_SRC_2**: INT2
- **CG_INT_SRC_3**: INT3
- **CG_INT_SRC_4**: INT4
- **CG_INT_SRC_5**: INT5
- **CG_INT_SRC_6**: INT6
- **CG_INT_SRC_7**: INT7
- **CG_INT_SRC_8**: INT8
- **CG_INT_SRC_9**: INT9
- **CG_INT_SRC_A**: INTA
- **CG_INT_SRC_B**: INTB
- **CG_INT_SRC_C**: INTC
- **CG_INT_SRC_D**: INTD
- **CG_INT_SRC_E**: INTE
- **CG_INT_SRC_F**: INTF
- **CG_INT_SRC_10**: INT10
- **CG_INT_SRC_11**: INT11
- **CG_INT_SRC_12**: INT12
- **CG_INT_SRC_13**: INT13
- **CG_INT_SRC_14**: INT14
- **CG_INT_SRC_15**: INT15
- **CG_INT_SRC_INTKWUP0**: INTKWUP0

- **CG_INT_SRC_INTKWUP1:** INTKWUP1
- **CG_INT_SRC_INTKSCAN:** INTKSCAN
- **CG_INT_SRC_INTRTC:** INTRTC
- **CG_INT_SRC_INTPHC00:** INTPHC00
- **CG_INT_SRC_INTPHC01:** INTPHC01
- **CG_INT_SRC_INTPHC0EVRY:** INTPHC0EVRY
- **CG_INT_SRC_INTPHC10:** INTPHC10
- **CG_INT_SRC_INTPHC11:** INTPHC11
- **CG_INT_SRC_INTPHC1EVRY:** INTPHC1EVRY
-

Description:

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

Return:

None

4.2.3.40 CG_GetResetFlag

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

Prototype:

CG_ResetFlag

CG_GetResetFlag(void)

Parameters:

None

Description:

This function gets the reset flag showing what triggered reset.

Return:

Reset flag:

PowerOnReset1 (Bit0) Reset by power on reset

PinReset (Bit1) Reset by reset pin

WDTReset (Bit 2) means reset from WDT.

STOP2Reset(Bit3) means reset flag by STOP2 mode release

SYSReset(Bit4) Reset by SYSResetREQ

PowerOnReset2 (Bit6) Reset by power on reset

4.2.3.41 CG_SetPeriphClkSupply

Enable or disable supplying clock Fc for PSC and ADC(unit A, B, C).

Prototype:

void

CG_SetPeriphClkSupply (uint32_t **Periph**, FunctionalState **NewState**)

Parameters:

Periph: The target peripheral that CG supplies fc for

- **CG_PSC_CLK_SUPPLY:** PSC clock supply
- **CG_ADC_C_CLK_SUPPLY:** ADC unit C clock supply
- **CG_ADC_B_CLK_SUPPLY:** ADC unit B clock supply
- **CG_ADC_A_CLK_SUPPLY:** ADC unit A clock supply

NewState: New state of clock Fc supply setting.

DISABLE: Enable supplying clock Fc

ENABLE: Disable supplying clock Fc

Description:

This function enables or disables supplying clock Fc for PSC and ADC (unit A, B, C).

Return:

None

4.2.3.42 CG_SetFclkPeriphA

Enable or disable supplying clock fclk to modules group A.

Prototype:

void

CG_SetFclkPeriphA(uint32_t **Periph**, FunctionalState **NewState**)

Parameters:

Periph: The target peripheral that CG supplies clock fclk for

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

- **FCLK_A_TMRB10:** Clock supply control for TMRB channel 10
- **FCLK_A_TMRB11:** Clock supply control for TMRB channel 11
- **FCLK_A_TMRB12:** Clock supply control for TMRB channel 12
- **FCLK_A_TMRB13:** Clock supply control for TMRB channel 13
- **FCLK_A_TMRB14:** Clock supply control for TMRB channel 14
- **FCLK_A_TMRB15:** Clock supply control for TMRB channel 15
- **FCLK_A_TMRB16:** Clock supply control for TMRB channel 16
- **FCLK_A_TMRB17:** Clock supply control for TMRB channel 17
- **FCLK_A_TMRB18:** Clock supply control for TMRB channel 18
- **FCLK_A_TMRB19:** Clock supply control for TMRB channel 19
- **FCLK_A_DAC0** : Clock supply control for DAC channel 0
- **FCLK_A_DAC1** : Clock supply control for DAC channel 1
- **FCLK_A_EBIF** : Clock supply control for EBIF

- **FCLK_A_UART0** : Clock supply control for UART channel 0
- **FCLK_A_UART1** : Clock supply control for UART channel 1
- **FCLK_A_DMACA** : Clock supply control for DMAC Unit A
- **FCLK_A_DMACB** : Clock supply control for DMAC Unit B
- **FCLK_A_DMACC** : Clock supply control for DMAC Unit C
- **FCLK_A_PORTA** : Clock supply control for port A
- **FCLK_A_PORTB** : Clock supply control for port B
- **FCLK_A_PORTC** : Clock supply control for port C
- **FCLK_A_PORTD** : Clock supply control for port D
- **FCLK_A_PORTE** : Clock supply control for port E
- **FCLK_A_PORTF** : Clock supply control for port F
- **FCLK_A_PORTG** : Clock supply control for port G
- **FCLK_A_PORTH** : Clock supply control for port H
- **FCLK_A_PORTJ** : Clock supply control for port J
- **FCLK_A_PORTK** : Clock supply control for port K
- **FCLK_A_PORTL** : Clock supply control for port L
- **FCLK_A_PORTM** : Clock supply control for port M
- **FCLK_A_PORTN** : Clock supply control for port N
- **FCLK_A_ALL** : Clock supply control for all

NewState: New state of clock supply setting.

DISABLE: Enable supplying clock

ENABLE: Disable supplying clock

Description:

This function enables or disables supplying clock fclk to modules group A

Return:

None

4.2.3.43 CG_SetFclkPeriphB

Enable or disable supplying clock fclk to modules group B.

Prototype:

void

CG_SetFclkPeriphB(uint32_t **Periph**, FunctionalState **NewState**)

Parameters:

Periph: The target peripheral that CG supplies clock fclk for

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

- **FCLK_B_PORTP** : Clock supply control for port P
- **FCLK_B_PORTR** : Clock supply control for port R
- **FCLK_B_PORTT** : Clock supply control for port T

- **FCLK_B_PORTU** : Clock supply control for port U
- **FCLK_B_PORTV** : Clock supply control for port V
- **FCLK_B_PORTW** : Clock supply control for port W
- **FCLK_B_PORTY** : Clock supply control for port Y
- **FCLK_B_PORTAA**: Clock supply control for port AA
- **FCLK_B_PORTAB**: Clock supply control for port AB
- **FCLK_B_PORTAC**: Clock supply control for port AC
- **FCLK_B_PORTAD**: Clock supply control for port AD
- **FCLK_B_PORTAE**: Clock supply control for port AE
- **FCLK_B_PORTAF**: Clock supply control for port AF
- **FCLK_B_PORTAG**: Clock supply control for port AG
- **FCLK_B_PORTAH**: Clock supply control for port AH
- **FCLK_B_PORTAJ**: Clock supply control for port AJ
- **FCLK_B_ADCA** : Clock supply control for ADC Unit A
- **FCLK_B_ADCB** : Clock supply control for ADC Unit B
- **FCLK_B_ADCC** : Clock supply control for ADC Unit C
- **FCLK_B_EPHC** : Clock supply control for EPHC
- **FCLK_B_SBI** : Clock supply control for SBI
- **FCLK_B_WDT** : Clock supply control for WDT
- **FCLK_B_ALL** : Clock supply control for all

NewState: New state of clock supply setting.

DISABLE: Enable supplying clock

ENABLE: Disable supplying clock

Description:

This function enables or disables supplying clock fclk to modules group B

Return:

None

4.2.3.44 CG_SetFcPeriphA

Enable or disable supplying clock fc to modules group A

Prototype:

void

CG_SetFcPeriphA(uint32_t **Periph**, FunctionalState **NewState**)

Parameters:

Periph: The target peripheral that CG supplies clock fclk for

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

- **FC_A_ESIO0** : Clock supply control for ESIO channel 0
- **FC_A_ESIO1** : Clock supply control for ESIO channel 1

- **FC_A_ESIO2** : Clock supply control for ESIO channel 2
- **FC_A_TMRD** : Clock supply control for TMRD
- **FC_A_SIO_UART0** : Clock supply control for SIO/UART channel 0
- **FC_A_SIO_UART1** : Clock supply control for SIO/UART channel 1
- **FC_A_SIO_UART2** : Clock supply control for SIO/UART channel 2
- **FC_A_SIO_UART3** : Clock supply control for SIO/UART channel 3
- **FC_A_SIO_UART4** : Clock supply control for SIO/UART channel 4
- **FC_A_SIO_UART5** : Clock supply control for SIO/UART channel 5
- **FC_A_TMRB00** : Clock supply control for TMRB channel 00
- **FC_A_TMRB01** : Clock supply control for TMRB channel 01
- **FC_A_TMRB02** : Clock supply control for TMRB channel 02
- **FC_A_TMRB03** : Clock supply control for TMRB channel 03
- **FC_A_TMRB04** : Clock supply control for TMRB channel 04
- **FC_A_TMRB05** : Clock supply control for TMRB channel 05
- **FC_A_TMRB06** : Clock supply control for TMRB channel 06
- **FC_A_TMRB07** : Clock supply control for TMRB channel 07
- **FC_A_TMRB08** : Clock supply control for TMRB channel 08
- **FC_A_TMRB09** : Clock supply control for TMRB channel 09
- **FC_A_TMRCCAP0** : Clock supply control for TMRC capture channel 0
- **FC_A_TMRCCAP1** : Clock supply control for TMRC capture channel 1
- **FC_A_TMRCCAP2** : Clock supply control for TMRC capture channel 2
- **FC_A_TMRCCAP3** : Clock supply control for TMRC capture channel 3
- **FC_A_TMRCCMP0** : Clock supply control for channel 0 in TMRC compare circuit
- **FC_A_TMRCCMP1** : Clock supply control for channel 1 in TMRC compare circuit
- **FC_A_TMRCCMP2** : Clock supply control for channel 2 in TMRC compare circuit
- **FC_A_TMRCCMP3** : Clock supply control for channel 3 in TMRC compare circuit
- **FC_A_TMRCCMP4** : Clock supply control for channel 4 in TMRC compare circuit
- **FC_A_TMRCCMP5** : Clock supply control for channel 5 in TMRC compare circuit
- **FC_A_TMRCCMP6** : Clock supply control for channel 6 in TMRC compare circuit
-
- **FC_A_TMRCCMP7** : Clock supply control for channel 7 in TMRC compare circuit
- **FC_A_ALL** : Clock supply control for all

NewState: New state of clock supply setting.

DISABLE: Enable supplying clock

ENABLE: Disable supplying clock

Description:

This function enables or disables supplying clock fc to modules group A

Return:

None

4.2.3.45 CG_SetFcPeriphB

Enable or disable supplying clock fc to modules group B.

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:

- **FCLK_B_TMRCTBT** : Clock supply control for TBTcircuit of TMRC

NewState: New state of clock supply setting.

DISABLE: Enable supplying clock

ENABLE: Disable supplying clock

Description:

This function enables or disables supplying clock fc to modules group B

Return:

None

4.2.4 Data Structure Description

4.2.4.1 CG_ResetFlag

Data Fields:

uint32_t

All specifies CG reset source.

Bit Fields:

uint32_t

PowerOnReset1(Bit0) Reset by power on reset

uint32_t

PinReset (Bit1) Reset by reset pin

uint32_t

WDTReset(Bit2) Reset from WDT

uint32_t

STOP2Reset(Bit3) Reset flag by STOP2 mode release

uint32_t

SYSReset (Bit4) Reset by SYSResetREQ

uint32_t

Reserved1 (Bit5) Reserved

uint32_t

PowerOnReset2(Bit6) Reset by power on reset

uint32_t

Reserved2 (Bit7~bit31) Reserved

5. DAC

5.1 Overview

Features

- A high-resolution, 10-bit DA converter is built in.
- Built in full range buffer amplifier.
- Built in power down function.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_dac.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_dac.h containing the macros, data types, structures and API definitions for use by applications.

5.2 API Functions

5.2.1 Function List

- ◆ void DAC_SetOutputCode(TSB_DA_TypeDef * **DACx**, uint16_t **OutputCode**);
- ◆ void DAC_Start(TSB_DA_TypeDef * **DACx**);
- ◆ void DAC_Stop(TSB_DA_TypeDef * **DACx**);
- ◆ void DAC_SetVOutHoldTime(TSB_DA_TypeDef * **DACx**);

5.2.2 Detailed Description

Functions listed above can be divided into two parts:

- 1) Configure the DAC channel and set output value
DAC_SetOutputCode(), DAC_SetVOutHoldTime()
- 2) Start and stop control
DAC_Start(), DAC_Stop()

5.2.3 Function Documentation

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

TSB_DAA, TSB_DAB

5.2.3.1 DAC_SetOutputCode

Set output code for the specified DAC channel.

Prototype:

```
void  
DAC_SetOutputCode(TSB_DA_TypeDef * DACx,  
                  uint16_t OutputCode)
```

Parameters:

DACx is the specified DAC channel.

OutputCode is the value which will be converted to analog output. And its bit width is 10, so its max value is 0x3ff.

Description:

This function sets the output code for the specified DAC channel, which will be converted to analog output.

Return:

None

5.2.3.2 DAC_Start

Start the operation of the specified DAC channel

Prototype:

```
void  
DAC_Start(TSB_DA_TypeDef * DACx);
```

Parameters:

DACx is the specified DAC channel.

Description:

This function will start the Digital to Analog converting operation of the specified DAC channel.

Return:

None

5.2.3.3 DAC_Stop

Stop the operation of the specified DAC channel

Prototype:

```
void  
DAC_Stop(TSB_DA_TypeDef * DACx);
```

Parameters:

DACx is the specified DAC channel.

Description:

This function will stop the Digital to Analog converting operation of the specified DAC channel.

Return:

None

5.2.3.4 DAC_SetVOutHoldTime

Adjust VOut holdtime.

Prototype:

void

DAC_SetVOutHoldTime(TSB_DA_TypeDef * **DACx**);

Parameters:

DACx is the specified DAC channel.

Description:

Adjust VOut holdtime.

Return:

None

5.2.4 Data Structure Description

None

6. DMAC

6.1 Overview

TOSHIBA TMPM440 has three DMA controllers(UNITA, UNITB, UNITC) controlled by DMA request select registers, and each DMA controller has two channels. Each channel can operate in one of four transferring types (memory to memory, memory to peripheral, peripheral to memory, peripheral to peripheral). The priority of UNITA is highest and UNITB higher than UNITC. The priority of channel 0 is higher than channel 1.

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

This driver is contained in \Libraries\TX04_Periph_Driver\src\tmpm440_dmac.c, with \Libraries\TX04_Periph_Driver\inc\tmpm440_dmac.h containing the API definitions for use by applications.

6.2 API Functions

6.2.1 Function List

- void DMAC_Enable(TSB_DMAL_TypeDef * **DMACx**);
- void DMAC_Disable(TSB_DMAL_TypeDef * **DMACx**);
- DMAL_INTRReq DMAL_GetINTRReq(TSB_DMAL_TypeDef * **DMACx**);
- DMAL_TxINTRReq DMAL_GetTxINTRReq(TSB_DMAL_TypeDef * **DMACx**, DMAL_Channel **Chx**);
- void DMAL_ClearTxINTRReq(TSB_DMAL_TypeDef * **DMACx**, DMAL_Channel **Chx**, DMAL_INTSrc **INTSource**);
- DMAL_TxINTRReq DMAL_GetRawTxINTRReq(TSB_DMAL_TypeDef * **DMACx**, DMAL_Channel **Chx**);
- WorkState DMAL_GetChannelTxState(TSB_DMAL_TypeDef * **DMACx**, DMAL_Channel **Chx**);
- void DMACA_SetSWBurstReq(DMACA_ReqNum **BurstReq**);
- void DMACB_SetSWBurstReq(DMACB_ReqNum **BurstReq**);
- void DMACC_SetSWBurstReq(DMACC_ReqNum **BurstReq**);
- DMAL_BurstReqState DMAL_GetSWBurstReqState(TSB_DMAL_TypeDef * **DMACx**);
- void DMACB_SetSWSingleReq(DMACB_ReqNum **SingleReq**);

- void DMACC_SetSWSingleReq(DMACC_ReqNum **SingleReq**);
- DMAC_SingleReqState DMAC_GetSWSingleReqState(TSB_DMAC_TypeDef * **DMACx**);
- void DMAC_SetLinkedList(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**, uint32_t **LinkedAddr**);
- WorkState DMAC_GetFIFOState(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**);
- void DMAC_SetDMAHalt(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**, FunctionalState **NewState**);
- void DMAC_SetLockedTx(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**, FunctionalState **NewState**);
- void DMAC_SetTxINTConfig(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**, DMAC_INTSrc **INTSource**, FunctionalState **NewState**);
- void DMAC_SetDMAChannel(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**, FunctionalState **NewState**);
- void DMAC_Init(TSB_DMAC_TypeDef * **DMACx**, DMAC_Channel **Chx**, DMAC_InitTypeDef * **InitStruct**);

6.2.2 Detailed Description

Functions listed above can be divided into five parts:

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

6.2.3 Function Documentation

6.2.3.1 DMAC_Enable

Enable the DMA circuit.

Prototype:

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

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Description:

This function will Enable UNIT A circuit when **DMACx** is **DMAC_UNIT_A** and Enable UNIT B circuit when **DMACx** is **DMAC_UNIT_B** and Enable UNIT C circuit when **DMACx** is **DMAC_UNIT_C**

Notes:

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

Return:

None

6.2.3.2 DMAC_Disable

Disable the DMA circuit.

Prototype:

void

DMAC_Disable(TSB_DMAL_TypeDef * **DMACx**);

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Description:

This function will disable UNIT A circuit when **DMACx** is **DMAC_UNIT_A** and Enable UNIT B circuit when **DMACx** is **DMAC_UNIT_B** and Enable UNIT C circuit when **DMACx** is **DMAC_UNIT_C**.

Return:

None

6.2.3.3 DMAC_GetINTReq

Get DMA Channel interrupt request state.

Prototype:

DMAC_INTRReq

DMAC_GetINTRReq(TSB_DMAC_TypeDef * **DMACx**);

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Description:

This function will get UNIT A interrupt request state when **DMACx** is **DMAC_UNIT_A** and get UNIT B interrupt request state when **DMACx** is **DMAC_UNIT_B** and get UNIT C interrupt request state when **DMACx** is **DMAC_UNIT_C**.

Return:

The state of interrupt request.

6.2.3.4 DMAC_GetTxINTRReq

Get the specified DMA Channel transfer interrupt request state.

Prototype:

DMAC_TxINTRReq

DMAC_GetTxINTRReq(TSB_DMAC_TypeDef * **DMACx**,
DMAC_Channel **Chx**);

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:

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

Return:

The request states of DMA transfer interrupt.

The value returned can be one of the followings:

DMAC_TX_NO_REQ means there is no transfer interrupt request,

DMAC_TX_END_REQ means there is a transfer end interrupt request,

DMAC_TX_ERR_REQ means there is a transfer error interrupt request,

DMAC_TX_REQS means there is more than one interrupt request.

6.2.3.5 DMAC_ClearTxINTReq

Clear the transfer interrupt request.

Prototype:

void

```
DMAC_ClearTxINTReq(TSB_DMAC_TypeDef * DMACx,  
                   DMAC_Channel Chx,  
                   DMAC_INTSrc INTSource);
```

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

INTSource: select the release INT source, which can be one of:

- **DMAC_INT_TX_END** for DMA transfer end interrupt,
- **DMAC_INT_TX_ERR** for DMA transfer error interrupt.

Description:

This function will clear the transfer interrupt request. When **INTSource** is

DMAC_INT_TX_END, this function will clear DMA transfer end interrupt request.

When **INTSource** is **DMAC_INT_TX_ERR**, this function will clear DMA transfer error interrupt request.

Return:

None

6.2.3.6 DMAC_GetRawTxINTReq

Get the specified DMA Channel transfer raw interrupt request state.

Prototype:

DMAC_TxINTReq

```
DMAC_GetRawTxINTReq(TSB_DMAC_TypeDef * DMACx,  
                    DMAC_Channel Chx);
```

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:

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

Return:

The request states of DMA transfer raw interrupt.

The value returned can be one of the followings:

DMAC_TX_NO_REQ means there is no transfer raw interrupt request,

DMAC_TX_END_REQ means there is a transfer end interrupt request,

DMAC_TX_ERR_REQ means there is a transfer error interrupt request,

DMAC_TX_REQS means there is more than one interrupt request.

6.2.3.7 DMAC_GetChannelTxState

Get the specified DMA Channel transfer state.

Prototype:

WorkState

```
DMAC_GetChannelTxState(TSB_DMAC_TypeDef * DMACx,  
                      DMAC_Channel Chx);
```

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:

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

Return:

The DMA transfer status.

The value returned can be one of the followings:

BUSY or **DONE**

6.2.3.8 DMACA_SetSWBurstReq

Set DMACA burst transfer requests by software.

Prototype:

void

DMACA_SetSWBurstReq(DMACA_ReqNum **BurstReq**);

Parameters:

BurstReq: Select burst request number, which can be one of:

- **DMACA_ESIO0_RX** for ESIO0 Reception,
- **DMACA_ESIO0_TX** for ESIO0 Transmission,
- **DMACA_NORMAL_UNITA_ADC** normal UNITA A/D Conversion End,
- **DMACA_SIO3_UART3_RX** for SIO3/UART3 Reception,
- **DMACA_SIO3_UART3_TX** for SIO3/UART3 Transmission,
- **DMACA_SIO0_UART0_RX** for SIO0/UART0 Reception,
- **DMACA_SIO0_UART0_TX** for SIO0/UART0 Transmission,
- **DMACA_TMRB00_CMP_MATCH** for TMRB0 compare match,
- **DMACA_TMRB04_CMP_MATCH** for TMRB4 compare match,
- **DMACA_TMRB10_CMP_MATCH** for TMRB10 compare match,
- **DMACA_TMRB14_CMP_MATCH** for TMRB14 compare match,
- **DMACA_TMRC_CMP0_MATCH** for TMRB0 compare match,

- **DMACA_TMRC_CMP1_MATCH** for TMRB1 compare match,
- **DMACA_HIGHEST_UNITA_ADC** for Top priority UNITA A/D Conversion End,
- **DMACA_PHCNT0_CMP0_MATCH** for PHCNT0 compare match,
- **DMACA_PIN** FOR DREQA PIN.

Description:

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

Return:

None

6.2.3.9 DMACB_SetSWBurstReq

Set DMACB burst transfer requests by software.

Prototype:

void

DMACB_SetSWBurstReq(DMACB_ReqNum **BurstReq**);

Parameters:

BurstReq: Select burst request number, which can be one of:

- **DMACB_ESIO1_RX** for ESIO1 Reception,
- **DMACB_ESIO1_TX** for ESIO0 Transmission,
- **DMACB_NORMAL_UNITB_ADC** normal UNITB A/D Conversion End,
- **DMACB_SIO4_UART4_RX** for SIO4/UART4 Reception,
- **DMACB_SIO4_UART4_TX** for SIO4/UART4 Transmission,
- **DMACB_SIO1_UART1_RX** for SIO1/UART1 Reception,
- **DMACB_SIO1_UART1_TX** for SIO1/UART1 Transmission,
- **DMACB_UART0_RX** for UART0 Reception,
- **DMACB_UART0_TX** for UART0 Transmission,
- **DMACB_TMRB08_CAPTURE0** for TIMERB8 capture interrupt,
- **DMACB_TMRC0_CAPTURE0** for TIMERC0 capture0 interrupt,
- **DMACB_TMRC0_CAPTURE1** for TIMERC0 capture1 interrupt,
- **DMACB_HIGHEST_UNITB_ADC** for Top priority UNITB A/D Conversion End,
- **DMACB_PCNT1_CMP0_MATCH** for PCNT1 capture0 match,
- **DMACB_TMRD0_CMP_MATCH** for TIMERD0 capture match
- **DMACB_PIN** FOR DREQB PIN,

Description:

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

Return:

None

6.2.3.10 DMACC_SetSWBurstReq

Set DMACC burst transfer requests by software.

Prototype:

void

DMACC_SetSWBurstReq(DMACC_ReqNum **BurstReq**);

Parameters:

BurstReq: Select burst request number, which can be one of:

- **DMACC_ESIO2_RX** for ESIO2 Reception,
- **DMACC_ESIO2_TX** for ESIO2 Transmission,
- **DMACC_NORMAL_UNITC_ADC** normal UNITC A/D Conversion End,
- **DMACC_SIO5_UART5_RX** for SIO5/UART5 Reception,
- **DMACC_SIO5_UART5_TX** for SIO5/UART5 Transmission,
- **DMACC_SIO2_UART2_RX** for SIO2/UART2 Reception,
- **DMACC_SIO2_UART2_TX** for SIO2/UART2 Transmission,
- **DMACC_UART1_RX** for UART1 Reception,
- **DMACC_UART1_TX** for UART1 Transmission,
- **DMACC_TMRB19_CAPTURE0** for TIMERC19 capture interrupt,
- **DMACC_TMRC0_CAPTURE2** for TIMERC0 capture2 interrupt,
- **DMACC_TMRC0_CAPTURE3** for TIMERC0 capture3 interrupt,
- **DMACC_HIGHEST_UNITC_ADC** for Top priority UNITC A/D Conversion End,
- **DMACC_PHCP_CYCLE0** for PHCP cycle0 interrupt,
- **DMACC_TMRD10_CMP_MATCH** for TIMERD10 capture match
- **DMACC_PIN** FOR DREQC PIN,

Description:

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

Return:

None

6.2.3.11 DMAC_GetSWBurstReqState

Get DMA software burst request state.

Prototype:

DMAC_BurstReqState

DMAC_GetSWBurstReqState(TSB_DMAL_TypeDef * **DMACx**);

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Description:

This function will get specified UNIT software burst request state.

Return:

The DMA burst request status.

6.2.3.12 DMACB_SetSWSingleReq

Set DMA single transfer requests by software.

Prototype:

void

DMACB_SetSWSingleReq(DMACB_ReqNum **SingleReq**);

Parameters:

SingleReq: Select burst request number, which can be:

- **DMACB_UART0_RX** for UART0 Reception,
- **DMACB_UART0_TX** for UART0 Transmission,

Description:

This function will set DMACB single transfer requests by software. Execute DMACB requests by software and hardware peripheral at the same time is prohibitive.

Return:

None

6.2.3.13 DMACC_SetSWSingleReq

Set DMA single transfer requests by software.

Prototype:

void

DMACC_SetSWSingleReq(DMACC_ReqNum **SingleReq**);

Parameters:

SingleReq: Select burst request number, which can be:

- **DMACC_UART1_RX** for UART1 Reception,
- **DMACC_UART1_TX** for UART1 Transmission,

Description:

This function will set DMACC single transfer requests by software. Execute DMACC requests by software and hardware peripheral at the same time is prohibitive.

Return:

None

6.2.3.14 DMAC_GetSWSingleReqState

Get DMA software single request state.

Prototype:

DMAC_SingleReqState

DMAC_GetSWSingleReqState(TSB_DMAL_TypeDef * **DMACx**);

Parameters:

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

- **DMAC_UNIT_B** for DMA UNIT B,
- **DMAC_UNIT_C** for DMA UNIT C.

Description:

This function will get specified UNIT software single request state.

Return:

The DMA single request status.

6.2.3.15 DMAC_SetLinkedList

Set specified DMA Channel Linked List Item Register.

Prototype:

```
void  
DMAC_SetLinkedList(TSB_DMACH_TypeDef * DMACx,  
                  DMACH_Channel Chx,  
                  uint32_t LinkedAddr);
```

Parameters:

Chx is the specified DMA channel, which can be one of:

- **DMACH_CHANNEL_0** for DMA channel 0,
- **DMACH_CHANNEL_1** for DMA channel 1.

LinkedAddr: The start address of the next transfer information.
Max 0xFFFFFFFF0.

Description:

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

Note:

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

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

The items that can be set with Linked List are configured with the following 4 words:

- 1) DMACH_SrcAddr
- 2) DMACH_DestAddr
- 3) DMACH_LLI
- 4) DMACH_Control

Return:

None

6.2.3.16 DMACH_GetFIFOState

Indicates whether data is present in the channel FIFO

Prototype:

```
WorkState  
DMACH_GetFIFOState(TSB_DMACH_TypeDef * DMACx,
```

DMAC_Channel **Chx**);

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

Description:

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

Return:

The FIFO status

The value returned can be one of the followings:

BUSY or **DONE**

6.2.3.17 DMAC_SetDMAHalt

Set whether ignore DMA request.

Prototype:

void

DMAC_SetDMAHalt(TSB_DMAL_TypeDef * **DMACx**,
DMAC_Channel **Chx**,
FunctionalState **NewState**);

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

NewState: New state of DMA halt.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will set specified UNIT Channel ignore DMA request.

Return:

None

6.2.3.18 DMAC_SetLockedTx

Set whether locked transfer.

Prototype:

```
void  
DMAC_SetLockedTx(TSB_DMACH_TypeDef * DMACx,  
                 DMACH_Channel Chx,  
                 FunctionalState NewState);
```

Parameters:

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

- **DMACH_UNIT_A** for DMA UNIT A,
- **DMACH_UNIT_B** for DMA UNIT B.
- **DMACH_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMACH_CHANNEL_0** for DMA channel 0,
- **DMACH_CHANNEL_1** for DMA channel 1.

NewState: New state of DMA transfer.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

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

Return:

None

6.2.3.19 DMAC_SetTxINTConfig

Enable or disable the specified DMA Channel transfer interrupt.

Prototype:

void

```
DMAC_SetTxINTConfig(TSB_DMACH_TypeDef * DMACx,  
                    DMACH_Channel Chx,  
                    DMACH_INTSrc INTSource,  
                    FunctionalState NewState);
```

Parameters:

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

- **DMACH_UNIT_A** for DMA UNIT A,
- **DMACH_UNIT_B** for DMA UNIT B.
- **DMACH_UNIT_C** for DMA UNIT C

Chx is the specified DMA channel, which can be one of:

- **DMACH_CHANNEL_0** for DMA channel 0,
- **DMACH_CHANNEL_1** for DMA channel 1.

INTSource: select the release INT source, which can be one of:

- **DMACH_INT_TX_END** for DMA transfer end interrupt,
- **DMACH_INT_TX_ERR** for DMA transfer error interrupt.

NewState: New states of DMA transfer interrupt.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable or disable specified UNIT Channel transfer end interrupt when **INTSource** is **DMACH_INT_TX_END** and enable or disable specified UNIT Channel transfer error interrupt when **INTSource** is **DMACH_INT_TX_ERR**.

Return:

None

6.2.3.20 DMAC_SetDMAChannel

Enable or disable the specified DMA Channel.

Prototype:

void

```
DMAC_SetDMACChannel(TSB_DMAC_TypeDef * DMACx,  
                    DMAC_Channel Chx,  
                    FunctionalState NewState);
```

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A,
- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

NewState: New state of DMA channel.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

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

Return:

None

6.2.3.21 DMAC_Init

Initialize and configure the specified DMA channel.

Prototype:

void

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

Parameters:

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

- **DMAC_UNIT_A** for DMA UNIT A.

- **DMAC_UNIT_B** for DMA UNIT B.
- **DMAC_UNIT_C** for DMA UNIT C.

Chx is the specified DMA channel, which can be one of:

- **DMAC_CHANNEL_0** for DMA channel 0,
- **DMAC_CHANNEL_1** for DMA channel 1.

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

Description:

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

Note:

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

Return:

None

6.2.4 Data Structure Description

6.2.4.1 DMAC_InitTypeDef

Data Fields:

uint32_t

TxDirection Set transfer direction, which can be set as:

- **DMAC_MEMORY_TO_MEMORY**: transfer method is memory to memory.
- **DMAC_MEMORY_TO_PERIPH**: transfer method is memory to peripheral,
- **DMAC_PERIPH_TO_MEMORY**: transfer method is peripheral to memory.
- **DMAC_PERIPH_TO_PERIPH**: transfer method is peripheral to peripheral.

uint32_t

SrcAddr Set source address.

uint32_t

DstAddr Set destination address.

FunctionalState

SrcIncrementState Specifies whether the source address is incremented or not, which can be set as:

ENABLE or **DISABLE**.

FunctionalState

DstIncrementState Specifies whether the destination address is incremented or not, which can be set as:

ENABLE or **DISABLE**.

DMAC_BitWidth

SrcBitWidth Set transfer source bit width, which can be set as:

- **DMAC_BYTE** means transfer source bit width set as byte,
- **DMAC_HALF_WORD** means transfer source bit width set as half word,
- **DMAC_WORD** means transfer source bit width set as word.

DMAC_BitWidth

DstBitWidth Set transfer destination bit width, which can be set as:

- **DMAC_BYTE** means transfer destination bit width set as byte,
- **DMAC_HALF_WORD** means transfer destination bit width set as half word,
- **DMAC_WORD** means transfer destination bit width set as word.

DMAC_BurstSize

SrcBurstSize Set transfer source burst size, which can be set as:

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

DMAC_BurstSize

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

- **DMAC_1_BEAT** means transfer destination burst size set as 1 beat.
- **DMAC_4_BEATS** means transfer destination burst size set as 4 beats.
- **DMAC_8_BEATS** means transfer destination burst size set as 8 beats.

- **DMAC_16_BEATS** means transfer destination burst size set as 16 beats.
- **DMAC_32_BEATS** means transfer destination burst size set as 32 beats.
- **DMAC_64_BEATS** means transfer destination burst size set as 64 beats.
- **DMAC_128_BEATS** means transfer destination burst size set as 128 beats.
- **DMAC_256_BEATS** means transfer destination burst size set as 256 beats.

uint32_t

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

DMACA_ReqNum

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

- **DMACA_ESIO0_RX** for ESIO0 Reception.
- **DMACA_ESIO0_TX** for ESIO0 Transmission.
- **DMACA_NORMAL_UNITA_ADC** normal UNITA A/D Conversion End.
- **DMACA_SIO3_UART3_RX** for SIO3/UART3 Reception.
- **DMACA_SIO3_UART3_TX** for SIO3/UART3 Transmission.
- **DMACA_SIO0_UART0_RX** for SIO0/UART0 Reception.
- **DMACA_SIO0_UART0_TX** for SIO0/UART0 Transmission.
- **DMACA_TMRB00_CMP_MATCH** for TMRB0 compare match.
- **DMACA_TMRB04_CMP_MATCH** for TMRB4 compare match.
- **DMACA_TMRB10_CMP_MATCH** for TMRB10 compare match.
- **DMACA_TMRB14_CMP_MATCH** for TMRB14 compare match.
- **DMACA_TMRC_CMP0_MATCH** for TMRB0 compare match.
- **DMACA_TMRC_CMP1_MATCH** for TMRB1 compare match.
- **DMACA_HIGHEST_UNITA_ADC** for Top priority UNITA A/D Conversion End.
- **DMACA_PHCNT0_CMP0_MATCH** for PHCNT0 compare match.
- **DMACA_PIN** FOR DREQA PIN.

DMACA_ReqNum

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

- **DMACA_ESIO0_RX** for ESIO0 Reception.
- **DMACA_ESIO0_TX** for ESIO0 Transmission.
- **DMACA_NORMAL_UNITA_ADC** normal UNITA A/D Conversion End.
- **DMACA_SIO3_UART3_RX** for SIO3/UART3 Reception.
- **DMACA_SIO3_UART3_TX** for SIO3/UART3 Transmission.
- **DMACA_SIO0_UART0_RX** for SIO0/UART0 Reception.
- **DMACA_SIO0_UART0_TX** for SIO0/UART0 Transmission.
- **DMACA_TMRB00_CMP_MATCH** for TMRB0 compare match.
- **DMACA_TMRB04_CMP_MATCH** for TMRB4 compare match.
- **DMACA_TMRB10_CMP_MATCH** for TMRB10 compare match.
- **DMACA_TMRB14_CMP_MATCH** for TMRB14 compare match.
- **DMACA_TMRC_CMP0_MATCH** for TMRB0 compare match.
- **DMACA_TMRC_CMP1_MATCH** for TMRB1 compare match.

- **DMACA_HIGHEST_UNITA_ADC** for Top priority UNITA A/D Conversion End.
- **DMACA_PHCNT0_CMP0_MATCH** for PHCNT0 compare match.
- **DMACA_PIN** FOR DREQA PIN.

DMACB_ReqNum

B_TxDstPeriph Set transfer destination peripheral, which can be set as:

- **DMACB_ESIO1_RX** for ESIO1 Reception.
- **DMACB_ESIO1_TX** for ESIO0 Transmission.
- **DMACB_NORMAL_UNITB_ADC** normal UNITB A/D Conversion End.
- **DMACB_SIO4_UART4_RX** for SIO4/UART4 Reception.
- **DMACB_SIO4_UART4_TX** for SIO4/UART4 Transmission.
- **DMACB_SIO1_UART1_RX** for SIO1/UART1 Reception.
- **DMACB_SIO1_UART1_TX** for SIO1/UART1 Transmission.
- **DMACB_UART0_RX** for UART0 Reception.
- **DMACB_UART0_TX** for UART0 Transmission.
- **DMACB_TMRB08_CAPTURE0** for TIMERB8 capture interrupt.
- **DMACB_TMRC0_CAPTURE0** for TIMERC0 capture0 interrupt.
- **DMACB_TMRC0_CAPTURE1** for TIMERC0 capture1 interrupt.
- **DMACB_HIGHEST_UNITB_ADC** for Top priority UNITB A/D Conversion End.
- **DMACB_PCNT1_CMP0_MATCH** for PCNT1 capture0 match.
- **DMACB_TMRD0_CMP_MATCH** for TIMERD0 capture match.
- **DMACB_PIN** FOR DREQB PIN.

DMACB_ReqNum

B_TxSrcPeriph Set transfer source peripheral, which can be set as:

- **DMACB_ESIO1_RX** for ESIO1 Reception.
- **DMACB_ESIO1_TX** for ESIO0 Transmission.
- **DMACB_NORMAL_UNITB_ADC** normal UNITB A/D Conversion End.
- **DMACB_SIO4_UART4_RX** for SIO4/UART4 Reception.
- **DMACB_SIO4_UART4_TX** for SIO4/UART4 Transmission.
- **DMACB_SIO1_UART1_RX** for SIO1/UART1 Reception.
- **DMACB_SIO1_UART1_TX** for SIO1/UART1 Transmission.
- **DMACB_UART0_RX** for UART0 Reception.
- **DMACB_UART0_TX** for UART0 Transmission.
- **DMACB_TMRB08_CAPTURE0** for TIMERB8 capture interrupt.
- **DMACB_TMRC0_CAPTURE0** for TIMERC0 capture0 interrupt.
- **DMACB_TMRC0_CAPTURE1** for TIMERC0 capture1 interrupt.
- **DMACB_HIGHEST_UNITB_ADC** for Top priority UNITB A/D Conversion End.
- **DMACB_PCNT1_CMP0_MATCH** for PCNT1 capture0 match.
- **DMACB_TMRD0_CMP_MATCH** for TIMERD0 capture match.
- **DMACB_PIN** FOR DREQB PIN.

DMAACC_ReqNum

C_TxDstPeriph Set transfer destination peripheral, which can be set as:

- **DMACC_ESIO2_RX** for ESIO2 Reception.
- **DMACC_ESIO2_TX** for ESIO2 Transmission.
- **DMACC_NORMAL_UNITC_ADC** normal UNITC A/D Conversion End.
- **DMACC_SIO5_UART5_RX** for SIO5/UART5 Reception.
- **DMACC_SIO5_UART5_TX** for SIO5/UART5 Transmission.
- **DMACC_SIO2_UART2_RX** for SIO2/UART2 Reception.
- **DMACC_SIO2_UART2_TX** for SIO2/UART2 Transmission.
- **DMACC_UART1_RX** for UART1 Reception.
- **DMACC_UART1_TX** for UART1 Transmission.
- **DMACC_TMRB19_CAPTURE0** for TIMERC19 capture interrupt.
- **DMACC_TMRC0_CAPTURE2** for TIMERC0 capture2 interrupt.
- **DMACC_TMRC0_CAPTURE3** for TIMERC0 capture3 interrupt.
- **DMACC_HIGHEST_UNITC_ADC** for Top priority UNITC A/D Conversion End.
- **DMACC_PHCP_CYCLE0** for PHCP cycle0 interrupt.
- **DMACC_TMRD10_CMP_MATCH** for TIMERD10 capture match.
- **DMACC_PIN** FOR DREQC PIN.

DMACC_ReqNum

C_TxSrcPeriph Set transfer source peripheral, which can be set as:

- **DMACC_ESIO2_RX** for ESIO2 Reception.
- **DMACC_ESIO2_TX** for ESIO2 Transmission.
- **DMACC_NORMAL_UNITC_ADC** normal UNITC A/D Conversion End.
- **DMACC_SIO5_UART5_RX** for SIO5/UART5 Reception.
- **DMACC_SIO5_UART5_TX** for SIO5/UART5 Transmission.
- **DMACC_SIO2_UART2_RX** for SIO2/UART2 Reception.
- **DMACC_SIO2_UART2_TX** for SIO2/UART2 Transmission.
- **DMACC_UART1_RX** for UART1 Reception.
- **DMACC_UART1_TX** for UART1 Transmission.
- **DMACC_TMRB19_CAPTURE0** for TIMERC19 capture interrupt.
- **DMACC_TMRC0_CAPTURE2** for TIMERC0 capture2 interrupt.
- **DMACC_TMRC0_CAPTURE3** for TIMERC0 capture3 interrupt.
- **DMACC_HIGHEST_UNITC_ADC** for Top priority UNITC A/D Conversion End.
- **DMACC_PHCP_CYCLE0** for PHCP cycle0 interrupt.
- **DMACC_TMRD10_CMP_MATCH** for TIMERD10 capture match.
- **DMACC_PIN** FOR DREQC PIN.

FunctionalState

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

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

7. EPHC

7.1 Overview

TOSHIBA TPM440 has one channel enhanced two-phase pulse input counters. The 2-phase pulse input counter is an up-counter that increments/decrements depending on the combination of 2-phase input pulses or 1-phase input pulses. A cycle phase measurement is the function that measures the cycle of 2-phase input pulse and the phase differences between edges.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_ephc.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_ephc.h containing the macros, data types, structures and API definitions for use by applications.

7.2 API Functions

7.2.1 Function List

- ◆ void EPHC_Enable(TSB_EPHC_TypeDef * **EPHCx**);
- ◆ void EPHC_Disable(TSB_EPHC_TypeDef * **EPHCx**);
- ◆ void EPHC_Init(TSB_EPHC_TypeDef * **EPHCx**, EPHC_InitTypeDef * **InitStruct**);
- ◆ void EPHC_EnableInterrupt(TSB_EPHC_TypeDef * **EPHCx**, uint32_t **EnableINT**);
- ◆ void EPHC_DisableInterrupt(TSB_EPHC_TypeDef * **EPHCx**, uint32_t **DisableINT**);
- ◆ EPHC_INTFactor EPHC_GetINTFactor(TSB_EPHC_TypeDef * **EPHCx**);
- ◆ void EPHC_ClearINTFactor(TSB_EPHC_TypeDef * **EPHCx**, uint32_t **ClearINT**);
- ◆ void EPHC_ASetRunState(TSB_EPHC_TypeDef * **EPHCx**, uint8_t **Cmd**);
- ◆ void EPHC_AClearPulseCntValue(TSB_EPHC_TypeDef * **EPHCx**);
- ◆ uint16_t EPHC_AGetCompareValue(TSB_EPHC_TypeDef * **EPHCx**, uint8_t **CmpReg**);
- ◆ void EPHC_ASetCompareValue(TSB_EPHC_TypeDef * **EPHCx**, uint8_t **CmpReg**, uint16_t **CmpValue**);
- ◆ uint16_t EPHC_AGetPulseCntValue(TSB_EPHC_TypeDef * **EPHCx**);
- ◆ void EPHC_BSetRunState(TSB_EPHC_TypeDef * **EPHCx**, uint8_t **Cmd**);
- ◆ void EPHC_BSetDMAReq(TSB_EPHC_TypeDef * **EPHCx**, FunctionalState **NewState**, uint8_t **DMAReq**);
- ◆ uint32_t EPHC_BGetReadCntValue(TSB_EPHC_TypeDef * **EPHCx**);
- ◆ uint32_t EPHC_BGetCapRegValue(TSB_EPHC_TypeDef * **EPHCx**, uint8_t **CapReg**);
- ◆ uint32_t EPHC_BGetCapRegOverflow(TSB_EPHC_TypeDef * **EPHCx**, uint8_t **CapReg**);
- ◆ uint32_t EPHC_BGetCycleCntRegValue(TSB_EPHC_TypeDef * **EPHCx**, uint8_t

◆ *CntReg*);
uint32_t EPHC_BGetPhaseDiffRegValue(TSB_EPHC_TypeDef * **EPHCx**, uint8_t
PhaseReg);

7.2.2 Detailed Description

Functions listed above can be divided into three parts:

- 1) Configure and control the common functions of each EPHC channel are handled by the EPHC_Enable (),EPHC_Disable (),EPHC_Init(), EPHC_ASetRunState(),and EPHC_BSetRunState()..
- 2) The status indication of each EPHC channel is handled by EPHC_GetINTFactor(), EPHC_AGetPulseCntValue(), EPHC_AGetCompareValue(),EPHC_BgetReadCntValue(),EPHC_BGetCapRegValue(),EPHC_BgetCapRegOverflow(),EPHC_BgetCycleCntRegValue(),EPHC_BgetPhaseDiffReg Value().
- 3) EPHC_ClearINTFactor(), EPHC_EnableInterrupt(), EPHC_DisableInterrupt(), , EPHC_AClearPulseCntValue(),EPHC_ASetCompareValue() and EPHC_BSetDMAReq() handle other specified functions.

7.2.3 Function Documentation

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

TSB_EPHC

7.2.3.1 EPHC_Enable

Enable the specified EPHC channel.

Prototype:

void
EPHC_Enable(TSB_EPHC_TypeDef* **EPHCx**)

Parameters:

EPHCx is the specified EPHC channel.

Description:

This function enables the specified EPHC channel selected by **EPHCx**.

Return:

None

7.2.3.2 EPHC_Disable

Disable the specified EPHC channel.

Prototype:

void
EPHC_Disable(TSB_EPHC_TypeDef* **EPHCx**)

Parameters:

EPHCx is the specified EPHC channel.

Description:

This function disables the specified EPHC channel selected by **EPHCx**.

Return:

None

7.2.3.3 EPHC_ASetRunState

Start or stop of the specified EPHC channel.

Prototype:

void
EPHC_ASetRunState(TSB_EPHC_TypeDef * **EPHCx**,
uint32_t **Cmd**);

Parameters:

EPHCx is the specified EPHC channel.

Cmd The command for the up-and-down counter

- **EPHC_RUN** for start counter
- **EPHC_STOP** for stop counter

Description:

The up-and-down counter of the specified EPHC channel starts counting if **Cmd** is **EPHC_RUN** and up-and-down counter stops counting and the value in up-and-down counter register is clear if **Cmd** is **EPHC_STOP**.

Return:

None

7.2.3.4 EPHC_Init

Initialize the specified EPHC channel.

Prototype:

```
void  
EPHC_Init (TSB_EPHC_TypeDef * EPHCx,  
           EPHC_InitTypeDef * InitStruct);
```

Parameters:

EPHCx is the specified EPHC channel.

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

Description:

This function initializes the specified EPHC channel selected by **EPHCx**.

Return:

None

7.2.3.5 EPHC_GetINTFactor

Indicate what causes the interrupt.

Prototype:

```
EPHC_INTFactor  
EPHC_GetINTFactor(TSB_EPHC_TypeDef* EPHCx)
```

Parameters:

EPHCx is the specified EPHC channel.

Description:

This function should be used in ISR to indicate the factor of interrupt.

Return:

EPHC Interrupt factor. Each bit has the following meaning:

Compare0 (Bit0): Specified EPHC channel detected a match with the compare register0

Compare1 (Bit1): Specified EPHC channel detected a match with the compare register1

Overflow (Bit2): Specified EPHC channel detected counter is overflow

Underflow (Bit3): Specified EPHC channel detected counter is underflow

IN0Falling(Bit4): Specified EPHC channel detected EPHCxIN0 on falling edge

IN1Falling(Bit5): Specified EPHC channel detected EPHCxIN1 on falling edge

IN0Rising(Bit6): Specified EPHC channel detected EPHCxIN0 on rising edge

IN1Rising(Bit7): Specified EPHC channel detected EPHCxIN1 on rising edge

CycleMeasurement (Bit8): Specified EPHC channel detected Cycle error in 2-phase pulse cycle measurement

7.2.3.6 EPHC_ClearINTFactor

Clear specified interrupt factor flag.

Prototype:

void

EPHC_ClearINTFactor(TSB_EPHC_TypeDef * **EPHCx**,
uint32_t **ClearINT**)

Parameters:

EPHCx is the specified EPHC channel.

ClearINT Select the clear of EPHC interrupt factor. This parameter can be:

- **EPHC_FLG_CMP0F**: Clears the interrupt factor which monitors compare register0 match event.
- **EPHC_FLG_CMP1F**: Clears the interrupt factor which monitors compare register1 match event.
- **EPHC_FLG_OVFF**: Clears the interrupt factor which monitors overflow event.
- **EPHC_FLG_UBFF**: Clears the interrupt factor which monitors underflow event.
- **EPHC_FLG_SB0F**: Clears the interrupt factor which monitors EPHCxIN0 on falling edge.
- **EPHC_FLG_SB1F**: Clears the interrupt factor which monitors EPHCxIN1 on falling edge.
- **EPHC_FLG_SB2F**: Clears the interrupt factor which monitors EPHCxIN0 on rising edge
- **EPHC_FLG_SB3F**: Clears the interrupt factor which monitors EPHCxIN1 on rising edge
- **EPHC_FLG_DIRF**: Clears the interrupt factor which monitors Cycle error in 2-phase pulse cycle measurement
- **EPHC_FLG_ALL**: All interrupt factors can be cleared.
- Combination of effective interrupt factor.

Description:

Because of each interrupt factor is not cleared automatically.

This function is used to clear the specified interrupt factor.

Return:

None

Note:

Each interrupt factor is not cleared automatically, initialize before using them.

7.2.3.7 EPHC_EnableInterrupt

Enable specified EPHC interrupt

Prototype:

void

```
EPHC_EnableInterrupt(TSB_EPHC_TypeDef * EPHCx,  
                    uint32_t EnableINT);
```

Parameters:

EPHCx is the specified EPHC channel.

EnableINT Selects the clear of EPHC interrupt. This parameter can be:

- **EPHC_IE_INT_PCCP0**: Enable INTPHTx0 interrupt, which occurred if the counter matches with compare register0.
- **EPHC_IE_INT_PCCP1**: Enable INTPHTx1 interrupt, which occurred if the counter matches with compare register1.
- **EPHC_IE_INT_PCOVF**: Enable 16-bit counter overflow interrupt
- **EPHC_IE_INT_PCUDF**: Enable 16-bit counter underflow interrupt.
- **EPHC_IE_INT_PCDT0**: Enable interrupt 2-phase pulse cycle phase measurement cycle 0.
- **EPHC_IE_INT_PCDT1**: Enable interrupt 2-phase pulse cycle phase measurement cycle 1.
- **EPHC_IE_INT_PCDT2**: Enable interrupt 2-phase pulse cycle phase measurement cycle 2.
- **EPHC_IE_INT_PCDT3**: Enable interrupt 2-phase pulse cycle phase measurement cycle 3.
- **EPHC_IE_INT_PCDIR**: Enable interrupt 2-phase pulse cycle phase measurement cycle error interrupt
- **EPHC_IE_INT_PCUOVF**: Enable interrupt 2-phase pulse cycle phase measurement overflow interrupt
- **EPHC_IE_INT_ALL**: All interrupt can be enabled
- Combination of effective EPHC interrupt.

Description:

Enable specified EPHC interrupt.

Return:

None

7.2.3.8 EPHC_DisableInterrupt

Disable specified EPHC interrupt.

Prototype:

```
void  
EPHC_DisableInterrupt(TSB_EPHC_TypeDef * EPHCx,  
uint32_t DisableINT);
```

Parameters:

EPHCx is the specified EPHC channel.

DisableINT Select the clear of EPHC interrupt. This parameter can be:

- **EPHC_IE_INT_PCCP0**: Disable INTPHTx0 interrupt, which occurred if the counter matches with compare register0.
- **EPHC_IE_INT_PCCP1**: Disable INTPHTx1 interrupt, which occurred if the counter matches with compare register1.
- **EPHC_IE_INT_PCOVF**: Disable 16-bit counter overflow interrupt
- **EPHC_IE_INT_PCUDF**: Disable 16-bit counter underflow interrupt.
- **EPHC_IE_INT_PCDT0**: Disable interrupt 2-phase pulse cycle phase measurement cycle 0.
- **EPHC_IE_INT_PCDT1**: Disable interrupt 2-phase pulse cycle phase measurement cycle 1.
- **EPHC_IE_INT_PCDT2**: Disable interrupt 2-phase pulse cycle phase measurement cycle 2.
- **EPHC_IE_INT_PCDT3**: Disable interrupt 2-phase pulse cycle phase measurement cycle 3.
- **EPHC_IE_INT_PCDIR**: Disable interrupt 2-phase pulse cycle phase measurement cycle error interrupt
- **EPHC_IE_INT_PCUOVF**: Disable interrupt 2-phase pulse cycle phase measurement overflow interrupt
- **EPHC_IE_INT_ALL**: All interrupt can be enabled
- Combination of effective EPHC interrupt.

Description:

Disable specified EPHC interrupt.

Return:

None

7.2.3.9 EPHC_AGetPulseCntValue

Get the counter value of specified EPHC channel

Prototype:

uint16_t

EPHC_AGetPulseCntValue(TSB_EPHC_TypeDef * **EPHCx**);

Parameters:

EPHCx is the specified EPHC channel.

Description:

This function returns the value in counter of the specified EPHC channel

Return:

The value of EPHC counter

7.2.3.10 EPHC_AClearPulseCntValue

Clear the counter value of specified EPHC channel

Prototype:

void

EPHC_AClearPulseCntValue(TSB_EPHC_TypeDef * **EPHCx**);

Parameters:

EPHCx is the specified EPHC channel.

Description:

This function clears the value in counter of the specified EPHC channel

Return:

None

7.2.3.11 EPHC_AGetCompareValue

Get the value of compare register0 or compare register1 of the specified EPHC channel.

Prototype:

uint16_t

EPHC_AGetCompareValue(TSB_EPHC_TypeDef * **EPHCx**,
uint8_t **CmpReg**)

Parameters:

EPHCx is the specified EPHC channel.

CmpReg is used to choose to return the value of compare register0 or to return the value of compare register1, which can be one of the following,

- **EPHC_COMP_0**: specifying compare register0.
- **EPHC_COMP_1**: specifying compare register1.

Description:

This function returns the value of compare register0 of the specified EPHC channel if **CmpReg** is **EPHC_COMP_0**, and returns value of compare register1 of the specified EPHC channel if **CmpReg** is **EPHC_COMP_1**

Return:

The compare value

7.2.3.12 EPHC_ASetCompareValue

Set the value of compare register0 or compare register1 of the specified EPHC channel.

Prototype:

```
void  
EPHC_SetCompareValue(TSB_EPHC_TypeDef * EPHCx,  
                     uint8_t CmpReg,  
                     uint16_t CmpValue);
```

Parameters:

EPHCx is the specified EPHC channel.

CmpReg is used to choose to set the value of compare register0 or to set the value of compare register1, which can be one of the following,

- **EPHC_COMP_0**: specifying compare register0.
- **EPHC_COMP_1**: specifying compare register1.

CmpValue is the value to set to compare register

Description:

This function sets **CmpValue** to compare register0 of the specified EPHC channel if **CmpReg** is **EPHC_COMP_0**, and sets **CmpValue** to compare register1 of the specified EPHC channel if **CmpReg** is **EPHC_COMP_1**

Return:

None

7.2.3.13 EPHC_BSetRunState

Start or stop 24-bit counter of the specified EPHC channel.

Prototype:

void

```
EPHC_BSetRunState(TSB_EPHC_TypeDef * EPHCx,  
                  uint32_t Cmd);
```

Parameters:

EPHCx is the specified EPHC channel.

Cmd The command for the up-and-down counter

- **EPHC_RUN** for start counter
- **EPHC_STOP** for stop counter

Description:

The up-and-down 24-bit counter of the specified EPHC channel starts counting if **Cmd** is **EPHC_RUN** and up-and-down counter stops counting and the value in up-and-down counter register is clear if **Cmd** is **EPHC_STOP**.

Return:

None

7.2.3.14 EPHC_BSetDMAReq

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

Prototype:

void

```
EPHC_SetDMAReq(TSB_EPHC_TypeDef * EPHCx,  
               FunctionalState NewState,  
               uint8_t DMAReq);
```

Parameters:

EPHCx is the specified EPHC channel.

NewState specifies specified DMA monitor state of specified EPHC channel, which can be:

- **ENABLE**: enables specified DMA request
- **DISABLE**: disables specified DMA request

DMAReq specifies DMA request of the external inputs, which can be

- **EPHC_DMA_REQ_CAPTURE_0**: Select DMA request: input capture0.

Description:

This function enables or disables the selected DMA request for the specified EPHC channel.

Return:

None

Note:

When interrupt request is disabled, DMA request is not occurred even if DMA request is enabled.

7.2.3.15 EPHC_BGetReadCntValue

Get the 24-bit counter value of specified EPHC channel

Prototype:

Uint32_t

EPHC_BGetReadCntValue (TSB_EPHC_TypeDef * **EPHCx**);

Parameters:

EPHCx is the specified EPHC channel.

Description:

This function returns the value in 24-bit counter of the specified EPHC channel

Return:

The value of EPHC 24-bit counter is enabled.

7.2.3.16 EPHC_BGetCapRegValue

Get the capture register value of specified EPHC channel

Prototype:

Uint32_t

EPHC_BGetCapRegValue (TSB_EPHC_TypeDef * **EPHCx**,
uint8_t CapReg)

Parameters:

EPHCx is the specified EPHC channel.

CapReg is the Capture register number.

which can be:

- **EPHC_BCAP00_IN0RISING**: Captures a value of EPHCxIN0 on rising edge.

- **EPHC_BCAP10_IN1RISING** Captures a value of EPHCxIN1 on rising edge.
- **EPHC_BCAP30_IN0FALLING** Captures a value of EPHCxIN0 on falling edge.
- **EPHC_BCAP30_IN1FALLING** Captures a value of EPHCxIN0 on falling edge.

Description:

Get the capture register value of specified EPHC channel

Return:

Capture register value.

7.2.3.17 EPHC_BGetCapRegOverflow

Get the capture register value of specified EPHC channel

Prototype:

UInt32_t

EPHC_BGetCapRegOverflow (TSB_EPHC_TypeDef * **EPHCx**,
uint8_t CapReg)

Parameters:

EPHCx is the specified EPHC channel.

CapReg is the Capture register number.

which can be:

- **EPHC_BCAP00_IN0RISING**: Captures a value of EPHCxIN0 on rising edge.
- **EPHC_BCAP10_IN1RISING** Captures a value of EPHCxIN1 on rising edge.
- **EPHC_BCAP30_IN0FALLING** Captures a value of EPHCxIN0 on falling edge.
- **EPHC_BCAP30_IN1FALLING** Captures a value of EPHCxIN0 on falling edge.

Description:

Get the capture register found overflow occurs or not.

Return:

Capture register value.

7.2.3.18 EPHC_BGetCycleCntRegValue

Captures a value of cycle counter of specified EPHC channel..

Prototype:

UInt32_t

EPHC_BGetCapRegValue (TSB_EPHC_TypeDef * EPHCx,
uint8_t CntReg)

Parameters:

EPHCx is the specified EPHC channel.

CntReg is the Capture register number.

which can be:

- **EPHC_B0DAT_IN0RISING:** Captures a value of cycle counter between EPHCxIN0 on rising edges
- **EPHC_B1DAT_IN1RISING:** Captures a value of cycle counter between EPHCxIN1 on rising edge.
- **EPHC_B2DAT_IN0FALLING:** Captures a value of cycle counter between EPHCxIN0 on falling edge.
- **EPHC_B3DAT_IN1FALLING:** Captures a value of cycle counter between EPHCxIN0 on falling edge.
- **EPHC_BCDAT_COMMON:** Captures a value of cycle counter at the same timing of EPHCxB0DAT to EPHCxB3DAT

Description:

Captures a value of cycle counter.

Return:

The Value of cycle counter.

7.2.3.19 EPHC_BGetPhaseDiffRegValue

Captures a value of cycle counter of specified EPHC channel.

Prototype:

Uint32_t

EPHC_BGetPhaseDiffRegValue (TSB_EPHC_TypeDef * **EPHCx**,
uint8_t PhaseReg)

Parameters:

EPHCx is the specified EPHC channel.

CntReg is the Capture register number.

which can be:

- **EPHC_B0PDT:** EPHCx Phase Difference 0 Register.
- **EPHC_B1PDT:** EPHCx Phase Difference 1 Register.
- **EPHC_B2PDT:** EPHCx Phase Difference 2 Register.
- **EPHC_B3PDT:** EPHCx Phase Difference 3 Register.

Description:

Captures a value of cycle counter.

Return:

Value of cycle counter

7.2.4 Data Structure Description

7.2.4.1 EPHC_InitTypeDef

Data Fields:

uint32_t

CountDownEdgeSelForP1 Count down edge selection, which can be set as:

- **EPHC_CNT_MA1DN_NOCNTDOWN0:** No count up if EPHCxIN0 and EPHCxIN1 are modified.
- **EPHC_CNT_MA1DN_IN0RISING:** EPHCxIN0 on rising edge.
- **EPHC_CNT_MA1DN_IN1RISING:** EPHCxIN1 on rising edge.
- **EPHC_CNT_MA1DN_IN0FALLING:** EPHCxIN0 on falling edge.
- **EPHC_CNT_MA1DN_IN1FALLING:** EPHCxIN1 on falling edge.
- **EPHC_CNT_MA1DN_IN0BOTH:** EPHCxIN0 on both edges.
- **EPHC_CNT_MA1DN_IN1BOTH:** EPHCxIN0 on both edges.
- **EPHC_CNT_MA1DN_NOCNTDOWN7:** No count down if EPHCxIN0 and EPHCxIN1 are modified.

uint32_t

CountUpEdgeSelForP1 Count up edge selection, which can be set as:

- **EPHC_CNT_MA1UP_NOCNTUP0:** No count up if EPHCxIN0 and EPHCxIN1 are modified.
- **EPHC_CNT_MA1UP_IN0RISING:** EPHCxIN0 on rising edge.
- **EPHC_CNT_MA1UP_IN1RISING:** EPHCxIN1 on rising edge.
- **EPHC_CNT_MA1UP_IN0FALLING:** EPHCxIN0 on falling edge.
- **EPHC_CNT_MA1UP_IN1FALLING:** EPHCxIN1 on falling edge.
- **EPHC_CNT_MA1UP_IN0BOTH:** EPHCxIN0 on both edges.
- **EPHC_CNT_MA1UP_IN1BOTH:** EPHCxIN0 on both edges.
- **EPHC_CNT_MA1UP_NOCNTUP7:** No count down if EPHCxIN0 and EPHCxIN1 are modified.

uint32_t

InputClkSelection Input clock selection for 24-bit counter, which can be set as:

- **EPHC_CNT_BRCK_FC:** Input clock is fc.
- **EPHC_CNT_BRCK_FC_2:** Input clock is fc/2.
- **EPHC_CNT_BRCK_FC_4:** Input clock is fc/4.
- **EPHC_CNT_BRCK_FC_8:** Input clock is fc/8.

uint32_t

PhaseSelection Phase selection, which can be set as:

- **EPHC_CNT_PBDIR_POSITIVEPHASE:** Positive phase.
- **EPHC_CNT_PBDIR_NEGATIVEPHASE:** Negative phase.

uint32_t

ModeSetting Mode setting in 2-phase pulse counter function, which can be set as:

- **EPHC_CNT_MA12_PULSE_2**: 2-pulse counter mode.
- **EPHC_CNT_MA12_PULSE_1**: 1-pulse counter mode.

uint32_t

DirectionSetting Direction setting, which can be set as:

- **EPHC_CNT_MA2DIR_POSITIVEDIR**: Positive direction
- **EPHC_CNT_MA2DIR_NEGATIVEDIR**: Negative direction

uint32_t

NoiseFilterCtrl enables and disables the noise filter, which can be set as:

- **EPHC_CNT_NOISEFILTER_NONE**: No noise filter
- **EPHC_CNT_NOISEFILTER_2**: Filtered a signal of 2/fsys or less as noise.
- **EPHC_CNT_NOISEFILTER_4**: Filtered a signal of 4/fsys or less as noise.

uint32_t

CountClearCtrl clears or do nothing with the EPHC up-and-down counter, which can be set as:

- **EPHC_COUNT_CONTINUE**: Do nothing with the EPHC up-and-down counter.
- **EPHC_COUNT_CLR**: Clears the EPHC up-and-down counter

7.2.4.2 EPHC_INTFactor

Data Fields:

uint32_t

All: EPHC interrupt factor.

Bit

uint32_t

Compare0: 1 a match with the compare register0 is detected

uint32_t

Compare1: 1 a match with the compare register1 is detected

uint32_t

Overflow: 1 an up-and-down counter is overflow

uint32_t

UnderFlow: 1 an up-and-down counter is underflow

uint32_t

IN0Falling: 1 EPHCxIN0 on falling edge

uint32_t

IN1Falling: 1 EPHCxIN1 on falling edge

uint32_t

IN0Rising: 1 EPHCxIN0 on rising edge

uint32_t

IN0Rising: 1 EPHCxIN1 on rising edge

uint32_t

CycleMeasurement: 1 Cycle error in 2-phase pulse cycle measurement

uint32_t

Reserverd : 23 Reserverd

8. ESIO

8.1 Overview

TOSHIBA TPM440 has 3 channels Enhanced Serial I/O (ESIO). They can transmit and receive data in full duplex mode. It has SPI mode and SIO mode. It can speedily receive data with various peripheral devices.

ESIO has chip select signal pins (ESIOxCS0 or ESIOxCS1), serial clock signal pin (ESIOxSCK) and serial transmit and receive data signal pins (ESIOxTXD0 to 3 and ESIOxRXD0 to 3).

ESIO has four transmit data signal pins and four receive data signal pins. One transmit data signal pin and one receive data signal pin are used as one pair (The pair called for LINE). The number of pairs can be specified from one pair to four pairs. So, transfer bit rate can be four times of serial clock frequency.

The length of frame can be specified by one bit from 8-bit to 32-bit.

8.2 API Functions

8.2.1 Function List

- ◆ void ESIO_SWReset(TSB_ESIO_TypeDef * ESIOx);
- ◆ void ESIO_Enable(TSB_ESIO_TypeDef * ESIOx);
- ◆ void ESIO_Disable(TSB_ESIO_TypeDef * ESIOx);
- ◆ void ESIO_SetTxRxCtrl(TSB_ESIO_TypeDef * ESIOx, FunctionalState NewState);
- ◆ FunctionalState ESIO_GetTxRxCtrl(TSB_ESIO_TypeDef * ESIOx);
- ◆ void ESIO_SelectMode(TSB_ESIO_TypeDef * ESIOx, ESIO_Mode Mode);
- ◆ void ESIO_SelectTransferMode(TSB_ESIO_TypeDef * ESIOx, ESIO_TransferMode TrMode);
- ◆ void ESIO_SetCS(TSB_ESIO_TypeDef * ESIOx, ESIO_CSx CSx);
- ◆ void ESIO_SetTransferNum(TSB_ESIO_TypeDef * ESIOx, uint8_t Num);
- ◆ void ESIO_SetTxPinInIdle(TSB_ESIO_TypeDef * ESIOx, ESIO_TxPinInIdle PinMode);
- ◆ void ESIO_SetFIFOLevelINT(TSB_ESIO_TypeDef * ESIOx, uint8_t TxRx, uint8_t Level);
- ◆ void ESIO_SetDMA(TSB_ESIO_TypeDef * ESIOx, uint8_t TxRx, FunctionalState NewState);
- ◆ void ESIO_SetINT(TSB_ESIO_TypeDef * ESIOx, uint32_t IntSrc, FunctionalState NewState);
- ◆ void ESIO_InitFIFO(TSB_ESIO_TypeDef * ESIOx, uint8_t TxRx);
- ◆ void ESIO_SetBaudRate(TSB_ESIO_TypeDef * ESIOx, ESIO_BaudClock Clk, uint8_t Divider);

- ◆ void ESIO_Init(TSB_ESIO_TypeDef * ESIOx, ESIO_InitTypeDef * Init);
- ◆ void ESIO_SetTxData(TSB_ESIO_TypeDef * ESIOx, uint32_t dat);
- ◆ uint32_t ESIO_GetRxData(TSB_ESIO_TypeDef * ESIOx);
- ◆ FunctionalState ESIO_IsRegisterModifiable(TSB_ESIO_TypeDef * ESIOx);
- ◆ ESIO_StatusFlag ESIO_GetStatus(TSB_ESIO_TypeDef * ESIOx);
- ◆ ESIO_ParityErrNum ESIO_GetParityErrorFlag(TSB_ESIO_TypeDef * ESIOx);
- ◆ void ESIO_ClrAllParityErrFlag(TSB_ESIO_TypeDef * ESIOx);
- ◆ uint32_t ESIO_GetHorizontalParityError(TSB_ESIO_TypeDef * ESIOx, ESIO_LINEx Line);
- ◆ void ESIO_ClrHorizontalParityError(TSB_ESIO_TypeDef * ESIOx, ESIO_LINEx Line);
- ◆ ESIO_VerticalParityErrFrame ESIO_GetVerticalParityErrFrame(TSB_ESIO_TypeDef * ESIOx);
- ◆ void ESIO_ClrAllVerticalParityErrNum(TSB_ESIO_TypeDef * ESIOx);

8.2.2 Detailed Description

Functions listed above can be divided into five parts:

- 1) Configure and control the common functions of each ESIO channel:
ESIO_SWReset(), ESIO_Enable(), ESIO_Disable(),
ESIO_SelectMode(), ESIO_SelectTransferMode(),
ESIO_SetTxPinInIdle(), ESIO_Init(), ESIO_SetTransferNum(),
ESIO_SetBaudRate().
- 2) Configure FIFO and DMA:
ESIO_SetFIFOLevelINT(), ESIO_SetDMA(), ESIO_InitFIFO().
- 3) Transfer control:
ESIO_SetTxData(), ESIO_GetRxData(), ESIO_SetTxRxCtrl(), ESIO_SetCS().
- 4) The status indication of each ESIO channel :
ESIO_GetStatus(), ESIO_GetParityErrorFlag(), ESIO_ClrAllParityErrFlag(),
ESIO_GetHorizontalParityError(), ESIO_ClrHorizontalParityError(),
ESIO_GetVerticalParityErrFrame(), ESIO_ClrAllVerticalParityErrNum(),
ESIO_IsRegisterModifiable(), ESIO_GetTxRxCtrl().
- 5) Configure interrupt,
ESIO_SetINT().

8.2.3 Function Documentation

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

TSB_ESIO0, TSB_ESIO1, TSB_ESIO2

8.2.3.1 ESIO_SWReset

Software reset ESIO module.

Prototype:

void
ESIO_SWReset(TSB_ESIO_TypeDef * **ESIOx**)

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function software reset the specified ESIO channel selected by **ESIOx**.

Return:

None.

8.2.3.2 ESIO_Enable

Enable the specified ESIO channel.

Prototype:

void
ESIO_Enable(TSB_ESIO_TypeDef * **ESIOx**)

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function enables the specified ESIO channel selected by **ESIOx**.

Return:

None.

Note:

This function must be called at first.

8.2.3.3 ESIO_Disable

Disable the specified ESIO channel

Prototype:

void
ESIO_Disable(TSB_ESIO_TypeDef * **ESIOx**)

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function disables the specified ESIO channel selected by **ESIOx**.

Return:

None.

8.2.3.4 ESIO_SetTxRxCtrl

Enable/Disable ESIO Transfer.

Prototype:

```
void  
ESIO_SetTxRxCtrl (TSB_ESIO_TypeDef * ESIOx,  
                  FunctionalState NewState)
```

Parameters:

ESIOx is the specified ESIO channel.

NewState specify the state for transfer, which can be:

- **ENABLE** for enable transfer
- **DISABLE** for disable transfer

Description:

This function enables/disables ESIO transfer of the specified ESIO channel selected by **ESIOx**.

Return:

None.

8.2.3.5 ESIO_GetTxRxCtrl

Get the ESIO transfer control status.

Prototype:

```
FunctionalState  
ESIO_GetTxRxCtrl(TSB_ESIO_TypeDef * ESIOx)
```

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function gets the ESIO transfer control status

Return:

The transfer control status of specify ESIO unit:

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

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

8.2.3.6 ESIO_SelectMode

Select ESIO mode: as SIO or SPI.

Prototype:

void

```
ESIO_SelectMode(TSB_ESIO_TypeDef * ESIOx,  
                ESIO_Mode Mode)
```

Parameters:

ESIOx is the specified ESIO channel.

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

- **ESIO_MODE_SPI:** The ESIO channel work as SPI mode (Using ESIOxCS pins) .
- **ESIO_MODE_SIO:** The ESIO channel work as SIO mode (Not using ESIOxCS pins) .

Description:

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

Return:

None.

8.2.3.7 ESIO_SelectTransferMode

Select ESIO transfer mode

Prototype:

void

```
ESIO_SelectTransferMode(TSB_ESIO_TypeDef * ESIOx,  
                        ESIO_TransferMode TrMode)
```

Parameters:

ESIOx is the specified ESIO channel.

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

- **ESIO_TRMODE_TX**: The specified ESIO channel work as Half duplex (Transmit) mode.
- **ESIO_TRMODE_RX**: The specified ESIO channel work as Half duplex (Receive) mode.
- **ESIO_TRMODE_TXRX**: The specified ESIO channel work as Full duplex mode.

Description:

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

Return:

None.

8.2.3.8 ESIO_SetCS

Select chip selection pin.

Prototype:

```
void  
ESIO_SetCS(TSB_ESIO_TypeDef * ESIOx,  
           ESIO_CSx CSx)
```

Parameters:

ESIOx is the specified ESIO channel.

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

- **ESIO_CS0**: enable chip connect to CS0.
- **ESIO_CS1**: enable chip connect to CS1.

Description:

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

Return:

None.

8.2.3.9 ESIO_SetTransferNum

Set the number of transfer frame.

Prototype:

```
void  
ESIO_SetTransferNum(TSB_ESIO_TypeDef * ESIOx,  
                   uint8_t Num)
```

Parameters:

ESIOx is the specified ESIO 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.

8.2.3.10 ESIO_SetTxPinInIdle

Set the status of ESIOxTXD pin during idle cycle

Prototype:

```
void  
ESIO_SetTxPinInIdle(TSB_ESIO_TypeDef * ESIOx,  
                    ESIO_TxPinInIdle PinMode);
```

Parameters:

ESIOx is the specified ESIO channel.

PinMode Specify the mode of ESIOxTXD pin during idle cycle. This parameter can be:

- **ESIO_TXPIN_FINALBIT**: Keep a final bit.
- **ESIO_TXPIN_KEEPLow**: Keep a Low level output.
- **ESIO_TXPIN_KEEPhigh**: Keep a High level output.

Description:

This function sets the status of ESIOxTXD pin of the specified ESIO during idle cycle

Return:

None.

8.2.3.11 ESIO_SetFIFOLevelINT

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

Prototype:

```
void  
ESIO_SetFIFOLevelINT(TSB_ESIO_TypeDef * ESIOx,
```

uint8_t *TxRx*,
uint8_t *Level*)

Parameters:

ESIOx is the specified ESIO channel.

TxRx specify transmit or receive. This parameter can be:

- **ESIO_TX**: specify transmit mode.
- **ESIO_RX**: specify receive mode.

Level specify the FIFO fill level to generate interrupt. This parameter can be: 0 to 8 (depended on the number of LINEs and the length of frame)

Description:

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

Return:

None.

8.2.3.12 ESIO_SetDMA

Set Transfer DMA request control.

Prototype:

```
void  
ESIO_SetDMA(TSB_ESIO_TypeDef * ESIOx,  
            uint8_t TxRx,  
            FunctionalState NewState)
```

Parameters:

ESIOx is the specified ESIO channel.

TxRx specify transmit or receive. This parameter can be:

- **ESIO_TX**: specify transmit mode.
- **ESIO_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 ESIO channel.
- **DISABLE**: disable DMA of the specified ESIO channel.

Description:

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

Return:

None.

8.2.3.13 ESIO_SetINT

Enable/Disable the interrupt source.

Prototype:

void

```
ESIO_SetINT(TSB_ESIO_TypeDef * ESIOx,  
            uint32_t IntSrc,  
            FunctionalState NewState)
```

Parameters:

ESIOx is the specified ESIO channel.

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

- **ESIO_INT_PERR: Parity error interrupt control.**
- **ESIO_INT_RXEND: Receive completion interrupt control.**
- **ESIO_INT_RXFIFO: Receive FIFO interrupt control.**
- **ESIO_INT_TXEND: Transmit completion interrupt control.**
- **ESIO_INT_TXFIFO: Transmit FIFO interrupt control.**
- **ESIO_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 ESIO channel.
- **DISABLE:** disable Interrupt of the specified ESIO channel.

Description:

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

Return:

None.

8.2.3.14 ESIO_InitFIFO

Initialize transfer FIFO pointer.

Prototype:

void

```
ESIO_InitFIFO(TSB_ESIO_TypeDef * ESIOx,  
              uint8_t TxRx);
```

Parameters:

ESIOx is the specified ESIO channel.

TxRx specify transmit or receive. This parameter can be:

- **ESIO_TX:** specify transmit mode.

- **ESIO_RX**: specify receive mode.

Description:

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

Return:

None.

8.2.3.15 ESIO_SetBaudRate

Set the clock and divider for baud rate generator

Prototype:

void

```
ESIO_SetBaudRate(TSB_ESIO_TypeDef * ESIOx,  
                 ESIO_BaudClock Clk,  
                 uint8_t Divider);
```

Parameters:

ESIOx is the specified ESIO channel.

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

- **ESIO_PHIT0_DIVIDE_2** : Input clock to the baud rate generator T0/2
- **ESIO_PHIT0_DIVIDE_4** : Input clock to the baud rate generator T0/4
- **ESIO_PHIT0_DIVIDE_8** : Input clock to the baud rate generator T0/8
- **ESIO_PHIT0_DIVIDE_16** : Input clock to the baud rate generator T0/16
- **ESIO_PHIT0_DIVIDE_32** : Input clock to the baud rate generator T0/32
- **ESIO_PHIT0_DIVIDE_64** : Input clock to the baud rate generator T0/64
- **ESIO_PHIT0_DIVIDE_128** : Input clock to the baud rate generator T0/128
- **ESIO_PHIT0_DIVIDE_256** : Input clock to the baud rate generator T0/256
- **ESIO_PHIT0_DIVIDE_512** : Input clock to the baud rate generator T0/512
- **ESIO_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 ESIO channel.

Return:

None.

8.2.3.16 ESIO_Init

Initialize the specified ESIO channel.

Prototype:

```
void  
ESIO_Init(TSB_ESIO_TypeDef * ESIOx,  
          ESIO_InitTypeDef * Init);
```

Parameters:

ESIOx is the specified ESIO channel.

Init is the structure containing basic ESIO configuration.

Description:

This function initializes the specified ESIO channel with the structure ESIO_InitTypeDef .

Return:

None.

Note:

Call ESIO_SetTransferNum() function at first

8.2.3.17 ESIO_SetTxData

Write data to transmit FIFO.

Prototype:

```
void  
ESIO_SetTxData(TSB_ESIO_TypeDef * ESIOx,  
               uint32_t Dat);
```

Parameters:

ESIOx is the specified ESIO channel.

Dat: The data which will be sent

Description:

This function writes data to transmit FIFO.

Return:

None.

8.2.3.18 ESIO_GetRxData

Initialize transfer FIFO pointer.

Prototype:

```
uint32_t  
ESIO_GetRxData(TSB_ESIO_TypeDef * ESIOx);
```

Parameters:

ESIOx is the specified ESIO channel.

Description:

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

Return:

The received data.

8.2.3.19 ESIO_IsRegisterModifiable

Check if ESIO registers is modifiable.

Prototype:

```
FunctionalState  
ESIO_IsRegisterModifiable(TSB_ESIO_TypeDef * ESIOx);
```

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function checks if ESIO registers is modifiable.

Return:

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

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

8.2.3.20 ESIO_GetStatus

Get the ESIO status.

Prototype:

```
ESIO_StatusFlag
```

ESIO_GetStatus(TSB_ESIO_TypeDef * **ESIOx**);

Parameters:

ESIOx is the specified ESIO channel.

Description:

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

Return:

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

8.2.3.21 ESIO_GetParityErrorFlag

Get the ESIO error information.

Prototype:

ESIO_ParityErrNum

ESIO_GetParityErrorFlag(TSB_ESIO_TypeDef * **ESIOx**);

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function returns the ESIO error information of the specified ESIO channel by reading ESIO Parity Error Flag Register

Return:

The structure of the error number for LINEy and Horizontal/Vertical error status of the specified ESIO channel

8.2.3.22 ESIO_ClrAllParityErrFlag

Clear all ESIO error flag.

Prototype:

void

ESIO_ClrAllParityErrFlag(TSB_ESIO_TypeDef * **ESIOx**);

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function clears all ESIO error flag of the specified ESIO channel.

Return:

None.

8.2.3.23 ESIO_GetHorizontalParityError

Get the ESIO status.

Prototype:

uint32_t

```
ESIO_GetHorizontalParityError(TSB_ESIO_TypeDef * ESIOx,  
                             ESIO_LINEx Line);
```

Parameters:

ESIOx is the specified ESIO channel.

Line select the LINE number

- **ESIO_LINE0**: ESIO Horizontal Parity Error flag Register0.
- **ESIO_LINE1**: ESIO Horizontal Parity Error flag Register1.
- **ESIO_LINE2**: ESIO Horizontal Parity Error flag Register2.
- **ESIO_LINE3**: ESIO Horizontal Parity Error flag Register3.

Description:

This function returns Horizontal parity error flag by reading the ESIO Horizontal Parity Error flag Register of the specified ESIO channel.

Return:

The horizontal error information for LINEy.

8.2.3.24 ESIO_ClrHorizontalParityError

Clear the ESIO horizontal error information.

Prototype:

void

```
ESIO_ClrHorizontalParityError(TSB_ESIO_TypeDef * ESIOx,  
                              ESIO_LINEx Line);
```

Parameters:

ESIOx is the specified ESIO channel.

Line select the LINE number

- **ESIO_LINE0**: ESIO Horizontal Parity Error flag Register0.

- **ESIO_LINE1:** ESIO Horizontal Parity Error flag Register1.
- **ESIO_LINE2:** ESIO Horizontal Parity Error flag Register2.
- **ESIO_LINE3:** ESIO Horizontal Parity Error flag Register3.

Description:

This function clears Horizontal parity error flag by writing '1' to the ESIO Horizontal Parity Error flag Register of the specified ESIO channel.

Return:

None.

8.2.3.25 ESIO_GetVerticalParityErrFrame

Get the ESIO status.

Prototype:

```
ESIO_VericalParityErrFrame ESIO_GetVerticalParityErrFrame(TSB_ESIO_TypeDef  
* ESIOx);
```

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function returns the ESIO vertical error information by reading ESIO Vertical Parity Error Frame Number Register of the specified ESIO channel.

Return:

The vertical error infomation for LINEy.

8.2.3.26 ESIO_ClrAllVerticalParityErrNum

Clear the ESIO vertical error information.

Prototype:

```
void  
ESIO_ClrAllVerticalParityErrNum(TSB_ESIO_TypeDef * ESIOx);
```

Parameters:

ESIOx is the specified ESIO channel.

Description:

This function clears the ESIO vertical error information by writing '1' to ESIO Vertical Parity Error Frame Number Register of the specified ESIO channel.

Return:

None.

8.2.4 Data Structure Description

8.2.4.1 ESIO_InitTypeDef

Data Fields:

uint8_t

DataDirection Transfer direction selection, which can be set as:

- **ESIO_LSB_FIRST**: transfer begin from the LSB
- **ESIO_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,which can be : 0 to 15.

UInt8_t

NumberOfLINE The number of LINE setting, which can be : 0 to 4.

UInt8_t

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

- **ESIO_CS_ACTIVE_LOW**: Low level
- **ESIO_CS_ACTIVE_HIGH**: High level

uint8_t

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

uint8_t

CS1ActiveLevel low active or high active for pin CS1,which can be set as :

- **ESIO_LSB_FIRST**: transfer begin from the LSB
- **ESIO_MSB_FIRST**: transfer begin from the MSB

uint8_t

CS0ActiveLevel low active or high active for pin CS0,which can be set as :

- **ESIO_LSB_FIRST**: transfer begin from the LSB
- **ESIO_MSB_FIRST**: transfer begin from the MSB

uint8_t

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

uint8_t

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

FunctionalState

HorizontalParityCheck Horizontal parity can be added in burst transfer.

Horizontal parity must be disabled in single transfer. it can be set as :

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

uint8_t

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

- **ESIO_PARITY_EVEN:** Enable even horizontal parity check.
- **ESIO_PARITY_ODD:** Enable odd horizontal parity check

FunctionalState

VerticalParityCheck Add vertical parity, which can be set as :

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

uint8_t

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

- **ESIO_PARITY_EVEN:** Enable even horizontal parity check.
- **ESIO_PARITY_ODD:** Enable odd horizontal parity check

8.2.4.2 ESIO_StatusFlag

Data Fields:

uint32_t

All: ESIO 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		
<i>TxFIFOFillLevel</i> : 4		Transmit FIFO fill level
uint32_t		
<i>TxFIFOEmpty</i> : 1		Transmit FIFO is empty
uint32_t		
<i>TxFIFO_INT</i> : 1		Transmit FIFO interrupt generated
uint32_t		
<i>TxCompleted</i> : 1		Transmission completed
uint32_t		
<i>TxRUN</i> : 1		Transmit shift is operating
uint32_t		
<i>Reserved2</i> : 7		Reserved
uint32_t		
<i>ESIO_RegUsing</i> : 1		ESIO registers must not be modified.

8.2.4.3 ESIO_ParityErrNum

Data Fields:

uint32_t

All: ESIO Parity Error Num.

Bit

uint32_t		
<i>LINE0_ParityErrNum</i> : 2		LINE 0 parity error numbers
uint32_t		
<i>LINE1_ParityErrNum</i> : 2		LINE 1 parity error numbers
uint32_t		
<i>LINE2_ParityErrNum</i> : 2		LINE 2 parity error numbers
uint32_t		
<i>LINE3_ParityErrNum</i> : 2		LINE 3 parity error numbers
uint32_t		
<i>HorizontalParityErr</i> : 1		Horizontal parity error occurred.
uint32_t		
<i>VerticalParityErr</i> : 1		Vertical parity error occurred.
uint32_t		
<i>Reserved1</i> : 22		Reserved

8.2.4.4 ESIO_VerticalParityErrFrame

Data Fields:

uint32_t

All: ESIO Vertical Parity Error Frame.

Bit

uint32_t

LINE0_VeriticalParityErrFrame: 8	The number of frame which has first vertical parity error in LINE 0
uint32_t	
LINE1_VeriticalParityErrFrame: 8	The number of frame which has first vertical parity error in LINE 1
uint32_t	
LINE2_VeriticalParityErrFrame: 8	The number of frame which has first vertical parity error in LINE 2
uint32_t	
LINE3_VeriticalParityErrFrame: 8	The number of frame which has first vertical parity error in LINE 3

9. EXB

9.1 Overview

The TMPM440 has a built-in external bus interface to connect to external memory, I/Os, etc. This interface consists of an external bus interface circuit (EBIF), a chip selector (CS) and a wait controller.

The chip selector and wait controller designate mapping addresses in a 2-block address space and also control wait states and data bus widths (8- or 16-bit) in these space.

The external bus interface circuit (EBIF) controls the timing of external buses based on the chip selector and wait controller settings.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_exb.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_exb.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 Function List

- ◆ void EXB_SetBusMode(uint8_t **BusMode**);
- ◆ void EXB_SetBusCycleExtension(uint8_t **Cycle**);
- ◆ void EXB_SetEndianType(uint8_t **ChipSelect**, EXB_EndianType **EndianType**);
- ◆ void EXB_Enable(uint8_t **ChipSelect**);
- ◆ void EXB_Disable(uint8_t **ChipSelect**);
- ◆ void EXB_Init(uint8_t **ChipSelect**, EXB_InitTypeDef* **InitStruct**);

9.2.2 Detailed Description

Functions listed above can be divided into three parts:

- 1) Configure the EXB bus mode, bus cycle extension, data bus widths and the cycle of external buses, based on the chip selector.

EXB_SetBusMode(), EXB_SetBusCycleExtension() and EXB_Init().

- 2) Enable and disable control

EXB_Enable(), EXB_Disable().

- 3) Set the endian type for external memory or peripheral I/O.

EXB_SetEndianType();

9.2.3 Function Documentation

9.2.3.1 EXB_SetBusMode

Set external bus mode for EXB.

Prototype:

void

EXB_SetBusMode(uint8_t **BusMode**)

Parameters:

BusMode : select EXB bus mode.

The value could be the following values:

- **EXB_BUS_SEPARATE** for separate bus mode.
- **EXB_BUS_MULTIPLEX** for multiplex bus mode.

Description:

This function sets the bus mode for the external bus. When **BusMode** is **EXB_BUS_SEPARATE**, the bus mode will be separate mode. When **BusMode** is **EXB_BUS_MULTIPLEX**, the bus mode will be multiplex mode.

Return:

None

9.2.3.2 EXB_SetBusCycleExtension

Set the bus cycle to be double or quadruple.

Prototype:

void

EXB_SetBusCycleExtension(uint8_t **Cycle**)

Parameters:

Cycle: Set the bus cycle to be double or quadruple.

The value could be the following values:

- **EXB_CYCLE_NONE:** EXB bus cycle will not be extended.
- **EXB_CYCLE_DOUBLE:** EXB bus cycle will be double.
- **EXB_CYCLE_QUADRUPLE:** EXB bus cycle will be quadruple.

Description:

This function will set bus cycle extension for the setup cycles, wait cycles and recovery cycles of the bus timing, which can be double or quadruple.

Return:

None

9.2.3.3 EXB_SetEndianType

Set the endian type for external memory or peripheral I/O.

Prototype:

void

EXB_SetEndianType (uint8_t **ChipSelect**, EXB_EndianType **EndianType**)

Parameters:

ChipSelect is the specified chip.

The value could be the following values:

- **EXB_CS0:** for chip 0
- **EXB_CS1:** for chip 1

EndianType classifier a type

This parameter can be one of the following values:

- **SAME_ENDIAN_AS_CPU:** Set the endian type same as cpu
- **NOT_SAME_ENDIAN_AS_CPU:** Set the endian type not same as cpu

Description:

This function will set Set the endian type for external memory or peripheral I/O

Return:

None

9.2.3.4 EXB_Enable

Enable the specified chip.

Prototype:

void

EXB_Enable(uint8_t **ChipSelect**)

Parameters:

ChipSelect is the specified chip.

The value could be the following values:

- **EXB_CS0**: for chip 0
- **EXB_CS1**: for chip 1

Description:

This function will enable the access to the specified chip.

Return:

None

9.2.3.5 EXB_Disable

Disable the specified chip.

Prototype:

void

EXB_Disable(uint8_t **ChipSelect**)

Parameters:

ChipSelect is the specified chip.

The value could be the following values:

- **EXB_CS0**: for chip 0
- **EXB_CS1**: for chip 1

Description:

This function will disable the access to the specified chip.

Return:

None

9.2.3.6 EXB_Init

Initialize the specified chip.

Prototype:

void

EXB_Init (uint8_t **ChipSelect**,
EXB_InitTypeDef* **InitStruct**)

Parameters:

ChipSelect is the specified chip.

The value could be the following values:

- **EXB_CS0**: for chip 0
- **EXB_CS1**: for chip 1

InitStruct is the structure containing basic EXB configuration including address space size, chip start address, data bus width and the cycle of external buses. (Refer to “Data Structure Description” for details)

Description:

This function will initialize the EXB interface for the specified chip.

Return:

None

9.2.4 Data Structure Description

9.2.4.1 EXB_InitTypeDef

Data Fields:

uint8_t

AddrSpaceSize Set the address space size, which can be set as:

- **EXB_16M_BYTE**: address space is 16Mbyte,
- **EXB_8M_BYTE**: address space is 8Mbyte,
- **EXB_4M_BYTE**: address space is 4Mbyte,
- **EXB_2M_BYTE**: address space is 2Mbyte,
- **EXB_1M_BYTE**: address space is 1Mbyte,
- **EXB_512K_BYTE**: address space is 512Kbyte,
- **EXB_256K_BYTE**: address space is 256Kbyte,
- **EXB_128K_BYTE**: address space is 128Kbyte,
- **EXB_64K_BYTE**: address space is 64Kbyte.

UInt16_t

StartAddr Set the start address. The max value is 0x1FF.

uint8_t

BusWidth Set the data bus width, which can be set as:

- **EXB_BUS_WIDTH_BIT_8**: data bus width is 8bit,
- **EXB_BUS_WIDTH_BIT_16**: data bus width is 16bit.

EXB_CyclesTypeDef

Cycles Set the cycle of external buses, which consists of following members:

InternalWait, **ReadSetupCycle**, **WriteSetupCycle**, **ALEWaitCycle** (For multiplex bus mode only), **ReadRecoveryCycle**, **WriteRecoveryCycle** and

ChipSelectRecoveryCycle. (Refer to “EXB_CyclesTypeDef” for details)

9.2.4.2 EXB_CyclesType Def

Data Fields:

uint8_t

InternalWait Set the internal wait, which can be set as:

- **EXB_INTERNAL_WAIT_0**: 0 wait,
- **EXB_INTERNAL_WAIT_1**: 1 wait,
- **EXB_INTERNAL_WAIT_2**: 2 waits,
- **EXB_INTERNAL_WAIT_3**: 3 waits,
- **EXB_INTERNAL_WAIT_4**: 4 waits,
- **EXB_INTERNAL_WAIT_5**: 5 waits,
- **EXB_INTERNAL_WAIT_6**: 6 waits,
- **EXB_INTERNAL_WAIT_7**: 7 waits,
- **EXB_INTERNAL_WAIT_8**: 8 waits,
- **EXB_INTERNAL_WAIT_9**: 9 waits,
- **EXB_INTERNAL_WAIT_10**: 10 waits,
- **EXB_INTERNAL_WAIT_11**: 11 waits,
- **EXB_INTERNAL_WAIT_12**: 12 waits,
- **EXB_INTERNAL_WAIT_13**: 13 waits,
- **EXB_INTERNAL_WAIT_14**: 14 waits,
- **EXB_INTERNAL_WAIT_15**: 15 waits.

uint8_t

ReadSetupCycle Set the read setup cycle, which can be set as:

- **EXB_CYCLE_0**: 0 cycle,
- **EXB_CYCLE_1**: 1 cycle,
- **EXB_CYCLE_2**: 2 cycles,
- **EXB_CYCLE_4**: 4 cycles.

uint8_t

WriteSetupCycle Set the write setup cycle, which can be set as:

- **EXB_CYCLE_0**: 0 cycle,
- **EXB_CYCLE_1**: 1 cycle,
- **EXB_CYCLE_2**: 2 cycles,
- **EXB_CYCLE_4**: 4 cycles.

uint8_t

ALEWaitCycle Set the ALE waits cycle for multiplex bus, which can be set as:

- **EXB_CYCLE_0**: 0 cycle,
- **EXB_CYCLE_1**: 1 cycle,
- **EXB_CYCLE_2**: 2 cycles,
- **EXB_CYCLE_4**: 4 cycles.

uint8_t

ReadRecoveryCycle Set the read recovery cycle, which can be set as:

- **EXB_CYCLE_0**: 0 cycle,
- **EXB_CYCLE_1**: 1 cycle,
- **EXB_CYCLE_2**: 2 cycles,
- **EXB_CYCLE_3**: 3 cycles,
- **EXB_CYCLE_4**: 4 cycles,
- **EXB_CYCLE_5**: 5 cycles,
- **EXB_CYCLE_6**: 6 cycles,
- **EXB_CYCLE_8**: 8 cycles.

uint8_t

WriteRecoveryCycle Set the write recovery cycle, which can be set as:

- **EXB_CYCLE_0**: 0 cycle,
- **EXB_CYCLE_1**: 1 cycle,
- **EXB_CYCLE_2**: 2 cycles,
- **EXB_CYCLE_3**: 3 cycles,
- **EXB_CYCLE_4**: 4 cycles,
- **EXB_CYCLE_5**: 5 cycles,
- **EXB_CYCLE_6**: 6 cycles,
- **EXB_CYCLE_8**: 8 cycles.

uint8_t

ChipSelectRecoveryCycle Set the chip select recovery cycle, which can be:

- **EXB_CYCLE_0**: 0 cycle,
- **EXB_CYCLE_1**: 1 cycle,
- **EXB_CYCLE_2**: 2 cycles,
- **EXB_CYCLE_4**: 4 cycles.

10. FC

10.1 Overview

TMPM440 device contains flash memory, the flash size of TMPM440F10XBG is 1024Kbytes, and flash size of TMPM440FEXBG is 768kbytes.

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\TX04_Periph_Driver\src\tmpm440_fc.c with \Libraries\TX04_Periph_Driver\inc\ tmpm440_fc.h containing the API definitions for use by applications.

10.2 API Functions

10.2.1 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)
- ◆ void FC_SetCtrlReg(FunctionalState **NewState**)

10.2.2 Detailed Description

Functions listed above can be divided into four parts:

- 1) The security function restricts flash ROM data readout and debugging.
FC_SetSecurityBit(), FC_GetSecurityBit().

- 2) The functions get the automatic operation status and each block protection status:
FC_GetBusyState(), FC_GetBlockProtectState().
- 3) The functions change the protection status of each block:
FC_ProgramBlockProtectState(), FC_EraseBlockProtectState().
- 4) Use automatic operation command to write or erase the content of flash.
FC_WritePage(), FC_EraseBlock(), FC_EraseChip().
- 5) Others:
FC_SetCtrlReg().

10.2.3 Function Documentation

10.2.3.1 FC_SetSecurityBit

Set the value of SECBIT register.

Prototype:

void

FC_SetSecurityBit (FunctionalState **NewState**)

Parameters:

NewState: Select the state of SECBIT register.

This parameter can be one of the following values:

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

Description:

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

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

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

Return:

None

10.2.3.2 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.

10.2.3.3 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.

10.2.3.4 FC_GetBlockProtectState

Get the block protection status.

Prototype:

FunctionalState

FC_GetBlockProtectState(uint8_t **BlockNum**)

Parameters:

BlockNum: The flash block number

TMPM440F10XBG:

➤ **FC_BLOCK_0** to **FC_BLOCK_27**

TMPM440FEXBG:

➤ **FC_BLOCK_0** to **FC_BLOCK_23**

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

10.2.3.5 FC_ProgramBlockProtectState

Program the protection bits.

Prototype:

FC_Result

FC_ProgramProtectState(uint8_t **BlockNum**)

Parameters:

TMPM440F10XBG:

➤ **FC_BLOCK_0** to **FC_BLOCK_27**

TMPM440FEXBG:

➤ **FC_BLOCK_0** to **FC_BLOCK_23**

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.

10.2.3.6 FC_EraseBlockProtectState

Erase the protection bits.

Prototype:

FC_Result

FC_EraseBlockProtectState(uint8_t **BlockGroup**)

Parameters:

BlockGroup:The flash block group

- **FC_BLOCK_GROUP_0:** block 0 to 7.
- **FC_BLOCK_GROUP_1:** block 8 to 13.
- **FC_BLOCK_GROUP_2:** block 14 to 21.
- **FC_BLOCK_GROUP_3:** block 22 to 27(for TPM440F10XBG);
block 22 to 23(for TPM440FEXBG);

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.

10.2.3.7 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 FC_PAGE_SIZE(TMPM440:512 Bytes).

Description:

This API is used to write data to specified page.

It contains 64 words in a page. The flash can only be written page by page.

The automatic page programming is allowed only once for a page already erased. No programming can be performed twice or more time irrespective of data value whether it is "1" or "0".

***Note:** An attempt to rewrite a page two or more times without erasing the content can cause damages to the device.

Return:

Result of the operation to write data to the specified page.

FC_SUCCESS: data is written to the specified page accurately.

FC_ERROR_PROTECTED: The block is protected. The write operation can't be executed.

FC_ERROR_OVER_TIME: Write operation over time error.

10.2.3.8 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.

10.2.3.9 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.

10.2.3.10 FC_SetCtrlReg

Enable or Disable flash buffer

Prototype:

void

FC_SetCtrlReg(FunctionalState **NewState**)

Parameters:

NewState: Flash buffer is enabled /disabled.

This parameter can be one of the following values:

ENABLE: Flash buffer is enabled.

DISABLE: Flash buffer is disabled.

10.2.4 Data Structure Description

None

11. FUART

11.1 Overview

TOSHIBA TPM440 contains the Asynchronous serial channel (Full UART) with Modem control.

TOSHIBA TPM440 contains two channels Full UART: FUART0, FUART1.

The FUART driver APIs provide a set of functions to configure the Full UART 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/TX04_Periph_Driver/src/tmpm440_fuart.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_fuart.h containing the macros, data types, structures and API definitions for use by applications.

11.2 API Functions

11.2.1 Function List

- ◆ void FUART_Enable(TSB_FUART_TypeDef * **FUARTx**)
- ◆ void FUART_Disable(TSB_FUART_TypeDef * **FUARTx**)
- ◆ uint32_t FUART_GetRxData(TSB_FUART_TypeDef * **FUARTx**)
- ◆ void FUART_SetTxData(TSB_FUART_TypeDef * **FUARTx**, uint32_t **Data**)
- ◆ FUART_Err FUART_GetErrStatus(TSB_FUART_TypeDef * **FUARTx**)
- ◆ void FUART_ClearErrStatus(TSB_FUART_TypeDef * **FUARTx**)
- ◆ WorkState FUART_GetBusyState(TSB_FUART_TypeDef * **FUARTx**)
- ◆ FUART_StorageStatus FUART_GetStorageStatus(TSB_FUART_TypeDef * **FUARTx**, UART_Direction **Direction**)
- ◆ void FUART_Init(TSB_FUART_TypeDef * **FUARTx**, FUART_InitTypeDef * **InitStruct**)
- ◆ void FUART_EnableFIFO(TSB_FUART_TypeDef * **FUARTx**)
- ◆ void FUART_DisableFIFO(TSB_FUART_TypeDef * **FUARTx**)
- ◆ void FUART_SetSendBreak(TSB_FUART_TypeDef * **FUARTx**, FunctionalState **NewState**)
- ◆ void FUART_SetINTFIFOLevel(TSB_FUART_TypeDef * **FUARTx**, uint32_t **RxLevel**, uint32_t **TxLevel**)
- ◆ void FUART_SetINTMask(TSB_FUART_TypeDef * **FUARTx**, uint32_t **IntMaskSrc**)
- ◆ FUART_INTStatus FUART_GetINTMask(TSB_FUART_TypeDef * **FUARTx**)
- ◆ FUART_INTStatus FUART_GetRawINTStatus(TSB_FUART_TypeDef * **FUARTx**)
- ◆ FUART_INTStatus FUART_GetMaskedINTStatus(TSB_FUART_TypeDef * **FUARTx**)

- ◆ void FUART_ClearINT(TSB_FUART_TypeDef * **FUARTx**, FUART_INTStatus **INTStatus**)
- ◆ void FUART_SetDMAOnErr(TSB_FUART_TypeDef * **FUARTx**, FunctionalState **NewState**)
- ◆ void FUART_SetFIFODMA(TSB_FUART_TypeDef * **FUARTx**, FUART_Direction **Direction**, FunctionalState **NewState**)
- ◆ FUART_AllModemStatus FUART_GetModemStatus(TSB_FUART_TypeDef * **FUARTx**)

11.2.2 Detailed Description

Functions listed above can be divided into five parts:

- 1) Full UART Configuration and Initialization, common operation
FUART_Enable(), FUART_Disable, FUART_Init(), FUART_GetRxData(),
FUART_SetTxData(), FUART_GetErrStatus(), FUART_ClearErrStatus(),
FUART_GetBusyState(), FUART_GetStorageStatus(), FUART_SetSendBreak()
- 2) Configure FIFO and DMA.
FUART_EnableFIFO(), FUART_DisableFIFO(), FUART_SetINTFIFOLevel(),
FUART_SetFIFODMA, FUART_SetDMAOnErr().
- 3) Configure interrupt, get interrupt status and clear interrupt.
FUART_SetINTMask(), FUART_GetINTMask(), FUART_GetRawINTStatus(),
FUART_GetMaskedINTStatus, FUART_ClearINT().
- 4) Modem control.
FUART_GetModemStatus().

11.2.3 Function Documentation

Note: in all of the following APIs, parameter “TSB_FUART_TypeDef* **FUARTx**” can be **FUART0** or **FUART1**.

11.2.3.1 FUART_Enable

Enable the specified Full UART channel.

Prototype:

```
void  
FUART_Enable(TSB_FUART_TypeDef * FUARTx)
```

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will enable the specified Full UART channel selected by **FUARTx**.

Return:

None

11.2.3.2 FUART_Disable

Disable the specified Full UART channel.

Prototype:

void

FUART_Disable(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will disable the specified Full UART channel selected by **FUARTx**.

Return:

None

11.2.3.3 FUART_GetRxData

Get received data from the specified Full UART channel.

Prototype:

uint32_t

FUART_GetRxData(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will get the data received from the specified Full UART channel selected by **FUARTx**. It is appropriate to call the function after

FUART_GetStorageStatus(FUARTx, FUART_RX) returns **FUART_STORAGE_NORMAL** or **FUART_STORAGE_FULL**.

Return:

The data received from the specified Full UART channel

11.2.3.4 FUART_SetTxData

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

Prototype:

void

```
FUART_SetTxData(TSB_FUART_TypeDef * FUARTx,  
                uint32_t Data)
```

Parameters:

FUARTx: The specified Full UART channel.

Data: A frame to be sent, which can be 5-bit, 6-bit, 7-bit or 8-bit, depending on the initialization. The Data range is 0x00 to 0xFF.

Description:

This API will set data to be sent and start transmitting via the specified Full UART channel selected by **FUARTx**.

Return:

None

11.2.3.5 FUART_GetErrStatus

Get receive error status.

Prototype:

FUART_Err

```
FUART_GetErrStatus(TSB_FUART_TypeDef * FUARTx)
```

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will get the error status after a data has been transferred, so this API must be executed after **FUART_GetRxData(FUARTx)**, only in this read sequence can the right error status information be got.

Return:

FUART_NO_ERR means there is no error in the last transfer.

FUART_OVERRUN means that overrun occurs in the last transfer.

FUART_PARITY_ERR means either even parity or odd parity fails.

FUART_FRAMING_ERR means there is framing error in the last transfer.

FUART_BREAK_ERR means there is break error in the last transfer.

FUART_ERRS means that 2 or more errors occurred in the last transfer.

11.2.3.6 FUART_ClearErrStatus

Clear receive error status.

Prototype:

void

FUART_ClearErrStatus(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will clear all the receive errors, including framing, parity, break and overrun errors.

Return:

None

11.2.3.7 FUART_GetBusyState

Get the state that whether the specified Full UART channel is transmitting data or stopped.

Prototype:

WorkState

FUART_GetBusyState(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will get the work state of the specified Full UART channel to see if it is transmitting data or stopped.

Return:

Work state of the specified Full UART channel:

BUSY: The Full UART is transmitting data

DONE: The Full UART has stopped transmitting data

11.2.3.8 FUART_GetStorageStatus

Get the FIFO or hold register status.

Prototype:

FUART_StorageStatus

FUART_GetStorageStatus(TSB_FUART_TypeDef * **FUARTx**,
FUART_Direction **Direction**)

Parameters:

FUARTx: The specified Full UART channel.

Direction: The direction of Full UART

- **FUART_RX:** for receive FIFO or receive hold register
- **FUART_TX:** for transmit FIFO or transmit hold register

Description:

When FIFO is enabled, this API will get the transmit or receive FIFO status.

When FIFO is disabled, this API will get the transmit or receive hold register status.

Return:

FUART_StorageStatus: The FIFO or hold register status.

FUART_STORAGE_EMPTY: The FIFO or the hold register is empty.

FUART_STORAGE_NORMAL: The FIFO is normal, not empty and not full.

FUART_STORAGE_INVALID: The FIFO or the hold register is in invalid status.

FUART_STORAGE_FULL: The FIFO or the hold register is full.

11.2.3.9 FUART_Init

Initialize and configure the specified Full UART channel.

Prototype:

void

FUART_Init(TSB_FUART_TypeDef * **FUARTx**,
FUART_InitTypeDef * **InitStruct**)

Parameters:

FUARTx: The specified Full UART channel.

InitStruct: The structure containing Full UART configuration including baud rate, data bits per transfer, stop bits, parity, transfer mode and flow control (Refer to “Data Structure Description” for details).

Description:

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

This API must be executed before Full UART is enabled.

Return:

None

11.2.3.10 FUART_EnableFIFO

Enable the transmit and receive FIFO.

Prototype:

void

FUART_EnableFIFO(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will enable the transmit and receive FIFO of the specified UART channel selected by **FUARTx**.

Return:

None

11.2.3.11 FUART_DisableFIFO

Disable the transmit and receive FIFO and the mode will be changed to character mode.

Prototype:

FUART_DisableFIFO(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will disable the transmit and receive FIFO of the specified UART channel selected by **FUARTx**. Then the Full UART work mode will be changed from FIFO mode to character mode.

Return:

None

11.2.3.12 FUART_SetSendBreak

Generate the break condition for Full UART.

Prototype:

void

FUART_SetSendBreak(TSB_FUART_TypeDef * **FUARTx**,
FunctionalState **NewState**)

Parameters:

FUARTx: The specified Full UART channel.

NewState: New state of the FUART send break.

- **ENABLE**: Enable the send break to generate transmit break condition
- **DISABLE**: Disable the send break

Description:

This API is used to generate the transmit break condition. For generation of the transmit break condition, the send break function must be enabled by this API while at least one frame or longer being transmitted. Even when the break condition is generated, the contents of the transmit FIFO are not affected.

Return:

None

11.2.3.13 FUART_SetINTFIFOLevel

Set the Receive and Transmit interrupt FIFO level.

Prototype:

void

FUART_SetINTFIFOLevel(TSB_FUART_TypeDef * **FUARTx**,
uint32_t **RxLevel**,
uint32_t **TxLevel**)

Parameters:

FUARTx: The specified Full UART channel.

RxLevel: Receive interrupt FIFO level. (Receive FIFO is 32 location deep)

- **FUART_RX_FIFO_LEVEL_4:** The data in Receive FIFO become ≥ 4 words
- **FUART_RX_FIFO_LEVEL_8:** The data in Receive FIFO become ≥ 8 words
- **FUART_RX_FIFO_LEVEL_16:** The data in Receive FIFO become ≥ 16 words
- **FUART_RX_FIFO_LEVEL_24:** The data in Receive FIFO become ≥ 24 words
- **FUART_RX_FIFO_LEVEL_28:** The data in Receive FIFO become ≥ 28 words

TxLevel: Transmit interrupt FIFO level. (Transmit FIFO is 32 location deep)

- **FUART_TX_FIFO_LEVEL_4:** The data in Transmit FIFO become ≤ 4 words
- **FUART_TX_FIFO_LEVEL_8:** The data in Transmit FIFO become ≤ 8 words
- **FUART_TX_FIFO_LEVEL_16:** The data in Transmit FIFO become ≤ 16 words
- **FUART_TX_FIFO_LEVEL_24:** The data in Transmit FIFO become ≤ 24 words
- **FUART_TX_FIFO_LEVEL_28:** The data in Transmit FIFO become ≤ 28 words

Description:

This API is used to define the FIFO level at which UARTRXINTR and UARTRXINTR are generated. The interrupts are generated based on a transition through a level rather than based on the level.

Return:

None

11.2.3.14 FUART_SetINTMask

Mask(Enable) interrupt source of the specified channel.

Prototype:

void

FUART_SetINTMask(TSB_FUART_TypeDef * **FUARTx**,
uint32_t **IntMaskSrc**)

Parameters:

FUARTx: The specified Full UART channel.

IntMaskSrc: The interrupt source to be masked(enabled).

To enable no interrupt, use the parameter:

- **FUART_NONE_INT_MASK**

To enable the interrupt one by one, use the “OR” operation with below parameter:

- **FUART_CTS_MODEM_INT_MASK:** Enable CTS modem interrupt
- **FUART_RX_FIFO_INT_MASK:** Enable receive FIFO interrupt
- **FUART_TX_FIFO_INT_MASK:** Enable transmit FIFO interrupt
- **FUART_RX_TIMEOUT_INT_MASK:** Enable receive timeout interrupt
- **FUART_FRAMING_ERR_INT_MASK:** Enable framing error interrupt
- **FUART_PARITY_ERR_INT_MASK:** Enable parity error interrupt

- **FUART_BREAK_ERR_INT_MASK**: Enable break error interrupt
 - **FUART_OVERRUN_ERR_INT_MASK**: Enable overrun error interrupt
- To enable all the interrupts, use the parameter:
- **FUART_ALL_INT_MASK**

Description:

This API will enable the interrupt source of the specified channel. With using this API, interrupts specified by **IntMaskSrc** will be enabled, the other interrupts will be disabled.

Return:

None

11.2.3.15 FUART_GetINTMask

Get the mask(Enable) setting for each interrupt source.

Prototype:

FUART_INTStatus

FUART_GetINTMask(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will get the Full UART interrupt configuration. This API can get the information that which interrupts are enabled and which interrupts are disabled.

Return:

FUART_INTStatus: The union that indicates interrupt enable configuration.
(Refer to “Data Structure Description” for details).

11.2.3.16 FUART_GetRawINTStatus

Get the raw interrupt status of the specified Full UART channel.

Prototype:

FUART_INTStatus

FUART_GetRawINTStatus(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will get the raw interrupt status of the specified Full UART channel specified by **FUARTx**.

Return:

FUART_INTStatus: The union that indicates the raw interrupt status.
(Refer to “Data Structure Description” for details).

11.2.3.17 FUART_GetMaskedINTStatus

Get the masked interrupt status of the specified Full UART channel.

Prototype:

FUART_INTStatus

FUART_GetMaskedINTStatus(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will get the masked interrupt status of the specified Full UART channel specified by **FUARTx**.

Return:

FUART_INTStatus: The union that indicates the masked interrupt status.
(Refer to “Data Structure Description” for details).

11.2.3.18 FUART_ClearINT

Clear the interrupts of the specified Full UART channel.

Prototype:

void

FUART_ClearINT(TSB_FUART_TypeDef * **FUARTx**,
FUART_INTStatus **INTStatus**)

Parameters:

FUARTx: The specified Full UART channel.

INTStatus: The union that indicates the interrupts to be cleared. When a bit of this parameter is set to 1, the associated interrupt is cleared.
(Refer to “Data Structure Description” for details).

Description:

This API can clear the interrupts of the specified channel selected by **FUARTx**.

Return:

None

11.2.3.19 FUART_SetDMAOnErr

Enable or disable the DMA receive request output on assertion of a UART error interrupt.

Prototype:

void

FUART_SetDMAOnErr(TSB_FUART_TypeDef * **FUARTx**,
FunctionalState **NewState**)

Parameters:

FUARTx: The specified Full UART channel.

NewState: New state of the DMA receive request output on assertion of a UART error interrupt.

- **ENABLE**: The DMA on error is available, the DMA receive request output, UARTRXDMASREQ or UARTRXDMABREQ, is disabled on assertion of a UART error interrupt.
- **DISABLE**: The DMA on error is not available, the DMA receive request output, UARTRXDMASREQ or UARTRXDMABREQ, is enabled on assertion of a UART error interrupt.

Description:

This API is used to enable or disable the DMA receive request output on assertion of a UART error interrupt.

Return:

None

11.2.3.20 FUART_SetFIFODMA

Enable or Disable the Transmit FIFO DMA or Receive FIFO DMA.

Prototype:

void

FUART_SetFIFODMA(TSB_FUART_TypeDef * **FUARTx**,

FUART_Direction **Direction**,
FunctionalState **NewState**)

Parameters:

FUARTx: The specified Full UART channel.

Direction: The direction of Full UART.

➤ **FUART_RX**: Receive FIFO

➤ **FUART_TX**: Transmit FIFO

NewState: New state of the FIFO DMA.

➤ **ENABLE**: Enable FIFO DMA

➤ **DISABLE**: Disable FIFO DMA

Description:

This API will enable or disable the Transmit FIFO DMA or Receive FIFO DMA.

The bus width must be set to 8-bits, if you transfer the data of transmit / receive FIFO by using DMAC.

Return:

None

11.2.3.21 FUART_GetModemStatus

Get the Modem Status: CTS

Prototype:

FUART_AllModemStatus

FUART_GetModemStatus(TSB_FUART_TypeDef * **FUARTx**)

Parameters:

FUARTx: The specified Full UART channel.

Description:

This API will enable the transmit and receive FIFO of the specified UART channel selected by **FUARTx**.

Return:

FUART_AllModemStatus: The union that indicates all the modem status.
(Refer to “Data Structure Description” for details).

11.2.4 Data Structure Description

11.2.4.1 FUART_InitTypeDef

Data Fields:

uint32_t

BaudRate configures the Full UART communication baud rate, it can't be 0(bps) and must be smaller than 2950000(bps).

uint32_t

DataBits specifies data bits per transfer, which can be set as:

- **UART_DATA_BITS_5** for 5-bit mode
- **UART_DATA_BITS_6** for 6-bit mode
- **UART_DATA_BITS_7** for 7-bit mode
- **UART_DATA_BITS_8** for 8-bit mode

uint32_t

StopBits specifies the length of stop bit transmission, 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_0_PARITY** for 0 parity
- **UART_1_PARITY** for 1 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:

- **UART_ENABLE_TX** for enabling transmission
- **UART_ENABLE_RX** for enabling reception
- **UART_ENABLE_TX | UART_ENABLE_RX** for enabling both reception and transmission

uint32_t

FlowCtrl Enable or disable the hardware flow control, which can be set as:

- **UART_NONE_FLOW_CTRL** for no flow control
- **UART_CTS_FLOW_CTRL** for enabling CTS flow control
- **UART_CTS_FLOW_CTRL | UART_RTS_FLOW_CTRL** for enabling both CTS and RTS flow control

11.2.4.2 FUART_INTStatus

Data Fields:

uint32_t

All: Full UART interrupt status or mask.

Bit

uint32_t

Reserved: 1 Reserved

uint32_t

CTS: 1 CTS modem interrupt

uint32_t

Reserved: 1 Reserved

uint32_t

Reserved: 1 Reserved

uint32_t

RxFIFO: 1 Receive FIFO interrupt

uint32_t

TxFIFO: 1 Transmit FIFO interrupt

uint32_t

RxTimeout: 1 Receive timeout interrupt

uint32_t

FramingErr: 1 Framing error interrupt

uint32_t

ParityErr: 1 Parity error interrupt

uint32_t

BreakErr: 1 Break error interrupt

uint32_t

OverrunErr: 1 Overrun error interrupt

uint32_t

Reserved: 21 Reserved

11.2.4.3 FUART_AllModemStatus

Data Fields:

uint32_t

All: Full UART All Modem Status

Bit

uint32_t

CTS: 1 CTS modem status

uint32_t

Reserved: 31 Reserved

12. GPIO

12.1 Overview

For TOSHIBA TMPM440 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/TX04_Periph_Driver/src/tmpm440_gpio.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_gpio.h containing the macros, data types, structures and API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- uint8_t GPIO_ReadData (GPIO_Port **GPIO_x**);
- uint8_t GPIO_ReadDataBit (GPIO_Port **GPIO_x**, uint8_t **Bit_x**) ;
- void GPIO_WriteData (GPIO_Port **GPIO_x**, uint8_t **Data**) ;
- void GPIO_WriteDataBit (GPIO_Port **GPIO_x**, uint8_t **Bit_x**, uint8_t **BitValue**) ;
- void GPIO_Init (GPIO_Port **GPIO_x**, uint8_t **Bit_x**,
GPIO_InitTypeDef ***GPIO_InitStruct**);
- void GPIO_SetOutput(GPIO_Port **GPIO_x**, uint8_t **Bit_x**);
- void GPIO_SetInput(GPIO_Port **GPIO_x**, uint8_t **Bit_x**);
- void GPIO_SetOutputEnableReg(GPIO_Port **GPIO_x**, uint8_t **Bit_x**, FunctionalState **NewState**);
- void GPIO_SetInputEnableReg(GPIO_Port **GPIO_x**, uint8_t **Bit_x**, FunctionalState **NewState**);
- void GPIO_SetPullUp(GPIO_Port **GPIO_x**, uint8_t **Bit_x**, FunctionalState **NewState**);
- void GPIO_SetPullDown(GPIO_Port **GPIO_x**, uint8_t **Bit_x**, FunctionalState **NewState**);
- void GPIO_SetOpenDrain (GPIO_Port **GPIO_x**, uint8_t **Bit_x**, FunctionalState **NewState**);
- void GPIO_SetOpenDrain2 (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_WriteDataByte(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t Value);
- void GPIO_ToggleDataByte(GPIO_Port GPIO_x, uint8_t Bit_x);

12.2.2 Detailed Description

Functions listed above can be divided into three parts:

- 1) Write/Read GPIO or GPIO pin are handled by GPIO_ReadData(), GPIO_ReadDataBit(), GPIO_WriteData(), GPIO_WriteDataBit(), GPIO_WriteDataByte() and GPIO_ToggleDataByte().
- 2) Initialize and configure the common functions of each GPIO port are handled by GPIO_SetOutput(), GPIO_SetInput(), GPIO_SetOutputEnableReg(), GPIO_SetInputEnableReg(), GPIO_SetPullUp(), GPIO_SetPullDown(), GPIO_SetOpenDrain() and GPIO_Init().
- 3) GPIO_EnableFuncReg () and GPIO_DisableFuncReg () handle other specified functions.

12.2.3 Function Documentation

12.2.3.1 GPIO_ReadData

Read specified GPIO Data register.

Prototype:

uint8_t

GPIO_ReadData(GPIO_Port **GPIO_x**)

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U

- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAA** : GPIO port AA
- **GPIO_PAB** : GPIO port AB
- **GPIO_PAC** : GPIO port AC
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

Description:

This function will read specified GPIO Data register.

Return:

The value read from DATA register.

12.2.3.2 GPIO_ReadDataBit

Read specified GPIO pin.

Prototype:

```
uint8_t  
GPIO_ReadDataBit(GPIO_Port GPIO_x,  
                  uint8_t Bit_x)
```

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N

- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAA** : GPIO port AA
- **GPIO_PAB** : GPIO port AB
- **GPIO_PAC** : GPIO port AC
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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.

12.2.3.3 GPIO_WriteData

Write specified value to GPIO Data register.

Prototype:

void

```
GPIO_WriteData(GPIO_Port GPIO_x,  
                uint8_t Data)
```

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

Description:

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

Return:

None

12.2.3.4 GPIO_WriteDataBit

Write specified value of single bit to GPIO pin.

Prototype:

void

```
GPIO_WriteDataBit(GPIO_Port GPIO_x,  
                  uint8_t Bit_x,  
                  uint8_t BitValue)
```

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U

- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

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

BitValue: The new value of GPIO pin, which can be set as:

- **GPIO_BIT_VALUE_0**: Clear GPIO pin,
- **GPIO_BIT_VALUE_1**: Set GPIO pin.

Description:

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

Return:

None

12.2.3.5 GPIO_Init

Initialize GPIO port function.

Prototype:

```
void  
GPIO_Init(GPIO_Port GPIO_x,  
          uint8_t Bit_x,  
          GPIO_InitTypeDef * GPIO_InitStruct)
```

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B

- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAA** : GPIO port AA
- **GPIO_PAB** : GPIO port AB
- **GPIO_PAC** : GPIO port AC
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

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

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

Description:

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

Return:

None

12.2.3.6 GPIO_SetOutput

Set specified GPIO pin as output port.

Prototype:

```
void  
GPIO_SetOutput(GPIO_Port GPIO_x,  
                uint8_t Bit_x)
```

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF

- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

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

Description:

This function will set specified GPIO pin as output port.

Return:

None

12.2.3.7 GPIO_SetInput

Set specified GPIO Pin as input port.

Prototype:

void

GPIO_SetInput(GPIO_Port **GPIO_x**,
uint8_t **Bit_x**)

Parameters:

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

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

- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAA** : GPIO port AA
- **GPIO_PAB** : GPIO port AB
- **GPIO_PAC** : GPIO port AC
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

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

Description:

This function will set specified GPIO pin as input port.

Return:

None

12.2.3.8 **GPIO_SetOutputEnableReg**

Enable or disable specified GPIO Pin output function.

Prototype:

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

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

- **GPIO_BIT_0**: GPIO pin 0,
- **GPIO_BIT_1**: GPIO pin 1,
- **GPIO_BIT_2**: GPIO pin 2,
- **GPIO_BIT_3**: GPIO pin 3,
- **GPIO_BIT_4**: GPIO pin 4,
- **GPIO_BIT_5**: GPIO pin 5,
- **GPIO_BIT_6**: GPIO pin 6,
- **GPIO_BIT_7**: GPIO pin 7,

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

NewState:

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

Description:

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

Return:

None

12.2.3.9 GPIO_SetInputEnableReg

Enable or disable specified GPIO Pin input function.

Prototype:

void

GPIO_SetInputEnableReg(GPIO_Port **GPIO_x**,
uint8_t **Bit_x**,
FunctionalState **NewState**)

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R

- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAA** : GPIO port AA
- **GPIO_PAB** : GPIO port AB
- **GPIO_PAC** : GPIO port AC
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

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

NewState:

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

Description:

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

Return:

None

12.2.3.10 GPIO_SetPullUp

Enable or disable specified GPIO Pin pull-up function.

Prototype:

```
void  
GPIO_SetPullUp(GPIO_Port GPIO_x,  
                uint8_t Bit_x,  
                FunctionalState NewState)
```

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAA** : GPIO port AA
- **GPIO_PAB** : GPIO port AB
- **GPIO_PAC** : GPIO port AC
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAH** : GPIO port AH
- **GPIO_PAJ** : GPIO port AJ.

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

- **GPIO_BIT_0**: GPIO pin 0,
- **GPIO_BIT_1**: GPIO pin 1,
- **GPIO_BIT_2**: GPIO pin 2,
- **GPIO_BIT_3**: GPIO pin 3,
- **GPIO_BIT_4**: GPIO pin 4,

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

NewState:

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

Description:

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

Return:

None

12.2.3.11 GPIO_SetPullDown

Enable or disable specified GPIO Pin pull-down function.

Prototype:

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

Parameters:

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

- **GPIO_PG**: GPIO port G.

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

- **GPIO_BIT_2**: GPIO pin 2,
- **GPIO_BIT_ALL**: All GPIO pins can be set.
- Combination of the effective bits.

NewState:

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

Description:

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

Return:

None

12.2.3.12 GPIO_SetOpenDrain

Set specified GPIO Pin as open drain port or CMOS port. with Open Drain Register 1

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_PE** : GPIO port E
- **GPIO_PH** : GPIO port H
- **GPIO_PK** : GPIO port K
- **GPIO_PM** : GPIO port M
- **GPIO_PR** : GPIO port R
- **GPIO_PW** : GPIO port W
- **GPIO_PAJ** : GPIO port AJ.

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_ALL**: All GPIO pins can be set.
- Combination of the effective bits.

NewState:

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

Description:

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

Return:

None

12.2.3.13 GPIO_SetOpenDrain2

Set specified GPIO Pin as open drain port or CMOS port. with Open Drain Register 2

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_PH** : GPIO port H

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_ALL**: All GPIO pins can be set.
- Combination of the effective bits.

NewState:

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

Description:

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

Return:

None

12.2.3.14 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
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAJ** : GPIO port AJ.

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

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

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

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

Description:

This function will enable GPIO pin specified function.

Return:

None

12.2.3.15 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
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H

- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAJ** : GPIO port AJ.

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

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

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

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

Description:

This function will disable GPIO pin specified function.

Return:

None

12.2.3.16 GPIO_WriteDataByte

Set a value to a specified bit of GPIO DATA register.

Prototype:

void

GPIO_WriteDataByte(GPIO_Port **GPIO_x**, uint8_t **Bit_x**, uint8_t **Value**)

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF
- **GPIO_PAG** : GPIO port AG
- **GPIO_PAJ** : GPIO port AJ.

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

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

- Combination of the effective bits.

Value: byte data. enable bit which is specified in bits.

Description:

Set a value to a specified bit of GPIO DATA register.

Return:

None

12.2.3.17 GPIO_ToggleDataByte

Set a value to a specified bit of GPIO DATA register.

Prototype:

void

GPIO_ToggleDataByte(GPIO_Port **GPIO_x**, uint8_t **Bit_x**)

Parameters:

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

- **GPIO_PA** : GPIO port A
- **GPIO_PB** : GPIO port B
- **GPIO_PC** : GPIO port C
- **GPIO_PD** : GPIO port D
- **GPIO_PE** : GPIO port E
- **GPIO_PF** : GPIO port F
- **GPIO_PG** : GPIO port G
- **GPIO_PH** : GPIO port H
- **GPIO_PJ** : GPIO port J
- **GPIO_PK** : GPIO port K
- **GPIO_PL** : GPIO port L
- **GPIO_PM** : GPIO port M
- **GPIO_PN** : GPIO port N
- **GPIO_PP** : GPIO port P
- **GPIO_PR** : GPIO port R
- **GPIO_PT** : GPIO port T
- **GPIO_PU** : GPIO port U
- **GPIO_PV** : GPIO port V
- **GPIO_PW** : GPIO port W
- **GPIO_PY** : GPIO port Y
- **GPIO_PAD** : GPIO port AD
- **GPIO_PAE** : GPIO port AE
- **GPIO_PAF** : GPIO port AF

- **GPIO_PAG** : GPIO port AG
- **GPIO_PAJ** : GPIO port AJ.

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

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

Description:

Toggle a value to a specified bit of GPIO DATA register.

Return:

None

12.2.4 Data Structure Description

12.2.4.1 GPIO_InitTypeDef

Data Fields:

uint8_t

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

- **GPIO_INPUT**: Set GPIO pin as input port
- **GPIO_OUTPUT**: Set GPIO pin as output port
- **GPIO_IO_MODE_NONE**: Don't change GPIO pin I/O mode.

uint8_t

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

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

uint8_t

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

- **GPIO_OPEN_DRAIN_ENABLE**: Set specified GPIO pin as open drain port.
- **GPIO_OPEN_DRAIN_DISABLE**: Set specified GPIO pin as CMOS port.

➤ **GPIO_OPEN_DRAIN_NONE:** Don't have open-drain function or needn't change.
uint8_t

OpenDrain2 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.

13. KSCAN

13.1 Overview

TOSHIBA TPM440 has one key scan channel. The key matrix scan contains the following features:

- 1) KSCAN inputs/outputs are 8 channels in each.
A 64- (8×8) key matrix at the maximum is controlled by software.
- 2) Chattering cancel can be controlled.
- 3) Scan results can be read.
- 4) It operates on low frequency oscillation (fs).

13.2 API Functions

13.2.1 Function List

- ◆ void KSCAN_Enable(void)
- ◆ void KSCAN_Disable(void)
- ◆ void KSCAN_SetSCLK(uint8_t ksclk)
- ◆ void KSCAN_SetInputCtrlMask(uint8_t InputCH, FunctionalState NewState)
- ◆ void KSCAN_SetOutputCtrl(uint8_t OutputCH, FunctionalState NewState)
- ◆ void KSCAN_SetStrobeOutput(uint8_t cycle)
- ◆ void KSCAN_Start(void)
- ◆ void KSCAN_Stop(void)
- ◆ uint8_t KSCAN_ReadStart(void)
- ◆ void KSCAN_KSBRInitial(void)
- ◆ void KSCAN_SWReset(void)
- ◆ void KSCAN_SetConsecutiveMatches(uint8_t match)
- ◆ void KSCAN_SetChatCancelTime(uint8_t timeN)
- ◆ void KSCAN_SetStrobeWidth(uint8_t StrobeWidth)
- ◆ uint8_t KSCAN_ReadBufChecked(uint8_t output)
- ◆ void KSCAN_MaskReadBuf(uint32_t MaskCH, uint32_t MaskBit, FunctionalState NewState)
- ◆ void KSCAN_SetINTReq(FunctionalState NewState)

13.2.2 Detailed Description

Functions listed above can be divided into four parts:

- 1) Enable or disable KSCAN function

- KSCAN_Enable (), KSCAN_Disable (),KSCAN_Start(),KSCAN_Stop(),
KSCAN_ReadStart()
- 2) Configure the KSCAN function
KSCAN_SetSCLK(), KSCAN_SetInputCtrl(), KSCAN_SetOutputCtrl(),
KSCAN_SetConsecutiveMatches(),KSCAN_SetChatCancelTime(),
KSCAN_SetStrobeWidth(),KSCAN_ReadBufChecked(),
KSCAN_MaskReadBuf().KSCAN_SetStrobeOutput().
- 3) KSCAN reset
KSCAN_KSBRInitial(), KSCAN_SWReset().
- 4) For KSCAN interrupt
KSCAN_SetINTReq ()

13.2.3 Function Documentation

13.2.3.1 KSCAN_Enable

Enable KSCAN function.

Prototype:

void KSCAN_Enable(void)

Parameters:

None

Description:

Enable KSCAN function.

Return:

None

13.2.3.2 KSCAN_Disable

DisableKSCAN function.

Prototype:

void KSCAN_Disable (void)

Parameters:

None

Description:

Disable KSCAN function.

Return:

None

13.2.3.3 KSCAN_SetSCLK

Selects KSCAN operation clock (ksclk).

Prototype:

Void:

KSCAN_SetSCLK(uint8_t **ksclk**)

Parameters:

Ksclk: KSCAN operation clock.

- **KSCAN_KSCL_FS** : Set KSCAN operation clock as FS
- **KSCAN_KSCL_TBOUT**: Set KSCAN operation clock as TBOUT

Description:

Select KSCAN operation clock.

Return:

None

13.2.3.4 KSCAN_SetInputCtrl

Set KSCAN input control.

Prototype:

Void

KSCAN_SetInputCtrl (uint8_t **InputCH**, FunctionalState **NewState**)

Parameters:

InputCH: Select which input channel to set.

- **KSCAN_CH_0**: KSCAN Channel 0
- **KSCAN_CH_1**: KSCAN Channel 1
- **KSCAN_CH_2**: KSCAN Channel 2
- **KSCAN_CH_3**: KSCAN Channel 3
- **KSCAN_CH_4**: KSCAN Channel 4
- **KSCAN_CH_5**: KSCAN Channel 5
- **KSCAN_CH_6**: KSCAN Channel 6
- **KSCAN_CH_7**: KSCAN Channel 7
- **KSCAN_CH_ALL**: All KSCAN Channels
- Combination of the effective Channels.

NewState : New state of mask keyscan input

- **ENABLE** : Enable mask keyscan input
- **DISABLE**: Disable mask keyscan input

Description:

Set KSCAN input control.

Return:

None

13.2.3.5 KSCAN_SetOutputCtrl

Set KSCAN output control.

Prototype:

Void

KSCAN_SetOutputCtrl (uint8_t **OutputCH**, FunctionalState **NewState**)

Parameters:

OutputCH: Select which output channel to set.

- **KSCAN_CH_0**: KSCAN Channel 0
- **KSCAN_CH_1**: KSCAN Channel 1
- **KSCAN_CH_2**: KSCAN Channel 2
- **KSCAN_CH_3**: KSCAN Channel 3
- **KSCAN_CH_4**: KSCAN Channel 4
- **KSCAN_CH_5**: KSCAN Channel 5
- **KSCAN_CH_6**: KSCAN Channel 6
- **KSCAN_CH_7**: KSCAN Channel 7
- **KSCAN_CH_ALL**: All KSCAN Channels
- Combination of the effective Channels.

NewState: Set the output type.

- **ENABLE** : keyscan output High
- **DISABLE**: keyscan output low

Description:

Set KSCAN output control.

Return:

None

13.2.3.6 KSCAN_SetStrobeOutput

Set the strobe output frame time of KSCAN outputs.

Prototype:

Void

KSCAN_SetStrobeOutput(uint8_t *cycle*)

Parameters:

cycle: set strobe width cycle.

This data can be set from 1 to 255.

Description:

Set the strobe output frame time of KSCAN outputs.

Return:

None

13.2.3.7 KSCAN_Start

Start key strobe outputs and counters.

Prototype:

void

KSCAN_Start(void)

Parameters:

None

Description:

start key strobe outputs and counters

Return:

None

13.2.3.8 KSCAN_Stop

Stop key strobe outputs and counters

Prototype:

void

KSCAN_Stope(void)

Parameters:

None

Description:

Stop key strobe outputs and counters

Return:

None

13.2.3.9 KSCAN_ReadStart

Read the kscan start condition.

Prototype:

void
KSCAN_Start(void)

Parameters:

None

Description:

Read the kscan start condition.

Return:

start condition

13.2.3.10 KSCAN_KSBRInitial

Initializes KSBR to 0

Prototype:

void
KSCAN_KSBRInitial(void)

Parameters:

None

Description:

Initializes KSBR to 0

Return:

None

13.2.3.11 KSCAN_SWReset

KSCAN software reset.

Prototype:

void
KSCAN_SWReset(void)

Parameters:

None

Description:

KSCAN software reset.

Return:

None

13.2.3.12 KSCAN_SetConsecutiveMatches

Set consecutive matches number for key on/off determination.

Prototype:

Void
KSCAN_SetConsecutiveMatches(uint8_t *match*)

Parameters:

Match: The consecutive match number.

- **KSCAN_MATCH_2C:** 2 consecutive matches
- **KSCAN_MATCH_4C:** 4 consecutive matches

Description:

Set consecutive matches number for key on/off determination.

Return:

None

13.2.3.13 KSCAN_SetChatCancelTime

Set a chattering cancel time: $16/\text{ksclk} * \text{timeN}$

Prototype:

Void

KSCAN_SetChatCancelTime(uint8_t *timeN*)

Parameters:

timeN: a chattering cancel time index.

This parameter can be 0 to 255.

Description:

Set a chattering cancel time: $16/\text{ksclk} * \text{timeN}$.

Return:

None

13.2.3.14 KSCAN_SetStrobeWidth

Set a strobe width.

Prototype:

Void

KSCAN_SetStrobeWidth(uint8_t *StrobeWidth*)

Parameters:

StrobeWidth: strobe width data.

This parameter can be 0 to 5.

Description:

Set a strobe width.

Return:

None

13.2.3.15 KSCAN_ReadBufChecked

A read result of KSCAN output can be checked.

Prototype:

Void

KSCAN_ReadBufChecked(uint8_t *output*)

Parameters:

output: Read this output.

This parameter can be 0 to 7.

Description:

A read result of KSCAN output can be checked.

Return:

None

13.2.3.16 KSCAN_MaskReadBuf

Masks read buffer of corresponding KSCAN output.

Prototype:

Void

KSCAN_MaskReadBuf(uint32_t **MaskCH**, uint32_t **MaskBit**, FunctionalState **NewState**)

Parameters:

MaskCH: Set this output to mask read buffer.

This parameter can be 0 to 7.

MaskBit: Set the bit of output to masked or not masked.

This parameter can be 1 to 255.

NewState: New state of KSCAN read buffer.

This parameter can be ENABLE (Masked) or DISABLE(Not masked).

Description:

Masks read buffer of corresponding KSCAN output.

Return:

None

13.2.3.17 KSCAN_SetINTRReq

Specifies a KSCAN interrupt request.

Prototype:

Void

KSCAN_SetINTRReq (FunctionalState **NewState**)

Parameters:

NewState : New state of KSCAN Interrupt request.

- **ENABLE** : Interrupt are not masked
- **DISABLE**: Interrupt are masked

Description:

Set KSCAN input control.

Return:

None

13.2.4 Data Structure Description

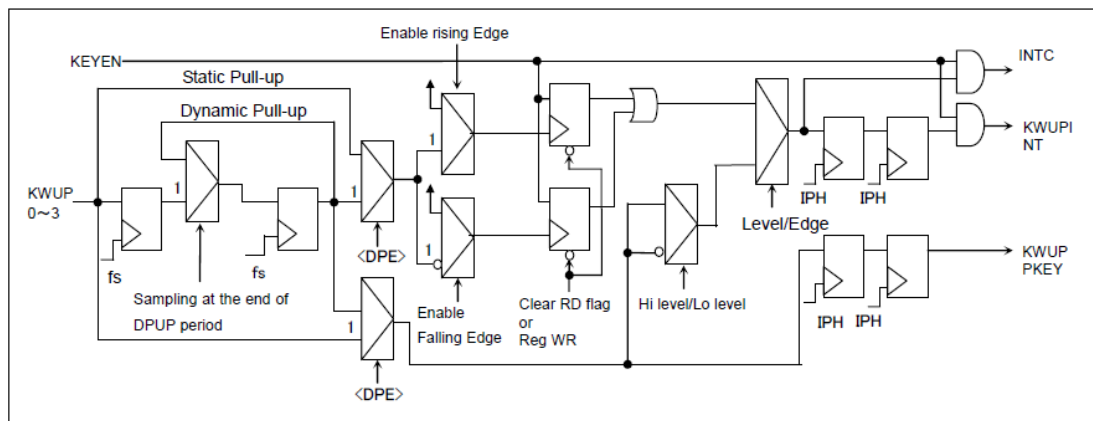
None

14. KWUP

14.1 Overview

TOSHIBA TMPM440 has two units of KWUP: 32 key inputs and 8 key inputs, which can be used for releasing the STOP mode or for external interrupts.

The following figure is KEY ON WAKE UP Block Diagram.



KEY ON WAKE UP Block Diagram

14.2 API Functions

14.2.1 Function List

- ◆ void KWUP_SetConfig(TSB_KWUP_TypeDef * **TSB_KWUPx**, KWUP_SettingTypeDef * **Settings**);
- ◆ KWUP_PortStatus KWUP_GetPortStatus(TSB_KWUP_TypeDef * **TSB_KWUPx**);
- ◆ void KWUP_SetPullUpConfig(TSB_KWUP_TypeDef * **TSB_KWUPx**, KWUP_PullUpCycles **T1**, KWUP_PullUpCycles **T2**);
- ◆ KWUP_INTStatus KWUP_GetINTStatus(TSB_KWUP_TypeDef * **TSB_KWUPx**);

14.2.2 Detailed Description

Functions listed above can be divided into two parts:

- 1) Configure KWUP settings Dynamic pull-up cycle and Dynamic pull-up period.
KWUP_SetConfig (), KWUP_SetPullUpConfig ().
- 2) Get PortStatus and int status.
KWUP_GetPortStatus (). KWUP_GetINTStatus().

14.2.3 Function Documentation

14.2.3.1 KWUP_SetConfig

Configure the key wake-up settings.

Prototype:

void

KWUP_SetConfig(TSB_KWUP_TypeDef * **TSB_KWUPx**, KWUP_SettingTypeDef * **Settings**);

Parameters:

TSB_KWUPx: Select the key wake-up unit.

The value could be the following values:

- **TSB_KWUPA** select unit A.
- **TSB_KWUPB** select unit B.

Settings: the pointer to a structure containing the key wake-up settings.

Description:

configure the key wake-up settings.

Return: None

14.2.3.2 KWUP_GetPortStatus

get port status sampled in dynamic pull-up period

Prototype:

KWUP_PortStatus

KWUP_GetPortStatus(TSB_KWUP_TypeDef * **TSB_KWUPx**);

Parameters:

TSB_KWUPx: Select the key wake-up unit.

The value could be the following values:

- **TSB_KWUPA** select unit A.
- **TSB_KWUPB** select unit B.

Description:

This function will get port status sampled in dynamic pull-up period

Return:

None

14.2.3.3 KWUP_SetPullUpConfig

Set Dynamic pull-up cycle and Dynamic pull-up period.

Prototype:

void

KWUP_SetPullUpConfig(TSB_KWUP_TypeDef * **TSB_KWUPx**,
KWUP_PullUpCycles **T1**, KWUP_PullUpCycles **T2**)

Parameters:

TSB_KWUPx: Select the key wake-up unit.

The value could be the following values:

- **TSB_KWUPA** select unit A.
- **TSB_KWUPB** select unit B.

T1: activating period when using dynamic pull-up.

- **KWUP_CYCLES_2_FS** pull-up period 2/fs,
- **KWUP_CYCLES_4_FS** pull-up period 4/fs,
- **KWUP_CYCLES_8_FS** pull-up period 8/fs,
- **KWUP_CYCLES_16_FS** pull-up period 16/fs,

T2: repeated dynamic pull-up operation cycles.

- **KWUP_CYCLES_256_FS** pull-up cycle 256/fs,
- **KWUP_CYCLES_512_FS** pull-up cycle 512/fs,
- **KWUP_CYCLES_1024_FS** pull-up cycle 1024/fs,
- **KWUP_CYCLES_2048_FS** pull-up cycle 2048/fs,

Description:

This function will Set the Dynamic pull-up cycle and Dynamic pull-up period of the port.

Return:

None

14.2.3.4 KWUP_GetINTStatus

get key interrupt status

Prototype:

KWUP_INTStatus

KWUP_GetINTStatus(TSB_KWUP_TypeDef * **TSB_KWUPx**);

Parameters:

TSB_KWUPx: Select the key wake-up unit.

The value could be the following values:

- **TSB_KWUPA** select unit A.
- **TSB_KWUPB** select unit B.

Description:

This function will get key interrupt status.

Return:

None

14.2.4 Data Structure Description

14.2.4.1 KWUP_SettingTypeDef

Data Fields:

KWUP_Input

KeyN key input select:

- **KWUP_INPUT_0** :select key input number 0,
- **KWUP_INPUT_1** :select key input number 1,
- **KWUP_INPUT_2** :select key input number 2
- **KWUP_INPUT_3** :select key input number 3,
- **KWUP_INPUT_4** :select key input number 4,
- **KWUP_INPUT_5** :select key input number 5,
- **KWUP_INPUT_6** :select key input number 6,
- **KWUP_INPUT_7** :select key input number 7,
- **KWUP_INPUT_8** :select key input number 8,
- **KWUP_INPUT_9** :select key input number 9,
- **KWUP_INPUT_10** :select key input number 10,
- **KWUP_INPUT_11** :select key input number 11,
- **KWUP_INPUT_12** :select key input number 12,
- **KWUP_INPUT_13** :select key input number 13,
- **KWUP_INPUT_14** :select key input number 14,
- **KWUP_INPUT_15** :select key input number 15,
- **KWUP_INPUT_16** :select key input number 16,
- **KWUP_INPUT_17** :select key input number 17,
- **KWUP_INPUT_18** :select key input number 18,
- **KWUP_INPUT_19** :select key input number 19,
- **KWUP_INPUT_20** :select key input number 20,
- **KWUP_INPUT_21** :select key input number 21,
- **KWUP_INPUT_22** :select key input number 22,
- **KWUP_INPUT_23** :select key input number 23,
- **KWUP_INPUT_24** :select key input number 24,
- **KWUP_INPUT_25** :select key input number 25,
- **KWUP_INPUT_26** :select key input number 26,
- **KWUP_INPUT_27** :select key input number 27,
- **KWUP_INPUT_28** :select key input number 28,
- **KWUP_INPUT_29** :select key input number 29,
- **KWUP_INPUT_30** :select key input number 30,
- **KWUP_INPUT_31** :select key input number 31,

KWUP_PullUpCtrl

PullUpCtrl selected static pull-up or dynamic pull-up.

- **KWUP_PUP_CTRL_BY_STATIC** : selected static pull-up,
- **KWUP_PUP_CTRL_BY_DYNAMIC** : selected dynamic pull-up,

KWUP_ActiveState

ActiveState active status setting:

- **KWUP_ACTIVE_BY_L_LEVEL** : Sets "Low" level active condition,
- **KWUP_ACTIVE_BY_H_LEVEL** : Sets "High" level active condition,

- **KWUP_ACTIVE_BY_RISING_EDGE** : Sets "Rising" level active condition,
- **KWUP_ACTIVE_BY_FALLING_EDGE**: Sets "Falling" level active condition,
- **KWUP_ACTIVE_BY_BOTH_EDGES**: Sets "Both edges" level active condition,

FunctionalState

INTNewState disable /enable KWUP interrupt input.

- **DISABLE**: disable KWUP interrupt input
- **ENABLE**: enable KWUP interrupt input

15. PHC

15.1 Overview

TOSHIBA TMPM440 is two-phase pulse input counters. The counter is increment or decremented by one depending on the state transition of the two phase pulse that is input through PHCxIN0 and PHCxIN1 and has phase difference. The noise filter can be enabled or disabled.

Two-phase pulse input counter has three operation mode, they are controlled by register.

1. Normal operation mode (Counts up/down at the 4th count)
2. 4-fold multiplication mode (Counts up/down at all counts)
3. Twofold multiplication mode (Counts up by the input of PHCxIN0 or PHCxIN1)

The PHC API provides a set of functions for using the TMPM440 PHC modules. It includes PHC channel set, mode set, noise filter set, interrupt set, PHC status read, PHC result value read and so on.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_phc.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_phc.h containing the macros, data types, structures and API definitions for use by applications.

15.2 API Functions

15.2.1 Function List

- ◆ void PHC_Enable(TSB_PHC_TypeDef * **PHCx**);
- ◆ void PHC_Disable(TSB_PHC_TypeDef * **PHCx**);
- ◆ void PHC_SetRunState(TSB_PHC_TypeDef * **PHCx**, uint32_t **Cmd**);
- ◆ void PHC_Init(TSB_PHC_TypeDef * **PHCx**, PHC_InitTypeDef * **InitStruct**);
- ◆ PHC_INTFactor PHC_GetINTFactor(TSB_PHC_TypeDef * **PHCx**);
- ◆ void PHC_ClearINTFactor(TSB_PHC_TypeDef * **PHCx**, uint32_t **ClearINT**);
- ◆ void PHC_EnableInterrupt(TSB_PHC_TypeDef * **PHCx**, uint32_t **EnableINT**);
- ◆ void PHC_DisableInterrupt(TSB_PHC_TypeDef * **PHCx**, uint32_t **DisableINT**);
- ◆ uint16_t PHC_GetPulseCntValue(TSB_PHC_TypeDef * **PHCx**);
- ◆ void PHC_ClearPulseCntValue(TSB_PHC_TypeDef * **PHCx**);
- ◆ uint16_t PHC_GetCompareValue(TSB_PHC_TypeDef * **PHCx**, uint8_t **CmpReg**);
- ◆ void PHC_SetCompareValue(TSB_PHC_TypeDef * **PHCx**, uint8_t **CmpReg**, uint16_t **CmpValue**);

- ◆ void PHC_SetDMAReq(TSB_PHC_TypeDef * **PHCx**, FunctionalState **NewState**, uint8_t **DMAReq**);

15.2.2 Detailed Description

Functions listed above can be divided into three parts:

- 1) Configure and control the common functions of each PHC channel are handled by the PHC_Enable (),PHC_Disable (),PHC_Init() and void PHC_SetRunState().
- 2) The status indication of each PHC channel is handled by PHC_GetINTFactor(), PHC_GetPulseCntValue(), PHC_GetCompareValue()
- 3) PHC_ClearINTFactor(), PHC_EnableInterrupt(), PHC_DisableInterrupt(), , PHC_ClearPulseCntValue(),PHC_SetCompareValue() and PHC_SetDMAReq() handle other specified functions.

15.2.3 Function Documentation

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

TSB_PCH0, TSB_PCH1

15.2.3.1 PHC_Enable

Enable the specified PHC channel.

Prototype:

void
PHC_Enable(TSB_PHC_TypeDef* **PHCx**)

Parameters:

PHCx is the specified PHC channel.

Description:

This function enables the specified PHC channel selected by **PHCx**.

Return:

None

15.2.3.2 PHC_Disable

Disable the specified PHC channel.

Prototype:

void

PHC_Disable(TSB_PHC_TypeDef* **PHCx**)

Parameters:

PHCx is the specified PHC channel.

Description:

This function disables the specified PHC channel selected by **PHCx**.

Return:

None

15.2.3.3 PHC_SetRunState

Start or stop of the specified PHC channel.

Prototype:

```
void  
PHC_SetRunState(TSB_PHC_TypeDef * PHCx,  
                uint32_t Cmd);
```

Parameters:

PHCx is the specified PHC channel.

Cmd The command for the up-and-down counter

- **PHC_RUN** for start counter
- **PHC_STOP** for stop counter

Description:

The up-and-down counter of the specified PHC channel starts counting if **Cmd** is **PHC_RUN** and up-and-down counter stops counting and the value in up-and-down counter register is clear if **Cmd** is **PHC_STOP**.

Return:

None

15.2.3.4 PHC_Init

Initialize the specified PHC channel.

Prototype:

```
void  
PHC_Init (TSB_PHC_TypeDef * PHCx,
```

```
PHC_InitTypeDef * InitStruct);
```

Parameters:

PHCx is the specified PHC channel.

InitStruct is the structure containing basic PHC configuration including count mode, noise filter control and counter clear control (refer to “Data Structure Description” for details).

Description:

This function initializes the specified PHC channel selected by **PHCx**.

Return:

None

15.2.3.5 PHC_GetINTFactor

Indicate what causes the interrupt.

Prototype:

```
PHC_INTFactor
```

```
PHC_GetINTFactor(TSB_PHC_TypeDef* PHCx)
```

Parameters:

PHCx is the specified PHC channel.

Description:

This function should be used in ISR to indicate the factor of interrupt. Bit of **Compare0** indicates if the counter matches with compare register0, bit of **Compare1** indicates if the counter matches with compare register1, bit of **Overflow** indicates if overflow had occurred before the interrupt, and bit of **Underflow** indicates if underflow had occurred before the interrupt.

Return:

PHC Interrupt factor. Each bit has the following meaning:

Compare0 (Bit0): Specified PHC channel detected a match with the compare register0

Compare1 (Bit1): Specified PHC channel detected a match with the compare register1

Overflow (Bit2): Specified PHC channel detected counter is overflow

Underflow (Bit3): Specified PHC channel detected counter is underflow

15.2.3.6 PHC_ClearINTFactor

Clear specified interrupt factor flag.

Prototype:

void

```
PHC_ClearINTFactor(TSB_PHC_TypeDef * PHCx,  
                  uint32_t ClearINT)
```

Parameters:

PHCx is the specified PHC channel.

ClearINT Select the clear of PHC interrupt factor. This parameter can be:

- **PHC_FLG_CMP0**: Clears the interrupt factor which monitors compare register0 match event.
- **PHC_FLG_CMP1**: Clears the interrupt factor which monitors compare register1 match event.
- **PHC_FLG_OVERFLOW**: Clears the interrupt factor which monitors overflow event.
- **PHC_FLG_UNDERFLOW**: Clears the interrupt factor which monitors underflow event.
- **PHC_FLG_ALL**: All interrupt factors can be cleared.
- Combination of effective interrupt factor.

Description:

Because of each interrupt factor is not cleared automatically.

This function is used to clear the specified interrupt factor.

Return:

None

Note:

Each interrupt factor is not cleared automatically, initialize before using them.

15.2.3.7 PHC_EnableInterrupt

Enable specified PHC interrupt

Prototype:

void

```
PHC_EnableInterrupt(TSB_PHC_TypeDef * PHCx,  
                   uint32_t EnableINT);
```

Parameters:

PHCx is the specified PHC channel.

EnableINT Selects the clear of PHC interrupt. This parameter can be:

- **PHC_CR_INT_COMP0**: Enable INTPHTx0 interrupt, which occurred if the counter matches with compare register0. (x can be 0,1,2,3)
- **PHC_CR_INT_COMP1**: Enable INTPHTx1 interrupt, which occurred if the counter matches with compare register1. (x can be 0,1,2,3)
- **PHC_CR_INT_COMP0_AND_1**: Enable both INTPHTx0 interrupt and INTPHTx1 interrupt (x can be 0,1,2,3)
- **PHC_CR_INT EVERY**: Enable INTPHEVRYx interrupt, which occurred if the counter value is changed. (x can be 0,1,2,3)
- **PHC_CR_INT_ALL**: All interrupt can be enabled
- Combination of effective PHC interrupt.

Description:

If **PHC_CR_INT_COMP0** is selected, the interrupt of the specified PHC channel INTPHTx0 happen when the value in counter and compare register0 are match.

If **PHC_CR_INT_COMP1** is selected, the interrupt of the specified PHC channel INTPHTx1 happen when the value in counter and compare register1 are match.

If **PHC_CR_INT_COMP0_AND_1**. the interrupt of the specified PHC channel INTPHTx0 happen when the value in counter and compare register0 are match, and the interrupt of the specified PHC channel INTPHTx1 happen when the value in counter and compare register1 are match.

If **PHC_CR_INT EVERY**: is selected, the interrupt of the specified PHC channel INTPHEVRYx happen when the value in counter is changed.

If **PHC_CR_INT_ALL**: is selected, all above interrupt of the specified PHC channel is enabled.

Return:

None

15.2.3.8 PHC_DisableInterrupt

Disable specified PHC interrupt

Prototype:

void

```
PHC_DisableInterrupt(TSB_PHC_TypeDef * PHCx,  
                     uint32_t DisableINT);
```

Parameters:

PHCx is the specified PHC channel.

DisableINT Select the clear of PHC interrupt. This parameter can be:

- **PHC_CR_INT_COMP0**: Disables INTPHTx0 interrupt, which occurred if the counter matches with compare register0. (x can be 0,1,2,3)
- **PHC_CR_INT_COMP1**: Disables INTPHTx1 interrupt, which occurred if the counter matches with compare register1. (x can be 0,1,2,3)
- **PHC_CR_INT_COMP0_AND_1**: Disables both INTPHTx0 interrupt and INTPHTx1 interrupt (x can be 0,1,2,3)
- **PHC_CR_INT EVERY**: Disables INTPHEVRYx interrupt, which occurred if the counter value is changed. (x can be 0,1,2,3)
- **PHC_CR_INT_ALL**: All interrupt can be disabled
- Combination of effective PHC interrupt.

Description:

If **PHC_CR_INT_COMP0** is selected, the interrupt of the specified PHC channel INTPHTx0 is disabled (x can be 0,1,2,3)

If **PHC_CR_INT_COMP1** is selected, the interrupt of the specified PHC channel INTPHTx1 is disabled (x can be 0,1,2,3)

If **PHC_CR_INT_COMP0_AND_1** is selected, the interrupt of the specified PHC channel INTPHTx0 and INTPHTx1 are disabled (x can be 0,1,2,3)

If **PHC_CR_INT EVERY**: is selected, the interrupt of the specified PHC channel INTPHEVRYx is disabled (x can be 0,1,2,3)

If **PHC_CR_INT_ALL**: is selected, all above interrupt of the specified PHC channel is disabled.

Return:

None

15.2.3.9 PHC_GetPulseCntValue

Get the counter value of specified PHC channel

Prototype:

uint16_t

PHC_GetPulseCntValue(TSB_PHC_TypeDef * **PHCx**);

Parameters:

PHCx is the specified PHC channel.

Description:

This function returns the value in counter of the specified PHC channel

Return:

The value of PHC counter

15.2.3.10 PHC_ClearPulseCntValue

Clear the counter value of specified PHC channel

Prototype:

void

PHC_ClearPulseCntValue(TSB_PHC_TypeDef * **PHCx**);

Parameters:

PHCx is the specified PHC channel.

Description:

This function clears the value in counter of the specified PHC channel

Return:

None

Note:

Pulse is counted not synchronized with MCU operation clock. Because there is a possibility that data is read while rewriting, depending on the timing, reading out twice is recommended. In this case if the data is different read out data again.

PHC counter is an up-and-down counter.

The counter value is **0x7FFF** after PHC_ClearPulseCntValue function is called.

15.2.3.11 PHC_GetCompareValue

Get the value of compare register0 or compare register1 of the specified PHC channel.

Prototype:

uint16_t

PHC_GetCompareValue(TSB_PHC_TypeDef * **PHCx**,
uint8_t **CmpReg**)

Parameters:

PHCx is the specified PHC channel.

CmpReg is used to choose to return the value of compare register0 or to return the value of compare register1, which can be one of the following,

- **PHC_COMP_0**: specifying compare register0.
- **PHC_COMP_1**: specifying compare register1.

Description:

This function returns the value of compare register0 of the specified PHC channel if **CmpReg** is **PHC_COMP_0**, and returns value of compare register1 of the specified PHC channel if **CmpReg** is **PHC_COMP_1**

Return:

The compare value

15.2.3.12 PHC_SetCompareValue

Set the value of compare register0 or compare register1 of the specified PHC channel.

Prototype:

```
void  
PHC_SetCompareValue(TSB_PHC_TypeDef * PHCx,  
                    uint8_t CmpReg,  
                    uint16_t CmpValue);
```

Parameters:

PHCx is the specified PHC channel.

CmpReg is used to choose to set the value of compare register0 or to set the value of compare register1, which can be one of the following,

- **PHC_COMP_0**: specifying compare register0.
- **PHC_COMP_1**: specifying compare register1.

CmpValue is the value to set to compare register

Description:

This function sets **CmpValue** to compare register0 of the specified PHC channel if **CmpReg** is **PHC_COMP_0**, and sets **CmpValue** to compare register1 of the specified PHC channel if **CmpReg** is **PHC_COMP_1**

Return:

None

15.2.3.13 PHC_SetDMAReq

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

Prototype:

```
void  
PHC_SetDMAReq(TSB_PHC_TypeDef * PHCx,
```

FunctionalState **NewState**,
uint8_t **DMAReq**);

Parameters:

PHCx is the specified PHC channel.

NewState specifies specified DMA monitor state of specified PHC channel, which can be:

- **ENABLE:** enables specified DMA request
- **DISABLE:** disables specified DMA request

DMAReq specifies DMA request of the external inputs, which can be

- **PHC_DMA_REQ_CAPTURE_2:** Select DMA request: input capture2.

Description:

This function enables or disables the selected DMA request for the specified PHC channel.

Return:

None

Note:

When interrupt request is disabled, DMA request is not occurred even if DMA request is enabled.

15.2.4 Data Structure Description

15.2.4.1 PHC_InitTypeDef

Data Fields:

uint32_t

Mode selects PHC working mode , which can be set as:

- **PHC_CR_MODE_NORMAL:** counter is increment or decrement when the state transition of the two-phase pulse changes at fourth count
- **PHC_CR_MODE_4TIMES:** counter is increment or decrement when the state transition of the two-phase pulse changes once
- **PHC_CR_MODE_2TIMES_IN0:** counter is increment when the state transition of PHCxIN0 once
- **PHC_CR_MODE_2TIMES_IN1:** counter is increment when the state transition of PHCxIN1 changes once

uint32_t

NoiseFilterCtrl enables and disables the noise filter, which can be set as:

- **PHC_CR_NOISEFILTER_ON**: enables the noise filter
- **PHC_CR_NOISEFILTER_OFF**: disables the noise filter

uint32_t

CountClearCtrl clears or do nothing with the PHC up-and-down counter, which can be set as:

- **PHC_COUNT_CONTINUE**: Do nothing with the PHC up-and-down counter.
- **PHC_COUNT_CLR**: Clears the PHC up-and-down counter

15.2.4.2 PHC_INTFactor

Data Fields:

uint32_t

All: PHC interrupt factor.

Bit

uint32_t

Compare0: 1 a match with the compare register0 is detected

uint32_t

Compare1: 1 a match with the compare register1 is detected

uint32_t

Overflow: 1 an up-and-down counter is overflow

uint32_t

UnderFlow: 1 an up-and-down counter is underflow

uint32_t

Reserverd: 28 Reserverd

16. RTC

16.1 Overview

The Real Time Clock (RTC) in the TPM440 has such functions as follow:

- Clock (hour, minute and second)
- Calendar (month, week, date and leap year)
- Selectable 12 (am/ pm) and 24 hour display
- Time adjustment +/- 30 seconds (by software)
- Alarm function (Alarm interrupt, ALARM pin)

The RTC driver APIs provide a set of functions to configure RTC clock and alarm, including such common parameters as year, leap year, month, date, day, hour, hour mode, minute and second and so on.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_rtc.c, with /Libraries/ TX04_Periph_Driver/inc/tmpm440_rtc.h containing the macros, data types, structures and API definitions for use by applications.

16.2 API Functions

16.2.1 Function List

- ◆ void RTC_SetSec(uint8_t **Sec**);
- ◆ uint8_t RTC_GetSec(void);
- ◆ void RTC_SetMin(RTC_FuncMode **NewMode**, uint8_t **Min**);
- ◆ uint8_t RTC_GetMin(RTC_FuncMode **NewMode**);
- ◆ uint8_t RTC_GetAMPM(RTC_FuncMode **NewMode**);
- ◆ void RTC_SetHour24(RTC_FuncMode **NewMode**, uint8_t **Hour**);
- ◆ void RTC_SetHour12(RTC_FuncMode **NewMode**, uint8_t **Hour**, uint8_t **AmPm**);
- ◆ uint8_t RTC_GetHour(RTC_FuncMode **NewMode**);
- ◆ void RTC_SetDay(RTC_FuncMode **NewMode**, uint8_t **Day**);
- ◆ uint8_t RTC_GetDay(RTC_FuncMode **NewMode**);
- ◆ void RTC_SetDate(RTC_FuncMode **NewMode**, uint8_t **Date**);
- ◆ uint8_t RTC_GetDate(RTC_FuncMode **NewMode**);
- ◆ void RTC_SetMonth(uint8_t **Month**);
- ◆ uint8_t RTC_GetMonth(void);
- ◆ void RTC_SetYear(uint8_t **Year**);
- ◆ uint8_t RTC_GetYear(void);
- ◆ void RTC_SetHourMode(uint8_t **HourMode**);

- ◆ uint8_t RTC_GetHourMode(void);
- ◆ void RTC_SetLeapYear(uint8_t **LeapYear**);
- ◆ uint8_t RTC_GetLeapYear(void);
- ◆ void RTC_SetTimeAdjustReq(void);
- ◆ RTC_ReqState RTC_GetTimeAdjustReq(void);
- ◆ void RTC_EnableClock(void);
- ◆ void RTC_DisableClock(void);
- ◆ void RTC_EnableAlarm(void);
- ◆ void RTC_DisableAlarm(void);
- ◆ void RTC_SetRTCINT(FunctionalState **NewState**);
- ◆ void RTC_SetAlarmOutput(uint8_t **Output**);
- ◆ void RTC_ResetClockSec(void);
- ◆ RTC_ReqState RTC_GetResetClockSecReq(void);
- ◆ void RTC_ResetAlarm(void);
- ◆ void RTC_SetDateValue(RTC_DateTypeDef * **DateStruct**);
- ◆ void RTC_GetDateValue(RTC_DateTypeDef * **DateStruct**);
- ◆ void RTC_SetTimeValue(RTC_TimeTypeDef * **TimeStruct**);
- ◆ void RTC_GetTimeValue(RTC_TimeTypeDef * **TimeStruct**);
- ◆ void RTC_SetClockValue(RTC_DateTypeDef * **DateStruct**, RTC_TimeTypeDef * **TimeStruct**);
- ◆ void RTC_GetClockValue(RTC_DateTypeDef * **DateStruct**, RTC_TimeTypeDef * **TimeStruct**);
- ◆ void RTC_SetAlarmValue(RTC_AlarmTypeDef * **AlarmStruct**);
- ◆ void RTC_GetAlarmValue(RTC_AlarmTypeDef * **AlarmStruct**);

16.2.2 Detailed Description

Functions listed above can be divided into five parts:

- 1) Configure the common functions of RTC date are handled by RTC_SetDay(), RTC_GetDay(), RTC_SetDate(), RTC_GetDate(), RTC_SetMonth(), RTC_GetMonth(), RTC_SetYear(), RTC_GetYear(), RTC_SetLeapYear(), RTC_GetLeapYear(), RTC_SetDateValue(), RTC_GetDateValue(),
- 2) Configure the common functions of RTC time are handled by RTC_SetSec(), RTC_GetSec(), RTC_SetMin(), RTC_GetMin(), RTC_SetHour24(), RTC_SetHour12(), RTC_GetHour(), RTC_SetHourMode(), RTC_GetHourMode(), RTC_GetAMPM(), RTC_SetTimeValue(), RTC_GetTimeValue().
- 3) RTC_EnableClock(), RTC_DisableClock(), RTC_SetTimeAdjustReq(), RTC_GetTimeAdjustReq(), RTC_ResetClockSec(), RTC_GetResetClockSec(), RTC_SetClockValue() and RTC_GetClockValue() handle for RTC clock function only.
- 4) RTC_EnableAlarm(), RTC_DisableAlarm(), RTC_ResetAlarm(), RTC_SetAlarmValue() and RTC_GetAlarmValue() handle for RTC alarm function only.
- 5) RTC_SetAlarmOutput() and RTC_SetRTCINT() handle other specified functions.

16.2.3 Function Documentation

16.2.3.1 RTC_SetSec

Set second value for RTC clock.

Prototype:

```
void  
RTC_SetSec(uint8_t Sec);
```

Parameters:

Sec: New second value, max is 59.

Description:

This function will set new second value for RTC clock. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

Return:

None.

16.2.3.2 RTC_GetSec

Get second value of RTC clock.

Prototype:

```
uint8_t  
RTC_GetSec(void);
```

Parameters:

None

Description:

This function will return second value of RTC clock.

Return:

Second value in the range:
0 ~ 59

16.2.3.3 RTC_SetMin

Set minute value for RTC clock or alarm.

Prototype:

```
void  
RTC_SetMin(RTC_FuncMode NewMode,  
           uint8_t Min);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE**: select clock function,
- **RTC_ALARM_MODE**: select alarm function.

Min: New min value, max 59

Description:

This function will set new minute value for RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and write new minute value for RTC alarm when **NewMode** is **RTC_ALARM_MODE**. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

Return:

None

16.2.3.4 RTC_GetMin

Get minute value of RTC clock or alarm.

Prototype:

```
uint8_t  
RTC_GetMin(RTC_FuncMode NewMode);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE**: select clock function,
- **RTC_ALARM_MODE**: select alarm function.

Description:

This function will return minute value of RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and return minute value of RTC alarm when **NewMode** is **RTC_ALARM_MODE**.

Return:

Minute value in the range:
0 ~ 59

16.2.3.5 RTC_GetAMPM

Get AM or PM state in the 12 Hour mode.

Prototype:

```
uint8_t  
RTC_GetAMPM(RTC_FuncMode NewMode);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE:** select clock function,
- **RTC_ALARM_MODE:** select alarm function.

Description:

This function will return AM or PM mode of RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and return AM or PM mode of RTC alarm when **NewMode** is **RTC_ALARM_MODE**.

Return:

The mode of time:

RTC_AM_MODE: Time mode is AM.

RTC_PM_MODE: Time mode is PM.

16.2.3.6 RTC_SetHour24

Set hour value for RTC clock or alarm in the 24 Hour mode.

Prototype:

```
void  
RTC_SetHour24(RTC_FuncMode NewMode,  
              uint8_t Hour);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE:** select clock function,
- **RTC_ALARM_MODE:** select alarm function.

Hour: New hour value, max is 23.

Description:

This function will set new hour value for RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and set new hour value for RTC alarm when **NewMode** is **RTC_ALARM_MODE**. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

* If hour mode is changed to 24H mode from 12H mode, **RTC_SetHour24()** should be called to rewrite the HOURR register.

Return:

None

16.2.3.7 RTC_SetHour12

Set hour value and AM/PM mode for RTC clock or alarm in the 12 Hour mode.

Prototype:

```
void  
RTC_SetHour12(RTC_FuncMode NewMode,  
              uint8_t Hour,  
              uint8_t AmPm);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE**: select clock function,
- **RTC_ALARM_MODE**: select alarm function.

Hour: New hour value, max is 11.

AmPm: New time mode, which can be set as:

- **RTC_AM_MODE**: select AM mode for 12H mode,
- **RTC_PM_MODE**: select PM mode for 12H mode.

Description:

This function will set new hour value and AM/PM mode for RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and set new hour value and AM/PM mode for RTC alarm when **NewMode** is **RTC_ALARM_MODE**. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

* If hour mode is changed to 12H mode from 24H mode, **RTC_SetHour12()** should be called to rewrite the HOURR register.

Return:

None

16.2.3.8 RTC_GetHour

Get hour value of RTC clock or alarm.

Prototype:

```
uint8_t  
RTC_GetHour(RTC_FuncMode NewMode);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE:** select clock function,
- **RTC_ALARM_MODE:** select alarm function.

Description:

This function will return hour value of RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and return hour value of RTC alarm when **NewMode** is **RTC_ALARM_MODE**.

Return:

In 24H mode, hour value in the range:

0 ~ 23

In 12H mode, hour value in the range:

0 ~ 11

16.2.3.9 RTC_SetDay

Set day value for RTC clock or alarm.

Prototype:

```
void  
RTC_SetDay(RTC_FuncMode NewMode,  
            uint8_t Day);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE:** select clock function,
- **RTC_ALARM_MODE:** select alarm function.

Day: New day value, which can be set as:

- **RTC_SUN:** Sunday.
- **RTC_MON:** Monday.
- **RTC_TUE:** Tuesday.
- **RTC_WED:** Wednesday.
- **RTC_THU:** Thursday.
- **RTC_FRI:** Friday.
- **RTC_SAT:** Saturday.

Description:

This function will set new day value for RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and set new day value for RTC alarm when **NewMode** is **RTC_ALARM_MODE**. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

Return:

None

16.2.3.10 RTC_GetDay

Get day value of RTC clock or alarm.

Prototype:

uint8_t

RTC_GetDay(RTC_FuncMode **NewMode**);

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE**: select clock function,
- **RTC_ALARM_MODE**: select alarm function.

Description:

This function will return day value of RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and return day value of RTC alarm when **NewMode** is **RTC_ALARM_MODE**.

Return:

Day value in the range:

0 ~ 6

16.2.3.11 RTC_SetDate

Set date value for RTC clock or alarm.

Prototype:

void

RTC_SetDate(RTC_FuncMode **NewMode**,
uint8_t **Date**);

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE**: select clock function,
- **RTC_ALARM_MODE**: select alarm function.

Date: New date value, ranging from 1 to 31.

Description:

This function will set new date value for RTC clock when **NewMode** is **RTC_CLOCK_MODE**, and set new date value RTC alarm when **NewMode** is **RTC_ALARM_MODE**. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

Return:

None

16.2.3.12 RTC_GetDate

Get date value of RTC clock or alarm.

Prototype:

```
uint8_t  
RTC_GetDate(RTC_FuncMode NewMode);
```

Parameters:

NewMode: New mode of RTC, which can be set as:

- **RTC_CLOCK_MODE**: select clock function,
- **RTC_ALARM_MODE**: select alarm function.

Description:

This function will return date value of RTC clock when NewMode is **RTC_CLOCK_MODE**, and return date value of RTC alarm when NewMode is **RTC_ALARM_MODE**.

Return:

Date value in the range:
1 ~ 31

16.2.3.13 RTC_SetMonth

Set month value for RTC clock.

Prototype:

```
void
```

RTC_SetMonth(uint8_t **Month**);

Parameters:

Month: New month value, ranging from 1 to 12.

Description:

This function will set new month value for RTC clock. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

Return:

None

16.2.3.14 RTC_GetMonth

Get month value of RTC clock.

Prototype:

uint8_t
RTC_GetMonth(void);

Parameters:

None

Description:

This function will return month value.

Return:

Month value in the range:
1 ~ 12

16.2.3.15 RTC_SetYear

Set year value for RTC clock.

Prototype:

void
RTC_SetYear(uint8_t **Year**);

Parameters:

Year: New year value, max is 99.

Description:

This function will set new year value for RTC clock. RTC register are updated synchronizing with the timing of INTRTC, so after calling this function, it should wait for RTC 1HZ interrupt occurs.

Return:

None

16.2.3.16 RTC_GetYear

Get year value of RTC clock.

Prototype:

```
uint8_t  
RTC_GetYear(void);
```

Parameters:

None

Description:

This function will return year value.

Return:

Year value in the range:
0 ~ 99

16.2.3.17 RTC_SetHourMode

Select 24-hour clock or 12-hour clock.

Prototype:

```
void  
RTC_SetHourMode(uint8_t HourMode);
```

Parameters:

HourMode: New mode of hour, which can be set as:

- **RTC_12_HOUR_MODE** : Select 12H mode,
- **RTC_24_HOUR_MODE**.: Select 24H mode.

Description:

This function will select 24H mode when **HourMode** is **RTC_24_HOUR_MODE** and select 12H mode when **HourMode** is **RTC_12_HOUR_MODE**.

* Before call this function, **RTC_DisableClock()** function should be called firstly. (See “RTC_DisableClock” for details)

Return:

None

16.2.3.18 RTC_GetHourMode

Get hour mode.

Prototype:

```
uint8_t  
RTC_GetHourMode(void);
```

Parameters:

None

Description:

This function will return hour mode.

Return:

Hour mode:

RTC_24_HOUR_MODE: Hour mode is 24H mode.

RTC_12_HOUR_MODE: Hour mode is 12H mode.

16.2.3.19 RTC_SetLeapYear

Set leap year state.

Prototype:

```
void  
RTC_SetLeapYear(uint8_t LeapYear);
```

Parameters:

LeapYear: The state of leap year, which can be set as:

- **RTC_LEAP_YEAR_0:** Current year is a leap year.
- **RTC_LEAP_YEAR_1:** Current year is the year following a leap year.
- **RTC_LEAP_YEAR_2:** Current year is two years after a leap year.
- **RTC_LEAP_YEAR_3:** Current year is three years after a leap year.

Description:

This function will change leap year state. If **LeapYear** is **RTC_LEAP_YEAR_0**, current year is a leap year. If **LeapYear** is **RTC_LEAP_YEAR_1**, current year is the year following a leap year. If **LeapYear** is **RTC_LEAP_YEAR_2**, current year is two years after a leap year. If **LeapYear** is **RTC_LEAP_YEAR_3**, current year is three years after a leap year.

Return:

None

16.2.3.20 RTC_GetLeapYear

Get leap year state.

Prototype:

uint8_t

RTC_GetLeapYear(void);

Parameters:

None

Description:

This function will return leap year state.

Return:

The state of the leap year.

16.2.3.21 RTC_SetTimeAdjustReq

Set time adjustment + or – 30 seconds.

Prototype:

void

RTC_SetTimeAdjustReq(void);

Parameters:

None

Description:

This function will set time adjust seconds. The request is sampled when the sec counter counts up. If the time elapsed is between 0 and 29 seconds, the sec counter is cleared to "0". If the time elapsed is between 30 and 59 seconds, the min counter is carried and sec counter is cleared to "0".

Return:

None

16.2.3.22 RTC_GetTimeAdjustReq

Get time adjust request state.

Prototype:

RTC_ReqState

RTC_GetTimeAdjustReq(void);

Parameters:

None

Description:

This function will get the state of time adjust request. In order not to request repeatedly, it should be called after calling **RTC_SetTimeAdjustReq()** function.

Return:

The state of time adjustment:

RTC_NO_REQ : No adjust request.

RTC_REQ: Adjust request.

16.2.3.23 RTC_EnableClock

Enable RTC clock function.

Prototype:

void

RTC_EnableClock(void);

Parameters:

None

Description:

This function will enable clock function.

Return:

None

16.2.3.24 RTC_DisableClock

Disable RTC clock function.

Prototype:

```
void  
RTC_DisableClock(void);
```

Parameters:

None

Description:

This function will disable clock function.

Return:

None

16.2.3.25 RTC_EnableAlarm

Enable RTC alarm function.

Prototype:

```
void  
RTC_EnableAlarm(void);
```

Parameters:

None

Description:

This function will enable alarm function.

Return:

None

16.2.3.26 RTC_DisableAlarm

Disable RTC alarm function.

Prototype:

```
void  
RTC_DisableAlarm(void);
```

Parameters:

None

Description:

This function will disable alarm function.

Return:

None

16.2.3.27 RTC_SetRTCINT

Enable or disable INTRTC.

Prototype:

void

RTC_SetRTCINT(FunctionalState **NewState**);

Parameters:

NewState: New state of INT RTC.

- **ENABLE**: Enable INTRTC.
- **DISABLE**: Disable INTRTC.

Description:

This function will enable RTCINT when **NewState** is **ENABLE**, and disable RTCINT when **NewState** is **DISABLE**.

Return:

None

16.2.3.28 RTC_SetAlarmOutput

Set output signals from ALARM pin.

Prototype:

void

RTC_SetAlarmOutput(uint8_t **Output**);

Parameters:

Output: Set ALARM pin output, which can be set as:

- **RTC_LOW_LEVEL**: "0" pulse
- **RTC_PULSE_1_HZ**: 1Hz cycle "0" pulse
- **RTC_PULSE_16_HZ**: 16Hz cycle "0" pulse
- **RTC_PULSE_2_HZ**: 2Hz cycle "0" pulse

- **RTC_PULSE_4_HZ**: 4Hz cycle “0” pulse
- **RTC_PULSE_8_HZ**: 8Hz cycle “0” pulse

Description:

This function will set output signal from ALARM pin. If **Output** is **RTC_LOW_LEVEL**, Alarm pin output is “0” pulse when the alarm register corresponds with the clock. If **Output** is **RTC_PULSE_n*_HZ**, Alarm pin output is n*Hz cycle “0” pulse. (n can be one of 1,2,4,8,16)

Return:

None

16.2.3.29 RTC_ResetClockSec

Reset RTC clock second counter.

Prototype:

```
void  
RTC_ResetClockSec(void);
```

Parameters:

None

Description:

This function will reset sec counter.

Return:

None

16.2.3.30 RTC_GetResetClockSecReq

Get reset RTC clock second counter request state.

Prototype:

```
RTC_ReqState  
RTC_GetResetClockSecReq(void);
```

Parameters:

None

Description:

Get request state for reset RTC clock second counter. The request is sampled using low-speed clock. In order to wait the clock stability, it should be called after calling **RTC_ResetClockSec()** function.

Return:

The state of reset clock request:

RTC_NO_REQ: No reset clock request.

RTC_REQ: Reset clock request.

16.2.3.31 RTC_ResetAlarm

Reset RTC alarm.

Prototype:

void

RTC_ResetAlarm(void);

Parameters:

None

Description:

This function will reset alarm.

Reset alarm registers, the related parameters will be set as follows.

Minute: 00, Hour: 00, Date: 01, Day of the week: Sunday

Return:

None

16.2.3.32 RTC_SetDateValue

Set the RTC clock date.

Prototype:

void

RTC_SetDateValue(RTC_DateTypeDef * ***DateStruct***);

Parameters:

DateStruct: The structure containing basic date configuration including leap year state, year, month, date and day. (Refer to “Data structure Description” for details)

Description:

This function will set RTC clock date, including leap year, year, month, date and day. **RTC_SetLeapYear()**, **RTC_SetYear()**, **RTC_SetMonth()**, **RTC_SetDate()** and **RTC_Setday()** will be called by it.

Return:

None

16.2.3.33 RTC_GetDateValue

Get the RTC clock date.

Prototype:

void

RTC_GetDateValue(RTC_DateTypeDef * **DateStruct**);

Parameters:

DateStruct: The structure containing basic date configuration. (Refer to “Data structure Description” for details)

Description:

This function will get RTC clock date, including leap year, year, month, date and day. **RTC_GetLeapYear()**, **RTC_GetYear()**, **RTC_GetMonth()**, **RTC_GetDate()** and **RTC_Getday()** will be called by it.

Return:

None

16.2.3.34 RTC_SetTimeValue

Set the RTC clock time.

Prototype:

void

RTC_SetTimeValue(RTC_TimeTypeDef * **TimeStruct**);

Parameters:

TimeStruct: The structure containing basic time configuration including hour mode, hour, AM/PM mode in 12H mode, minute and second. (Refer to “Data structure Description” for details)

Description:

This function will set RTC clock time, including hour mode, hour, AM/PM mode in 12H mode, minute and second. **RTC_SetHourMode()**, **RTC_SetHour12()**, **RTC_SetHour24()**, **RTC_SetMin()** and **RTC_SetSec()** will be called by it.

Return:

None

16.2.3.35 RTC_GetTimeValue

Get the RTC time.

Prototype:

void

RTC_GetTimeValue(RTC_TimeTypeDef * ***TimeStruct***);

Parameters:

TimeStruct: The structure containing basic Time configuration. (Refer to "Data structure Description" for details)

Description:

This function will Get RTC clock time, including hour mode, hour, AM/PM mode in 12H mode, minute and second. **RTC_GetHourMode()**, **RTC_GetHour()**, **RTC_GetAMPM()**, **RTC_GetMin()** and **RTC_GetSec()** will be called by it.

Return:

None

16.2.3.36 RTC_SetClockValue

Set the RTC clock date and time.

Prototype:

void

RTC_SetClockValue(RTC_DateTypeDef * ***DateStruct***,
RTC_TimeTypeDef * ***TimeStruct***);

Parameters:

DateStruct: The structure containing basic Date configuration including leap year state, year, month, date and day.

TimeStruct. The structure containing basic Time configuration including hour mode, hour, AM/PM mode in 12H mode, minute and second. (Refer to “Data structure Description” for details)

Description:

This function will set RTC clock date and time, including leap year, year, month, date, day, hour mode, hour, AM/PM mode in 12H mode, minute and second.

RTC_SetLeapYear(), **RTC_SetYear()**, **RTC_SetMonth()**, **RTC_SetDate()**, **RTC_SetDay()**, **RTC_SetHourMode()**, **RTC_SetHour24()**, **RTC_SetHour12()**, **RTC_SetMin()** and **RTC_SetSec()** will be called by it.

Return:

None

16.2.3.37 RTC_GetClockValue

Get the RTC clock date and time.

Prototype:

void

RTC_GetClockValue(RTC_DateTypeDef * **DateStruct**,
RTC_TimeTypeDef * **TimeStruct**);

Parameters:

DateStruct. The structure containing basic Date configuration including leap year state, year, month, date and day.

TimeStruct. The structure containing basic Time configuration including hour mode, hour, AM/PM mode in 12H mode, minute and second. (Refer to “Data structure Description” for details)

Description:

This function will get RTC clock date and time, including leap year, year, month, date, day, hour mode, hour, AM/PM mode in 12H mode, minute and second.

RTC_GetLeapYear(), **RTC_GetYear()**, **RTC_GetMonth()**, **RTC_GetDate()**, **RTC_GetDay()**, **RTC_GetHourMode()**, **RTC_GetHour()**, **RTC_GetAMPM()**, **RTC_GetMin()** and **RTC_GetSec()** will be called by it.

Return:

None

16.2.3.38 RTC_SetAlarmValue

Set the RTC alarm date and time.

Prototype:

void

RTC_SetAlarmValue(RTC_AlarmTypeDef * **AlarmStruct**);

Parameters:

AlarmStruct: The structure containing basic alarm configuration including date, day, hour, AM/PM mode in 12H mode and minute. (Refer to “Data structure Description” for details)

Description:

This function will set RTC alarm date and time, including date, day, hour, AM/PM mode in 12H mode and minute. **RTC_SetDate()**, **RTC_SetDay()**, **RTC_SetHour12()**, **RTC_SetHour24()** and **RTC_SetMin()** will be called by it.

Return:

None

16.2.3.39 RTC_GetAlarmValue

Get the RTC alarm date and time.

Prototype:

void

RTC_GetAlarmValue(RTC_AlarmTypeDef * **AlarmStruct**);

Parameters:

AlarmStruct: The structure containing basic alarm configuration including date, day, hour, AM/PM mode in 12H mode and minute. (Refer to “Data structure Description” for details)

Description:

This function will get RTC alarm date and time, including date, day, hour, AM/PM mode in 12H mode and minute. **RTC_GetDate()**, **RTC_GetDay()**, **RTC_GetHour()**, **RTC_GetAMPM()** and **RTC_GetMin()** will be called by it.

Return:

None

16.2.4 Data Structure Description

16.2.4.1 RTC_DateTypeDef

Data Fields:

uint8_t

LeapYear set leap year state, which can be set as:

- **RTC_LEAP_YEAR_0**: Current year is a leap year.
- **RTC_LEAP_YEAR_1**: Current year is the year following a leap year.
- **RTC_LEAP_YEAR_2**: Current year is two years after a leap year.
- **RTC_LEAP_YEAR_3**: Current year is three years after a leap year

uint8_t

Year new year value, max is 99.

uint8_t

Month new month value, ranging from 1 to 12.

uint8_t

Date new date value, ranging from 1 to 31.

uint8_t

Day new day value, which can be set as:

- **RTC_SUN**: Sunday.
- **RTC_MON**: Monday.
- **RTC_TUE**: Tuesday.
- **RTC_WED**: Wednesday.
- **RTC_THU**: Thursday.
- **RTC_FRI**: Friday.
- **RTC_SAT**: Saturday.

16.2.4.2 RTC_TimeTypeDef

Data Fields:

uint8_t

HourMode select 24H mode or 12H mode, which can be set as:

- **RTC_12_HOUR_MODE**: Hour mode is 12H mode
- **RTC_24_HOUR_MODE**: Hour mode is 24H mode

uint8_t

Hour new hour value, max value is 23 in 24H mode or 11 in 12H mode.

uint8_t

AmPm select AM/PM mode for 12H mode, which can be set as:

- **RTC_AM_MODE**: select AM mode for 12H mode,
- **RTC_PM_MODE**: select PM mode for 12H mode.

- **RTC_AMPM_INVALID:** when hour mode is 24H mode.

uint8_t

Min new minute value, max is 59.

uint8_t

Sec new second value, max is 59.

16.2.4.3 RTC_AlarmTypeDef

Data Fields:

uint8_t

Date new date value of RTC alarm, ranging from 1 to 31.

uint8_t

Day new day value of RTC alarm, which can be set as:

- **RTC_SUN:** Sunday.
- **RTC_MON:** Monday.
- **RTC_TUE:** Tuesday.
- **RTC_WED:** Wednesday.
- **RTC_THU:** Thursday.
- **RTC_FRI:** Friday.
- **RTC_SAT:** Saturday.

uint8_t

Hour new hour value of RTC alarm, max value is 23 in 24H mode, max value is 11 in 12H mode.

uint8_t

AmPm select AM/PM mode for 12H mode, which can be set as:

- **RTC_AM_MODE:** select AM mode for 12H mode,
- **RTC_PM_MODE:** select PM mode for 12H mode.
- **RTC_AMPM_INVALID:** when hour mode is 24H mode.

uint8_t

Min new minute value of RTC alarm, max is 59.

17. SBI

17.1 Overview

This device contains a Serial Bus Interface channel, which can operate in I2C bus mode with multi-master capability.

In I2C bus mode, the SBI is connected to external devices via SCL and SDA.

Data can be transferred in free data format by the SBI channel. In free data format, data is always sent by master-transmitter and received by slave-receiver.

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

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_sbi.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_sbi.h containing the macros, data types, structures and API definitions for use by applications.

17.2 API Functions

17.2.1 Function List

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

- ◆ void SBI_SetI2CFreeDataMode(TSB_SBI_TypeDef* **SBIx**, FunctionalState **NewState**);

17.2.2 Detailed Description

Functions listed above can be divided into four parts:

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

17.2.3 Function Documentation

Note: in all of the following APIs, parameter “TSB_SBI_TypeDef* **SBIx**” can only be **TSB_SBI0**.

17.2.3.1 SBI_Enable

Enable the specified SBI channel.

Prototype:

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

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

None

17.2.3.2 SBI_Disable

Disable the specified SBI channel.

Prototype:

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

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

None

17.2.3.3 SBI_SetI2CACK

Enable or disable the generation of ACK clock.

Prototype:

```
void  
SBI_SetI2CACK(TSB_SBI_TypeDef* SBIx,  
               FunctionalState NewState);
```

Parameters:

SBIx is the specified SBI channel.

NewState sets the generation of ACK clock, which can be:

- **ENABLE** for generating of ACK clock
- **DISABLE** for no ACK clock

Description:

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

Return:

None

17.2.3.4 SBI_InitI2C

Initialize the specified SBI channel in I2C mode.

Prototype:

```
void  
SBI_InitI2C(TSB_SBI_TypeDef* SBIx,  
             SBI_InitI2CTypeDef* InitI2CStruct);
```

Parameters:

SBIx is the specified SBI channel.

InitI2CStruct is the structure containing SBI configuration (refer to 10.2.4 Data Structure Description for details).

Description:

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

Return:

None

17.2.3.5 SBI_SetI2CBitNum

Specify the number of bits per transfer.

Prototype:

void

```
SBI_SetI2CBitNum(TSB_SBI_TypeDef* SBIx,  
                 uint32_t I2CBitNum);
```

Parameters:

SBIx is the specified SBI channel.

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

This parameter can be one of the following values:

- **SBI_I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **SBI_I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;
- **SBI_I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;
- **SBI_I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **SBI_I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;
- **SBI_I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **SBI_I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;
- **SBI_I2C_DATA_LEN_7**, which means that the data length number of bits per transfer is 7.

Description:

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

Return:

None

17.2.3.6 SBI_SWReset

Reset the state of the specified SBI channel.

Prototype:

```
void  
SBI_SWReset(TSB_SBI_TypeDef* SBIx);
```

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

None

17.2.3.7 SBI_ClearI2CINTReq

Clear SBI interrupt request in I2C bus mode.

Prototype:

```
void  
SBI_ClearI2CINTReq(TSB_SBI_TypeDef* SBIx);
```

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

None

17.2.3.8 SBI_Generatel2CStart

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

Prototype:

void

SBI_Generatel2CStart(TSB_SBI_TypeDef* **SBIx**);

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

None

17.2.3.9 SBI_Generatel2CStop

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

Prototype:

void

SBI_Generatel2CStop(TSB_SBI_TypeDef* **SBIx**);

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

None

17.2.3.10 SBI_GetI2CState

Get the SBI channel state in I2C bus mode.

Prototype:

```
SBI_I2CState  
SBI_GetI2CState(TSB_SBI_TypeDef* SBIx);
```

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

The state value of the SBI channel in I2C bus.

17.2.3.11 SBI_SetIdleMode

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

Prototype:

```
void  
SBI_SetIdleMode(TSB_SBI_TypeDef* SBIx,  
                FunctionalState NewState);
```

Parameters:

SBIx is the specified SBI channel.

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

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

Description:

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

Return:

None

17.2.3.12 SBI_SetSendData

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

Prototype:

```
void  
SBI_SetSendData(TSB_SBI_TypeDef* SBIx,
```


uint32_t **Data**);

Parameters:

SBIx is the specified SBI channel.

Data is a byte-data to be sent. The maximum value is 0xFF.

Description:

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

Return:

None

17.2.3.13 SBI_GetReceiveData

Get data received from the specified SBI channel.

Prototype:

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

Parameters:

SBIx is the specified SBI channel.

Description:

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

Return:

Data which has been received

17.2.3.14 SBI_SetI2CFreeDataMode

Set SBI channel working in I2C free data mode.

Prototype:

void

SBI_setI2CFreeDataMode(TSB_SBI_TypeDef* **SBIx**,
FunctionalState **NewState**);

Parameters:

SBIx is the specified SBI channel.

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

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

Description:

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

Return:

None

17.2.4 Data Structure Description

17.2.4.1 SBI_InitI2CTypeDef

Data Fields:

uint32_t

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

uint32_t

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

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

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

uint32_t

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

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

FunctionalState

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

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

17.2.4.2 SBI_I2CState

Data Fields:

uint32_t

All specifies state data in I2C mode

Bit Fields:

uint32_t

LastRxBit specifies last received bit monitor.

uint32_t

GeneralCall specifies general call detected monitor.

uint32_t

SlaveAddrMatch specifies slave address match monitor.

uint32_t

ArbitrationLost specifies arbitration last detected monitor.

uint32_t

INTReq specifies Interrupt request monitor.

uint32_t

BusState specifies bus busy flag.

uint32_t

TRx specifies transfer or Receive selection monitor.

uint32_t

MasterSlave specifies master or slave selection monitor.

uint32_t

RESERVED0 Reserved

18. TMRB

18.1 Overview

TOSHIBA TPM440 contains 20 channels of a 16-bit up-counter, two 16-bit timer registers, two 16-bit capture registers, two comparators, a capture input control, a timer flip-flop and its associated control circuit. (TMRB0 through TMRB19). Each channel can operate in the following modes:

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

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

- Frequency measurement
- Pulse width measurement
- Time difference measurement

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

All driver APIs are contained in `/Libraries/TX04_Periph_Driver/src/tmpm440_tmr.c`, with `/Libraries/TX04_Periph_Driver/inc/tmpm440_tmr.h` containing the macros, data types, structures and API definitions for use by applications.

18.2 API Functions

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

18.2.2 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_SetIdleMode(), TMRB_SetSyncMode(), TMRB_SetDoubleBuf(), TMRB_SetExtStartTrg() and TMRB_SetClkInCoreHalt (), TMRB_SetExtInput(), TMRB_SetDMAReq() handle other specified functions.

18.2.3 Function Documentation

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

“TSB_TB_TypeDef* **TBx**” can be one of the following values:

TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB4,
TSB_TB5, TSB_TB6, TSB_TB7, TSB_TB8, TSB_TB9,

TSB_TB10, TSB_TB11, TSB_TB12, TSB_TB13, TSB_TB14,
TSB_TB15, TSB_TB16, TSB_TB17, TSB_TB18, TSB_TB19.

18.2.3.1 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

18.2.3.2 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

18.2.3.3 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

18.2.3.4 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

18.2.3.5 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_TB7, TSB_TB8, TSB_TB9, TSB_TB10, TSB_TB11,
TSB_TB12, TSB_TB13, TSB_TB14, TSB_TB15, TSB_TB16,
TSB_TB17, TSB_TB18, TSB_TB19.**

CaptureTiming specifies TMRB capture timing, which can be

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

Description:

If **CaptureTiming** is set as **TMRB_CAPTURE_IN_RISING**, then at the time of the rising edge of input port TBxIN, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel.

If **CaptureTiming** is set as **TMRB_CAPTURE_IN_RISING_FALLING**, then at the time of the rising edge of input port TBxIN, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TBxIN pin input.

If **CaptureTiming** is set as **TMRB_CAPTURE_OUTPUT_EDGE**, then at the time of the rising edge of port TBxOUT pin, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TBxOUT pin input.

The flip-flop output of TMRB7, TMRB8 and TMRB9 can be used as the capture trigger of other channels.

TMRB0~4: TB8OUT

TMRB5~8: TB9OUT
TMRB10~14: TB18OUT
TMRB15~18: TB19OUT

Return:
None

18.2.3.6 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

18.2.3.7 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
}
```

18.2.3.8 TMRB_GetINTMask

Get the factor of interrupt mask of the specified TMRB channel.

Prototype:

TMRB_INTMask

TMRB_GetINTMask (TSB_TB_TypeDef* **TBx**)

Parameters:

TBx is the specified TMRB channel.

Description:

This function is used to get the factor of interrupt mask of the specified TMRB channel.

Return:

TMRB Interrupt factor be masked. Each bit has the following meaning:

- **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.

18.2.3.9 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

18.2.3.10 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 leading timing 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 can not exceed **TrailingTiming**.

18.2.3.11 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 trailing timing 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.

18.2.3.12 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

18.2.3.13 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

18.2.3.14 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_TB7, TSB_TB8, TSB_TB9, TSB_TB10, TSB_TB11,
TSB_TB12, TSB_TB13, TSB_TB14, TSB_TB15, TSB_TB16,
TSB_TB17, TSB_TB18, TSB_TB19.**

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

18.2.3.15 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_TB7, TSB_TB8, TSB_TB9, TSB_TB10, TSB_TB11,
TSB_TB12, TSB_TB13, TSB_TB14, TSB_TB15, TSB_TB16,
TSB_TB17, TSB_TB18, TSB_TB19.

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

18.2.3.16 TMRB_SetIdleMode

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

Prototype:

void

TMRB_SetIdleMode(TSB_TB_TypeDef* **TBx**,
FunctionalState **NewState**)

Parameters:

TBx is the specified TMRB channel.

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

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

Description:

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

Return:

None

18.2.3.17 TMRB_SetSyncMode

Enable or disable the synchronous mode of specified TMRB channel.

Prototype:

void

TMRB_SetSyncMode(TSB_TB_TypeDef* **TBx**,
FunctionalState **NewState**)

Parameters:

TBx is the specified TMRB channel, which can be

TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB4,
TSB_TB5, TSB_TB6, TSB_TB7, TSB_TB8, TSB_TB10,
TSB_TB11, TSB_TB12, TSB_TB13, TSB_TB14, TSB_TB15,
TSB_TB16, TSB_TB17.

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

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

Description:

If the synchronous mode is enabled for TMRB0 through TMRB3, their start timing is synchronized with TMRB0. If the synchronous mode is enabled for TMRB4 through TMRB7, their start timing is synchronized with TMRB4. If the synchronous mode is enabled for TMRB10 through TMRB13, their start timing is synchronized with TMRB10. If the synchronous mode is enabled for TMRB14 through TMRB17, their start timing is synchronized with TMRB14.

Return:

None

Note:

TMRB0 through TMRB3, TMRB4 through TMRB7, TMRB10 through TMRB13, TMRB14 through TMRB17 must start counting by calling **TMRB_SetRunState()** before TMRB0, TMRB4, TMRB14, TMRB17 start counting, so that start timing can be synchronized.

18.2.3.18 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

18.2.3.19 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

18.2.3.20 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

18.2.3.21 TMRB_SetExtInput

Set the external input source

Prototype:

void

TMRB_SetExtInput (TSB_TB_TypeDef* **TBx**)

Parameters:

TBx is the specified TMRB channel.

Description:

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

Return:

None

18.2.3.22 TMRB_SetDMAReq

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

Prototype:

void

TMRB_SetExtStartTrg (TSB_TB_TypeDef* **TBx**,
FunctionalState **NewState**,
uint8_t **DMAReq**)

Parameters:

TBx is the specified TMRB channel.

NewState enable or disable the DMA request, which can be

- **ENABLE:** enable the DMA request,
- **DISABLE:** disable the DMA request.

DMAReq specifies DMA request of the external inputs, which can be

- **TMRB_DMA_REQ_CMP_MATCH:** Select DMA request : compare match.
- **TMRB_DMA_REQ_CAPTURE_1:** Select DMA request : input capture1.
- **TMRB_DMA_REQ_CAPTURE_0:** Select DMA request : input capture0.

Description:

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

Return:

None

Note:

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

18.2.4 Data Structure Description

18.2.4.1 TMRB_InitTypeDef

Data Fields:

uint32_t

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

uint32_t

ClkDiv specifies the division of the source clock for the internal interval timer, which can be set as:

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

uint32_t

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

uint32_t

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

- **TMRB_FREE_RUN**, which means that the up-counter will not stop counting even when the value in it is match with 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 leading timing value to be written into TBnRG0, max. 0xFFFF, and it can not be set larger than **TrailingTiming**.

18.2.4.2 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.

18.2.4.3 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

Reserved: 29 -

18.2.4.4 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

19. TMRC

19.1 Overview

TOSHIBA M440 has 1 channel TMRC. The timer contains a 1 channel of 32-bit time base timer (TCT), 4 channels of 32-bit input capture register(TCCAP0 to 3) and 8 channels of 32-bit compare register (TCCMP0 to 7).

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_tmrc.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_tmrc.c containing the macros, data types, structures and API definitions for use by applications.

19.2 API Functions

19.2.1 Function List

- ◆ void TMRC_Enable(TSB_TC_TypeDef * **TCx**);
- ◆ void TMRC_Disable(TSB_TC_TypeDef * **TCx**);
- ◆ void TMRC_SetRunState(TSB_TC_TypeDef * **TCx**, uint32_t **Cmd**);
- ◆ void TMRC_SetIdleMode(TSB_TC_TypeDef * **TCx**, FunctionalState **NewState**);
- ◆ void TMRC_ExecuteSWCapture(TSB_TC_TypeDef * **TCx**);
- ◆ void TMRC_SetSrcClk(TSB_TC_TypeDef * **TCx**, uint32_t **ClkDiv**);
- ◆ void TMRC_SetNoiseFilter(TSB_TC_TypeDef * **TCx**, TMRC_NoiseFilter **NosFlt**);
- ◆ uint32_t TMRC_GetSWCaptureValue(TSB_TC_TypeDef * **TCx**);
- ◆ uint32_t TMRC_GetReadCaptureValue(TSB_TC_TypeDef * **TCx**);
- ◆ void TMRC_CMPRegConfig(TSB_TC_TypeDef * **TCx**, TMRC_CMPConfigTypeDef * **CMPConfigStruct**);
- ◆ void TMRC_SetCMPRegValue(TSB_TC_TypeDef * **TCx**, TMRC_CMPReg **CMPRegNum**, uint32_t **CmpRegVal**);
- ◆ uint32_t TMRC_GetCMPRegValue(TSB_TC_TypeDef * **TCx**, TMRC_CMPReg **CMPRegNum**);
- ◆ void TMRC_CAPRegConfig(TSB_TC_TypeDef * **TCx**, TMRC_CAPConfigTypeDef * **CAPConfigStruct**);
- ◆ uint32_t TMRC_GetCntCaptureValue(TSB_TC_TypeDef * **TCx**, TMRC_CAPReg **CAPRegNum**);

19.2.2 Detailed Description

Functions listed above can be divided into four parts:

- 1) Configure and control the common functions of each TMRC channel are handled by TMRC_Enable(), TMRC_Disable(), TMRC_SetRunState(), TMRC_CAPRegConfig(), TMRC_CMPRegConfig ().
- 2) Capture function of each TMRC channel is handled by TMRC_ExecuteSWCapture().
- 3) The status indication of each TMRC channel is handled by TMRC_GetCntCaptureValue (),TMRC_GetReadCaptureValue (),and TMRC_GetSWCaptureValue (),TMRC_SetCMPRegValue(),TMRC_GetCMPRegValue().
- 4) TMRC_SetIdleMode(),TMRC_SetNoiseFilter(),TMRC_SetSrcClk(), handle other specified functions.

19.2.3 Function Documentation

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

“TSB_TC_TypeDef* **TCx**” can be one of the following values:

TSB_TC.

19.2.3.1 TMRC_Enable

Enable the specified TMRC channel.

Prototype:

void

TMRC_Enable(TSB_TC_TypeDef* **TCx**)

Parameters:

TCx is the specified TMRC channel.

Description:

This function will enable the specified TMRC channel selected by **TCx**.

Return:

None

19.2.3.2 TMRC_Disable

Disable the specified TMRC channel.

Prototype:

void

TMRC_Disable(TSB_TC_TypeDef* **TCx**)

Parameters:

TCx is the specified TMRC channel.

Description:

This function will disable the specified TMRC channel selected by **TCx**.

Return:

None

19.2.3.3 TMRC_SetRunState

Start or stop counter of the specified TC channel.

Prototype:

```
void  
TMRC_SetRunState(TSB_TC_TypeDef* TCx,  
                 uint32_t Cmd)
```

Parameters:

TCx is the specified TMRC channel.

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

- **TMRC_RUN**: starting counting
- **TMRC_STOP**: stopping counting

Description:

The up-counter of the specified TMRC channel starts counting if **Cmd** is **TMRC_RUN** and up-counter stops counting and the value in up-counter register is clear if **Cmd** is **TMRC_STOP**.

Return:

None

19.2.3.4 TMRC_SetIdleMode

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

Prototype:

```
void  
TMRC_SetIdleMode(TSB_TC_TypeDef* TCx,  
                 FunctionalState NewState)
```

Parameters:

TCx is the specified TMRC channel.

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

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

Description:

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

Return:

None

19.2.3.5 TMRC_ExecuteSWCapture

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

Prototype:

void

TMRC_ExecuteSWCapture(TSB_TC_TypeDef* **TCx**)

Parameters:

TCx is the specified TMRC channel.

Description:

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

Return:

None

19.2.3.6 TMRC_SetSrcClk

TBT source clock selection.

Prototype:

void

TMRC_SetSrcClk (TSB_TC_TypeDef* **TCx**,
uint32_t **ClkDiv**)

Parameters:

TCx is the specified TMRC channel.

ClkDiv Source clock. which can be

- **TMRC_CLK_DIV_4:** source clock for timer is prescaler clock divided by 4;

- **TMRC_CLK_DIV_8:** source clock for timer is prescaler clock divided by 8;
- **TMRC_CLK_DIV_16:** source clock for timer is prescaler clock divided by 16;
- **TMRC_CLK_DIV_32:** source clock for timer is prescaler clock divided by 32;
- **TMRC_CLK_DIV_64:** source clock for timer is prescaler clock divided by 64;
- **TMRC_CLK_DIV_128:** source clock for timer is prescaler clock divided by 128;
- **TMRC_CLK_DIV_256:** source clock for timer is prescaler clock divided by 256;
- **TMRC_CLK_DIV_512:** source clock for timer is prescaler clock divided by 512;
- **TMRC_CLK_TCTBT_IN:** TCTBTIN pin input

Description:

TBT source clock selection.

Return:

None

19.2.3.7 TMRC_SetNoiseFilter

TCTBTIN input noise filter.

Prototype:

void

TMRC_SetNoiseFilter (TSB_TC_TypeDef* **TCx**,
TMRC_NoiseFilter **NosFlt**)

Parameters:

TCx is the specified TMRC channel.

NosFlt specifies TMRC capture timing, which can be

- **TMRC_FILTER:** Have noise filter.
- **TMRC_NOFILTER:** No noise filter

Description:

TCTBTIN input noise filter selection.

Return:

None

19.2.3.8 TMRC_GetSWCaptureValue

Get TMRC capture value.

Prototype:

uint32_t

TMRC_GetSWCaptureValue (TSB_TC_TypeDef* **TCx**)

Parameters:

TCx is the specified TMRC channel.

Description:

Get TMRC capture value.

Return:

TMRC capture value

19.2.3.9 TMRC_GetReadCaptureValue

Get TMRC read capture value.

Prototype:

uint32_t

TMRC_GetReadCaptureValue (TSB_TC_TypeDef* **TCx**)

Parameters:

TCx is the specified TMRC channel.

Description:

Get TMRC read capture value.

Return:

TMRC read capture value

19.2.3.10 TMRC_CMPRegConfig

Mask the specified TMRC interrupt.

Prototype:

void

TMRC_CMPRegConfig (TSB_TC_TypeDef* **TCx**,
TMRC_CMPConfigTypeDef **CMPCConfigStruct**)

Parameters:

TCx is the specified TMRC channel.

CMPCConfigStruct The structure containing basic compare control register configuration, (refer to "Data Structure Description" for details).

Description:

Compare control register configuration.

Return:

None

19.2.3.11 TMRC_SetCMPRegValue

Set the value for comparing the counter.

Prototype:

void

TMRC_SetCMPRegValue (TSB_TC_TypeDef* **TCx**,
TMRC_CMPReg **CMPRegNum**,
uint32_t **CMPRegVal**)

Parameters:

TCx is the specified TMRC channel.

CMPRegNum.

- **TMRC_CMP_0** : Compare register 0.
- **TMRC_CMP_1** : Compare register 1.
- **TMRC_CMP_2**: Compare register 2.
- **TMRC_CMP_3**: Compare register 3.
- **TMRC_CMP_4**: Compare register 4.
- **TMRC_CMP_5**: Compare register 5.
- **TMRC_CMP_6**: Compare register 6.
- **TMRC_CMP_7**: Compare register 7.

CMPRegVal specifies the value of duty, max. is 0xFFFFFFFF

Description:

Set the value for comparing the counter.

Return:

None

19.2.3.12 TMRC_GetCMPRegValue

Get the value for comparing the counter.

Prototype:

uint32_t

TMRC_GetCMPRegValue (TSB_TC_TypeDef* **TCx**,
TMRC_CMPReg **CMPRegNum**)

Parameters:

TCx is the specified TMRC channel.

CMPRegNum.

- **TMRC_CMP_0** : Compare register 0.
- **TMRC_CMP_1** : Compare register 1.
- **TMRC_CMP_2**: Compare register 2.
- **TMRC_CMP_3**: Compare register 3.
- **TMRC_CMP_4**: Compare register 4.
- **TMRC_CMP_5**: Compare register 5.
- **TMRC_CMP_6**: Compare register 6.
- **TMRC_CMP_7**: Compare register 7.

Description:

Get the value for comparing the counter.

Return:

The value for comparing the counter

19.2.3.13 TMRC_ CAPRegConfig

Mask the specified TMRC interrupt.

Prototype:

void

TMRC_CAPRegConfig (TSB_TC_TypeDef* **TCx**,
TMRC_CAPConfigTypeDef **CAPConfigStruct**)

Parameters:

TCx is the specified TMRC channel.

CAPConfigStruct The structure containing basic capture control register configuration, (refer to “Data Structure Description” for details).

Description:

Capture control register configuration.

Return:

None

19.2.3.14 TMRC_GetCntCaptureValue

Get TMRC counter captured value.

Prototype:

uint32_t

TMRC_GetCntCaptureValue (TSB_TC_TypeDef* **TCx**,
TMRC_CAPReg **CAPRegNum**)

Parameters:

TCx is the specified TMRC channel.

CAPRegNum.

- **TMRC_CAP_0** : Capture register 0.
- **TMRC_CAP_1** : Capture register 1.
- **TMRC_CAP_2**: Capture register 2.
- **TMRC_CAP_3**: Capture register 3.

Description:

Get TMRC counter captured value.

Return:

TMRC counter captured value.

19.2.4 Data Structure Description

19.2.4.1 TMRC_CMPConfigTypeDef

Data Fields:

uint32_t

TMRC_CMPReg Compare register number, which can be set as:

- **TMRC_CMP_0** : Compare register 0.
- **TMRC_CMP_1** : Compare register 1.
- **TMRC_CMP_2**: Compare register 2.
- **TMRC_CMP_3**: Compare register 3.
- **TMRC_CMP_4**: Compare register 4.
- **TMRC_CMP_5**: Compare register 5.

- **TMRC_CMP_6**: Compare register 6.
- **TMRC_CMP_7**: Compare register 7.

uint32_t

TMRC_FFReverseTrg TCFF0 reverse trigger, which can be set as:

- **TMRC_FF_REVERSE_TRG_DISABLE**: Trigger disable.
- **TMRC_FF_REVERSE_TRG_ENABLE**: Trigger enable.

uint32_t

TMRC_FlipFlopCtrl TCFF0 control, which can be set as:

- **TMRC_FLIPFLOP_INVERT**: Reverse a value of TCFF0.
- **TMRC_FLIPFLOP_SET**: Set "1" to TCFF0.
- **TMRC_FLIPFLOP_CLEAR**: Clear TCFF0 to "0".

uint32_t

TMRC_CMPDBCtr Double buffer of compare register, which can be set as:

- **TMRC_CMP_DB_DISABLE**: Disable.
- **TMRC_CMP_DB_ENABLE**: Enable.

uint32_t

ModeSetting Compare match detection, which can be set as:

- **TMRC_CMP_MATCH_DISABLE**: Disable.
- **TMRC_CMP_MATCH_ENABLE**: Enable.

19.2.4.2 TMRC_CAPConfigTypeDef

Data Fields:

uint32_t

TMRC_CAPReg Capture register number, which can be set as:

- **TMRC_CAP_0** : Capture register 0.
- **TMRC_CAP_1** : Capture register 1.
- **TMRC_CaP_2**: Capture register 2.
- **TMRC_CAP_3**: Capture register 3.

uint32_t

TMRC_NoiseFilter Noise filter control for TCIN0 to 3 pin input, which can be set as:

- **TMRC_NOFILTER**: No Noise filter.
- **TMRC_FILTER**: Noise filter.

uint32_t

TMRC_ValidEdgeSel Valid edge selection for TCIN0 to 3 input, which can be set as:

- **TMRC_NO_CAPTURE:** No capture.
- **TMRC_IN_RISING_EDGE:** Rising edge.
- **TMRC_IN_FALLING_EDGE:** Falling Edge.
- **TMRC_IN_BOTH_EDGE:** Both edge.

20. TMRD

20.1 Overview

The functions of these TMRD timer are identical except connecting internal bus.

TMRD consists of two timer units (TMRD0 and TMRD1) and two clock setting circuits (prescaler) for supplying clocks to these timer units. The functions are as follows.

- 16-bit interval timer
- 16-bit programmable pulse generation (PPG)

16-bit interval timer has the following two modes.

- Timer mode: TMRD0 and TMRD1 operate independently
- Interlock timer mode: The mode can be started with a TMRD0 timer and a TMRD1 timer simultaneously

16-bit programmable pulse generation outputs have the following two modes.

- PPG mode: TMRD0 and TMRD1 operate independently, and output a programmed 2ch+2ch square waveforms
- Interlock PPG mode: TMRD0 and TMRD1 operate together, and output a programmed 3ch+1ch or 4ch square waveforms

TMRD consists of two timer units and a clock setting circuit.

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/TX04_Periph_Driver/src/tmpm440_tmr.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_tmr.h containing the macros, data types, structures and API definitions for use by applications.

20.2 API Functions

20.2.1 Function List

- ◆ void TMRD_Enable(TSB_TD_TypeDef * **TDx** , TMRD_UNIT_Channel **CHx**)
- ◆ void TMRD_Disable(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**)
- ◆ void TMRD_SetRunStateInHalt(TSB_TD_TypeDef * **TDx**, uint8_t **RunState**)
- ◆ void TMRD_SetRunStateInIdle(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
uint8_t **RunState**)
- ◆ void TMRD_SetMode(TSB_TD_TypeDef * **TDx**, uint8_t **Mode**)
- ◆ void TMRD_SetClkDivision(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
uint8_t **ClkDiv**)
- ◆ void TMRD_SetUpCntCtrl(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
uint8_t **UpCntCtrl**)
- ◆ void TMRD_SetPPGInitLeadingEdge(TSB_TD_TypeDef * **TDx**, uint8_t **PPGChannel**,
uint8_t **WaveEdge**)
- ◆ void TMRD_SetCMPRegWritePath(TSB_TD_TypeDef * **TDx**,
TMRD_UNIT_Channel **CHx**,
uint8_t **WritePath**)
- ◆ void TMRD_SetCMP0INTSrc(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
uint8_t **INTSrc**)
- ◆ void TMRD_SetRunState(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
uint8_t **RunState**)
- ◆ void TMRD_SetPhaseRelation(TSB_TD_TypeDef * **TDx**, uint8_t **PhaseRelation**)
- ◆ void TMRD_EnableUpdateCMPReg(TSB_TD_TypeDef * **TDx** ,
TMRD_UNIT_Channel **CHx**)
- ◆ void TMRD_SetDMAReq(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
FunctionalState **NewState**)
- ◆ void TMRD_SetInitTiming(TSB_TD_TypeDef * **TDx**, TMRD_UNIT_Channel **CHx**,
TMRD_TimingTypeDef * **TimingStruct**)
- ◆ void TMRD_ChangeTiming(TSB_TD_TypeDef * **TDx**, uint8_t **TimingType**,
uint32_t **Timing**)
- ◆ uint16_t TMRD_GetTiming(TSB_TD_TypeDef * **TDx**, uint8_t **TimingType**)
- ◆ void TMRD_SetBitModulationCycle(TSB_TD_TypeDef * **TDx**,
TMRD_UNIT_Channel **CHx**,
uint8_t **BitModCycle**)
- ◆ void TMRD_SetBitModUpdateTiming(TSB_TD_TypeDef * **TDx**,
TMRD_UNIT_Channel **CHx**,
FunctionalState **NewState**)

20.2.2 Detailed Description

Functions listed above can be divided into five 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().

20.2.3 Function Documentation

Note: in all of the following APIs, unless otherwise specified, the parameter: “TSB_TD_TypeDef * **TDx**” can be one of the following values:

TSB_TD.

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.

20.2.3.1 TMRD_Enable

Enable clock signal input to TMRD channel.

Prototype:

```
void  
TMRD_Enable(TSB_TD_TypeDef * TDx,  
             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

20.2.3.2 TMRD_Disable

Disable the clock signal input to TMRD channel.

Prototype:

```
void  
TMRD_Disable(TSB_TD_TypeDef * TDx,  
              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

20.2.3.3 TMRD_SetRunStateInHalt

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

Prototype:

```
void  
TMRD_SetRunStateInHalt(TSB_TD_TypeDef * TDx,  
                        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

20.2.3.4 TMRD_SetRunStateIdle

Set TMRD operation during the IDLE mode.

Prototype:

void

```
TMRD_SetRunStateIdle(TSB_TD_TypeDef * TDx,  
                     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

20.2.3.5 TMRD_SetMode

Set the operation mode for TMRD.

Prototype:

void

```
TMRD_SetMode(TSB_TD_TypeDef * TDx,  
             uint8_t Mode)
```

Parameters:

TDx is the specified TMRD unit.

Mode specifies TMRD operation mode, which can be

- **TMRD_MODE_BOTH_TMR**: TMRD0: Timer mode, TMRD1: Timer mode.
- **TMRD_MODE_0TMR_1PPG**: TMRD0: Timer mode, TMRD1: PPG mode.
- **TMRD_MODE_0PPG_1TMR**: TMRD0: PPG mode, TMRD1: Timer mode.
- **TMRD_MODE_BOTH_PPG**: TMRD0: PPG mode, TMRD1: PPG mode.
- **TMRD_MODE_INTERLOCK_TMR**: Timer mode in which TMRD0 and TMRD1 start simultaneously.

- **TMRD_MODE_INTERLOCK_PPG_2CH**: Interlock PPG mode in which TMRD0 and TMRD1 to operate together. (ch00 of TMRD0 and ch10 of TMRD1 are updated simultaneously)
- **TMRD_MODE_INTERLOCK_PPG_3CH**: Interlock PPG mode in which TMRD0 and TMRD1 to operate together. (ch00 of TMRD0 and all channels of TMRD1 are updated simultaneously)

Description:

This function will set the operation mode for TMRD0 and TMRD1. If operation mode is set to interlock PPG mode, the phase relationships between pulse generated by TMRD1 and TMRD0 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.

20.2.3.6 TMRD_SetClkDivision

Set the clock prescaler of TMRD.

Prototype:

void

TMRD_SetClkDivision(TSB_TD_TypeDef* **TDx**,
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 TMRD0 and TMRD1.

Return:

None

Note:

In the interlock PPG mode, setting the clock prescaler of TMRD1 becomes invalid, and the prescaler of TMRD1 will keep the same value with TMRD0. So setting the prescaler of TMRD0 by using this function is enough.

20.2.3.7 TMRD_SetUpCntCtrl

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

Prototype:

Void

```
TMRD_SetUpCntCtrl(TSB_TD_TypeDef * TDx,  
                  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.

20.2.3.8 TMRD_SetPPGInitLeadingEdge

Set the initial leading/trailing edge of PPG channels.

Prototype:

void

```
TMRD_SetPPGInitLeadingEdge(TSB_TD_TypeDef * TDx,  
                           uint8_t PPGChannel,  
                           uint8_t WaveEdge)
```

Parameters:

TDx is the specified TMRD unit.

PPGChannel is the specified PPG output channel, which can be

- **TMRD_PPG_CHANNEL_A0**: PPG output signal a0.
- **TMRD_PPG_CHANNEL_A1**: PPG output signal a1.
- **TMRD_PPG_CHANNEL_B0**: PPG output signal b0.
- **TMRD_PPG_CHANNEL_B1**: PPG output signal b1.

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

20.2.3.9 TMRD_SetCMPRegWritePath

Set a write path to update the compare register.

Prototype:

void

```
TMRD_SetCMPRegWritePath(TSB_TD_TypeDef* TDx,  
                        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 TMRD0, the update method is the same as in PPG mode, which described above. For TMRD1, 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 TMRD1 is selected as the values in TMRD0, so only setting the writing path of TMRD0 is enough.

20.2.3.10 TMRD_SetCMP0INTSrc

Set the interrupt source of compare interrupt 0.

Prototype:

void

```
TMRD_SetCMP0INTSrc(TSB_TD_TypeDef* TDx,  
                   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 TMRD0 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 TMRD1 becomes invalid as an interrupt factor.

TMRD_INT_MATCH_PHASE is invalid as an interrupt factor of TMRD1.

20.2.3.11 TMRD_SetRunState

Set the count operation of TMRD.

Prototype:

Void

```
TMRD_SetRunState(TSB_TD_TypeDef* TDx,  
                 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 TMRD1, because TMRD1 will start operation in tandem with COUNTER0 of TMRD0.

20.2.3.12 TMRD_SetPhaseRelation

Set the phase relation of phase B to phase A.

Prototype:

void

TMRD_SetPhaseRelation(TSB_TD_TypeDef* ***TDx***,
uint8_t ***PhaseRelation***)

Parameters:

TDx is the specified TMRD unit.

PhaseRelation is the phase relation of phase B to phase A, which can be:

- **TMRD_PHASE_DELAY_OR_SAME**: Phase B delays or the same as phase A.
- **TMRD_PHASE_FAST_OR_SAME**: Phase B is fast or the same as phase A.

Description:

This function will set the phase relation of phase B to phase A.

Return:

None

Note:

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

20.2.3.13 TMRD_EnableUpdateCMPReg

Enable to update the compare register.

Prototype:

void

TMRD_EnableUpdateCMPReg(TSB_TD_TypeDef* ***TDx***,
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 TMRD0, the enable flag of TMRD1 will set at the same time, please don't use this function to set TMRD1 again. And this flag will be zero cleared when a match of compare register 05(CP05) is detected.

20.2.3.14 TMRD_SetDMAReq

Enable or disable the DMA request of INTTDxCMP0 signal.

Prototype:

void

TMRD_SetDMAReq(TSB_TD_TypeDef* ***TDx***,
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 INTTDxCMP0, which is a factor to generate a DMA request.

Return:

None

20.2.3.15 TMRD_SetInitTiming

Initialize TMRD timing parameters.

Prototype:

void

```
TMRD_SetInitTiming(TSB_TD_TypeDef* TDx,  
                  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-1

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 TMRD1)

20.2.3.16 TMRD_ChangeTiming

Change the specified TMRD timing value.

Prototype:

void

```
TMRD_ChangeTiming(TSB_TD_TypeDef* TDx,  
                  uint8_t TimingType,  
                  uint32_t Timing)
```

Parameters:

TDx is the specified TMRD unit.

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

- **TMRD_TIMING_TD0_CYCLE**: TMRD0 cycle timing, CPRG00.
- **TMRD_TIMING_A0_LEADING**: Signal a0 leading timing (CPRG01).
- **TMRD_TIMING_A0_TRAILING**: Signal a0 trailing timing (CPRG02).
- **TMRD_TIMING_A1_LEADING**: Signal a1 leading timing (CPRG03).
- **TMRD_TIMING_A1_TRAILING**: Signal a1 trailing timing (CPRG04).
- **TMRD_TIMING_PHASE_SHIFT**: Phase shift timing (CPRG05).
- **TMRD_TIMING_TD1_CYCLE**: TMRD1 cycle timing (CPRG10).
- **TMRD_TIMING_B0_LEADING**: Signal b0 leading timing (CPRG11).
- **TMRD_TIMING_B0_TRAILING**: Signal b0 trailing timing (CPRG12).
- **TMRD_TIMING_B1_LEADING**: Signal b1 leading timing (CPRG13).
- **TMRD_TIMING_B1_TRAILING**: Signal b1 trailing timing (CPRG14).

Timing is the timing value. The data range is 0x02~0x10000.

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.

20.2.3.17 TMRD_GetTiming

Get the specified TMRD timing value.

Prototype:

```
uint16_t  
TMRD_GetTiming(TSB_TD_TypeDef* TDx,  
               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**: TMRD0 cycle timing, CPRG00.
- **TMRD_TIMING_A0_LEADING**: Signal a0 leading timing (CPRG01).
- **TMRD_TIMING_A0_TRAILING**: Signal a0 trailing timing (CPRG02).
- **TMRD_TIMING_A1_LEADING**: Signal a1 leading timing (CPRG03).
- **TMRD_TIMING_A1_TRAILING**: Signal a1 trailing timing (CPRG04).
- **TMRD_TIMING_PHASE_SHIFT**: Phase shift timing (CPRG05).
- **TMRD_TIMING_TD1_CYCLE**: TMRD1 cycle timing (CPRG10).
- **TMRD_TIMING_B0_LEADING**: Signal b0 leading timing (CPRG11).
- **TMRD_TIMING_B0_TRAILING**: Signal b0 trailing timing (CPRG12).
- **TMRD_TIMING_B1_LEADING**: Signal b1 leading timing (CPRG13).
- **TMRD_TIMING_B1_TRAILING**: Signal b1 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.

20.2.3.18 TMRD_SetBitModulationCycle

Set the 1bit modulation cycle.

Prototype:

```
void  
TMRD_SetBitModulationCycle (TSB_TD_TypeDef* TDx,
```


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_A0**: PPG output signal a0.
- **TMRD_PPG_CHANNEL_A1**: PPG output signal a1.
- **TMRD_PPG_CHANNEL_B0**: PPG output signal b0.
- **TMRD_PPG_CHANNEL_B1**: PPG output signal b1.

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.

20.2.3.19 TMRD_SetBitModUpdateTiming

Set the 1bit modulation update timing.

Prototype:

void

TMRD_SetBitModUpdateTiming (TSB_TD_TypeDef* **TDx**,
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_A0**: PPG output signal a0.
- **TMRD_PPG_CHANNEL_A1**: PPG output signal a1.
- **TMRD_PPG_CHANNEL_B0**: PPG output signal b0.
- **TMRD_PPG_CHANNEL_B1**: PPG output signal b1.

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 timing 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, function `TMRD_SetBitModUpdateTiming ()` is ignored.

20.2.4 Data Structure Description

20.2.4.1 TMRD_TimingTypeDef

Data Fields:

uint32_t

Cycle specifies the TMRD0/1 cycle value, it will set to CPRG00/10.

uint16_t

LeadingTiming0 specifies the signal a0/b0 leading timing (CPRG01/11).

uint16_t

TrailingTiming0 specifies the signal a0/b0 trailing timing (CPRG02/12).

uint8_t

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

uint16_t

LeadingTiming1 specifies the signal a1/b1 leading timing (CPRG03/13).

uint16_t

TrailingTiming1 specifies the signal a1/b1 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.

Timer Unit	Compare register	16-bit interval timer	
		<TDCLE> = "0"	<TDCLE> = "1"
TMRD0	TD0CP0	$0x0000 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$
	TD0CP1	$0x0000 \leq \text{CPRG1}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG1}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP2	$0x0000 \leq \text{CPRG2}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP3	$0x0000 \leq \text{CPRG3}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG3}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP4	$0x0000 \leq \text{CPRG4}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP5	$0x0000 \leq \text{CPRG5}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG5}[15:0] \leq \text{CPRG0}[15:0]$
TMRD1	TD1CP0	$0x0000 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$
	TD1CP1	$0x0000 \leq \text{CPRG1}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG1}[15:0] \leq \text{CPRG0}[15:0]$
	TD1CP2	$0x0000 \leq \text{CPRG2}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$
	TD1CP3	$0x0000 \leq \text{CPRG3}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG3}[15:0] \leq \text{CPRG0}[15:0]$
	TD1CP4	$0x0000 \leq \text{CPRG4}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$

Setting range of compare register in 16-bit interval timer mode

Timer Unit	Compare register	16-bit programmable pulse generation	
		PPG	Interlock PPG
TMRD0	TD0CP0	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$
	TD0CP1	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$
	TD0CP2	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP3	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$
	TD0CP4	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP5	don't care	$0x0000 \leq \text{CPRG5}[15:0] < (\text{CPRG0}[15:0] + 2)$
TMRD1	TD1CP0	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	don't care
	TD1CP1	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$
	TD1CP2	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{TD0CP0} < \text{CPRG0}[15:0]$
	TD1CP3	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$
	TD1CP4	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{TD0CP0} < \text{CPRG0}[15:0]$

Setting range of compare register in 16-bit programmable rectangular wave output

21. SIO/UART

21.1 Overview

This device has several serial I/O channels. Each channel can operate in I/O Interface mode(synchronous communication) and UART mode (asynchronous communication), which can be 7-bit length, 8-bit length and 9-bit length.

In 9-bit UART mode, a wakeup function can be used when the master controller can start up slave controllers via the serial link (multi-controller system).

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

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_uart.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_uart.h containing the macros, data types, structures and API definitions for use by applications.

21.2 API Functions

21.2.1 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**, SIO_InitTypeDef* **InitStruct**)
- ◆ uint8_t SIO_GetRxData(TSB_SC_TypeDef* **SIOx**)
- ◆ void SIO_SetTxData(TSB_SC_TypeDef* **SIOx**, uint8_t **Data**)

21.2.2 Detailed Description

Functions listed above can be divided into three parts:

- 1) Initialize and configure the common functions of each UART channel are handled by UART_Enable(), UART_Disable(), UART_SetInputClock(), UART_Init(), UART_DefaultConfig(), SIO_Enable(), SIO_Disable(), SIO_SetInputClock() and SIO_Init().
- 2) Transfer control and error check of each UART channel are handled by UART_GetBufState(), UART_GetRxData(), UART_SetTxData(), UART_GetErrState(), SIO_GetRxData() and 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().

21.2.3 Function Documentation

Note: in all of the following APIs, parameter “TSB_SC_TypeDef* **UARTx**” can be one of the following values:

UART0, UART1, UART2, UART3, UART4, UART5.

parameter “TSB_SC_TypeDef* **SIOx**” can be one of the following values:

SIO0, SIO1, SIO2, SIO3, SIO4, SIO5.

21.2.3.1 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

21.2.3.2 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

21.2.3.3 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.

21.2.3.4 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

21.2.3.5 UART_Init

Initialize and configure the specified UART channel.

Prototype:

```
void  
UART_Init(TSB_SC_TypeDef* UARTx,  
           UART_InitTypeDef* InitStruct)
```

Parameters:

UARTx is the specified UART channel.

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

Description:

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

Return:

None

21.2.3.6 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

21.2.3.7 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

21.2.3.8 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

21.2.3.9 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.

21.2.3.10 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

21.2.3.11 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

21.2.3.12 UART_FIFOConfig

Enable or disable FIFO.

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

21.2.3.13 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

21.2.3.14 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_RTXCNT_NONE or **UART_RTXCNT_AUTODISABLE** .

Description:

Controls automatic disabling of transmission and reception.

Return:

None

21.2.3.15 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

21.2.3.16 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

21.2.3.17 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

21.2.3.18 UART_RxFIFOFillLevel

Receive FIFO fill level to generate receive interrupts.

Prototype:

void

```
UART_RxFIFOFillLevel (TSB_SC_TypeDef* UARTx,  
                      uint32_t UARTFIFOType,  
                      uint32_t RxFIFOLevel);
```

Parameters:

UARTx is the specified UART channel.

UARTFIFOType is the UART FIFO size type.

This parameter can be one of the following values:

UART_FIFO_4B or **UART_FIFO_32B**.

RxFIFOLevel is receive FIFO fill level.

When **UARTFIFOType** is **UART_FIFO_4B** this parameter can be one of the following values:

UART_RXFIFO4B_FLEVLE_4_2B, **UART_RXFIFO4B_FLEVLE_1_1B**,
UART_RXFIFO4B_FLEVLE_2_2B or **UART_RXFIFO4B_FLEVLE_3_1B**.

When **UARTFIFOType** is **UART_FIFO_32B** this parameter can be one of the following values:

UART_RXFIFO32B_FLEVLE_32_16B, **UART_RXFIFO32B_FLEVLE_1_1B**,
UART_RXFIFO32B_FLEVLE_2_2B, **UART_RXFIFO32B_FLEVLE_3_3B**,
UART_RXFIFO32B_FLEVLE_4_4B, **UART_RXFIFO32B_FLEVLE_5_5B**,
UART_RXFIFO32B_FLEVLE_6_6B, **UART_RXFIFO32B_FLEVLE_7_7B**,
UART_RXFIFO32B_FLEVLE_8_8B, **UART_RXFIFO32B_FLEVLE_9_9B**,
UART_RXFIFO32B_FLEVLE_10_10B, **UART_RXFIFO32B_FLEVLE_11_11B**,
UART_RXFIFO32B_FLEVLE_12_12B, **UART_RXFIFO32B_FLEVLE_13_13B**,
UART_RXFIFO32B_FLEVLE_14_14B, **UART_RXFIFO32B_FLEVLE_15_15B**,
UART_RXFIFO32B_FLEVLE_16_16B, **UART_RXFIFO32B_FLEVLE_17_1B**,
UART_RXFIFO32B_FLEVLE_18_2B, **UART_RXFIFO32B_FLEVLE_19_3B**,
UART_RXFIFO32B_FLEVLE_20_4B, **UART_RXFIFO32B_FLEVLE_21_5B**,
UART_RXFIFO32B_FLEVLE_22_6B, **UART_RXFIFO32B_FLEVLE_23_7B**,
UART_RXFIFO32B_FLEVLE_24_8B, **UART_RXFIFO32B_FLEVLE_25_9B**,
UART_RXFIFO32B_FLEVLE_26_10B, **UART_RXFIFO32B_FLEVLE_27_11B**,
UART_RXFIFO32B_FLEVLE_28_12B, **UART_RXFIFO32B_FLEVLE_29_13B**,
UART_RXFIFO32B_FLEVLE_30_14B, **UART_RXFIFO32B_FLEVLE_31_15B**.

Description:

Receive FIFO fill level to generate receive interrupts.

Return:

None

21.2.3.19 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

21.2.3.20 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

21.2.3.21 UART_TxFIFOFillLevel

Transmit FIFO fill level to generate transmit interrupts.

Prototype:

void

```
UART_TxFIFOFillLevel (TSB_SC_TypeDef* UARTx,  
                      uint32_t UARTFIFOType,  
                      uint32_t TxFIFOLevel);
```

Parameters:

UARTx is the specified UART channel.

UARTFIFOType is the UART FIFO size type.

This parameter can be one of the following values:

UART_FIFO_4B or **UART_FIFO_32B**.

TxFIFOLevel is transmit FIFO fill level.

When **UARTFIFOType** is **UART_FIFO_4B** this parameter can be one of the following values:

UART_TXFIFO4B_FLEVLE_0_0B, **UART_TXFIFO4B_FLEVLE_1_1B**,
UART_TXFIFO4B_FLEVLE_2_0B or **UART_TXFIFO4B_FLEVLE_3_1B**.

When **UARTFIFOType** is **UART_FIFO_32B** this parameter can be one of the following values:

UART_TXFIFO32B_FLEVLE_0_0B, **UART_TXFIFO32B_FLEVLE_1_1B**,
UART_TXFIFO32B_FLEVLE_2_0B, **UART_TXFIFO32B_FLEVLE_3_3B**,
UART_TXFIFO32B_FLEVLE_4_4B , **UART_TXFIFO32B_FLEVLE_5_5B**,
UART_TXFIFO32B_FLEVLE_6_6B, **UART_TXFIFO32B_FLEVLE_7_7B** ,
UART_TXFIFO32B_FLEVLE_8_8B, **UART_TXFIFO32B_FLEVLE_9_9B**,
UART_TXFIFO32B_FLEVLE_10_10B,**UART_TXFIFO32B_FLEVLE_11_11B**,
UART_TXFIFO32B_FLEVLE_12_12B,**UART_TXFIFO32B_FLEVLE_13_13B**,
UART_TXFIFO32B_FLEVLE_14_14B,**UART_TXFIFO32B_FLEVLE_15_15B**,
UART_TXFIFO32B_FLEVLE_16_16B,**UART_TXFIFO32B_FLEVLE_17_1B**,
UART_TXFIFO32B_FLEVLE_18_2B, **UART_TXFIFO32B_FLEVLE_19_3B**,
UART_TXFIFO32B_FLEVLE_20_4B, **UART_TXFIFO32B_FLEVLE_21_5B**,
UART_TXFIFO32B_FLEVLE_22_6B, **UART_TXFIFO32B_FLEVLE_23_7B**,
UART_TXFIFO32B_FLEVLE_24_8B, **UART_TXFIFO32B_FLEVLE_25_9B**,
UART_TXFIFO32B_FLEVLE_26_10B,**UART_TXFIFO32B_FLEVLE_27_11B**,
UART_TXFIFO32B_FLEVLE_28_12B,**UART_TXFIFO32B_FLEVLE_29_13B**,
UART_TXFIFO32B_FLEVLE_30_14B,**UART_TXFIFO32B_FLEVLE_31_15B**.

Description:

Transmit FIFO fill level to generate transmit interrupts.

Return:

None

21.2.3.22 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

21.2.3.23 UART_TxFIFOClear

TransmitFIFO clear.

Prototype:

void

UART_TxFIFOClear (TSB_SC_TypeDef* **UARTx**);

Parameters:

UARTx is the specified UART channel.

Description:

Transmit FIFO clear.

Return:

None

21.2.3.24 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

21.2.3.25 UART_GetRxFIFOFillLevelStatus

Status of receive FIFO fill level.

Prototype:

uint32_t

UART_GetRxFIFOFillLevelStatus (TSB_SC_TypeDef* **UARTx**,
uint32_t **UARTFIFOType**);

Parameters:

UARTx is the specified UART channel.

UARTFIFOType is the UART FIFO size type.

This parameter can be one of the following values:

UART_FIFO_4B or **UART_FIFO_32B**.

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_32B: TX FIFO fill level is 32 bytes.

21.2.3.26 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.

21.2.3.27 UART_GetTxFIFOFillLevelStatus

Status of transmit FIFO fill level.

Prototype:

uint32_t

UART_GetTxFIFOFillLevelStatus (TSB_SC_TypeDef* **UARTx**,
uint32_t **UARTFIFOType**);

Parameters:

UARTx is the specified UART channel.

UARTFIFOType is the UART FIFO size type.

This parameter can be one of the following values:

UART_FIFO_4B or **UART_FIFO_32B**.

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_32B: TX FIFO fill level is 32 bytes.

21.2.3.28 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.

21.2.3.29 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 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

21.2.3.30 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

21.2.3.31 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

21.2.3.32 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

21.2.3.33 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

21.2.3.34 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

21.2.3.35 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

21.2.3.36 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

21.2.3.37 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

21.2.4 Data Structure Description

21.2.4.1 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.

21.2.4.2 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 occurred 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.

22. WDT

22.1 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\TX04_Periph_Driver\src\tmpm440_wdt.c, with \Libraries\TX04_Periph_Driver\inc\tmpm440_wdt.h containing the API definitions for use by applications.

22.2 API Functions

22.2.1 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)

22.2.2 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().

22.2.3 Function Documentation

22.2.3.1 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 $2^{15}/\text{fsys}$
- **WDT_DETECT_TIME_EXP_17:** **DetectTime** is $2^{17}/\text{fsys}$
- **WDT_DETECT_TIME_EXP_19:** **DetectTime** is $2^{19}/\text{fsys}$
- **WDT_DETECT_TIME_EXP_21:** **DetectTime** is $2^{21}/\text{fsys}$
- **WDT_DETECT_TIME_EXP_23:** **DetectTime** is $2^{23}/\text{fsys}$
- **WDT_DETECT_TIME_EXP_25:** **DetectTime** is $2^{25}/\text{fsys}$

Description:

This function will set detection time for WDT.

Return:

None

22.2.3.2 WDT_SetIdleMode

Run or stop the WDT counter when the system enters IDLE mode.

Prototype:

void

WDT_SetIdleMode(FunctionalState **NewState**)

Parameters:

NewState: Run or stop WDT counter.

This parameter can be one of the following values:

- **ENABLE:** Run the WDT counter.
- **DISABLE:** Stop the WDT counter.

Description:

This function will run the WDT counter when the system enters IDLE mode when **NewState** is **ENABLE**, and stop the WDT counter when the system enters IDLE mode when **NewState** is **DISABLE**.

Notes:

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

Return:

None

22.2.3.3 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

22.2.3.4 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

22.2.3.5 WDT_Enable

Enable the WDT function.

Prototype:

void
WDT_Enable(void)

Parameters:

None

Description:

This function will enable WDT.

Return:

None

22.2.3.6 WDT_Disable

Disable the WDT function.

Prototype:

void
WDT_Disable(void)

Parameters:

None

Description:

This function will disable WDT.

Return:

None

22.2.3.7 WDT_WriteClearCode

Write the clear code.

Prototype:

void

WDT_WriteClearCode (void)

Parameters:

None

Description:

This function will clear the WDT counter.

Return:

None

22.2.4 Data Structure Description

22.2.4.1 WDT_InitTypeDef

Data Fields:

uint32_t

DetectTime Set WDT detection time, which can be set as:

- **WDT_DETECT_TIME_EXP_15:** *DetectTime* is $2^{15}/f_{sys}$
- **WDT_DETECT_TIME_EXP_17:** *DetectTime* is $2^{17}/f_{sys}$
- **WDT_DETECT_TIME_EXP_19:** *DetectTime* is $2^{19}/f_{sys}$
- **WDT_DETECT_TIME_EXP_21:** *DetectTime* is $2^{21}/f_{sys}$
- **WDT_DETECT_TIME_EXP_23:** *DetectTime* is $2^{23}/f_{sys}$
- **WDT_DETECT_TIME_EXP_25:** *DetectTime* is $2^{25}/f_{sys}$

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.

23. PSC

23.1 Overview

TOSHIBA TMPM440 incorporates PSC (Programmable Servo/Sequence Controller) as a servo controller for motors or a sequence controller for various equipments.

TOSHIBA TMPM440 contains four units PSC (Programmable servo controller). The PSC driver APIs provide a set of functions to configure the PSC access driver function.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm440_psc.c, with /Libraries/TX04_Periph_Driver/inc/tmpm440_psc.h containing the macros, data types, structures and API definitions for use by applications.

23.2 API Functions

23.2.1 Function List

- ◆ void PSC_SetAccumulator(uint32_t **UA0**);
- ◆ uint32_t PSC_GetAccumulator(void);
- ◆ void PSC_SetMultiplReg(uint32_t **UM0**);
- ◆ uint32_t PSC_GetMultiplReg(void);
- ◆ void PSC_SetShiftCountReg(uint32_t **UM1**);
- ◆ uint32_t PSC_GetShiftCountReg(void);
- ◆ void PSC_SetUpperLimitReg(uint32_t **UL0**);
- ◆ uint32_t PSC_GetUpperLimitReg(void);
- ◆ void PSC_SetLowerLimitReg(uint32_t **UL1**);
- ◆ uint32_t PSC_GetLowerLimitReg(void);
- ◆ void PSC_SetAddSubReg0(uint32_t **UR0**);
- ◆ uint32_t PSC_GetAddSubReg0(void);
- ◆ void PSC_SetAddSubReg1(uint32_t **UR1**);
- ◆ uint32_t PSC_GetAddSubReg1(void);
- ◆ void PSC_SetArithmeticParameter(PSC_ArithmeticPARAM **Param**);
- ◆ PSC_ArithmeticPARAM PSC_GetArithmeticParameter(void);
- ◆ void PSC_SetAddrPointer0(uint32_t **AP0**);
- ◆ uint32_t PSC_GetAddrPointer0(void);
- ◆ void PSC_SetAddrPointer1(uint32_t **AP1**);
- ◆ uint32_t PSC_GetAddrPointer1(void);
- ◆ void PSC_SetAddrPointer2(uint32_t **AP2**);
- ◆ uint32_t PSC_GetAddrPointer2(void);
- ◆ void PSC_SetAddrPointer3(uint32_t **AP3**);

- ◆ uint32_t PSC_GetAddrPointer3(void);
- ◆ void PSC_SetBreakPointer(uint32_t **BR0**);
- ◆ uint32_t PSC_GetBreakPointer(void);
- ◆ void PSC_SetProgramPointer(uint32_t **PG0**);
- ◆ uint32_t PSC_GetProgramPointer(void);
- ◆ void PSC_SetRepetProcVectPointer(uint32_t **VG0**);
- ◆ uint32_t PSC_GetRepetProcVectPointer(void);
- ◆ void PSC_SetBreakFUNC(FunctionalState **NewState**);
- ◆ void PSC_SetStepFUNC(FunctionalState **NewState**);
- ◆ void PSC_SetExecutionFUNC(FunctionalState **NewState**);
- ◆ void PSC_SetArithmeticResult(PSC_ArithmeticResult **ArithmeticResult**);
- ◆ PSC_ArithmeticResult PSC_GetArithmeticResult(void);
- ◆ void PSC_SetTriggerEventFUNC(uint32_t **Event**, FunctionalState **NewState**);
- ◆ FunctionalState PSC_GetTriggerEventState(uint32_t **Event**);
- ◆ void PSC_SetTriggerEventEdge(uint32_t **Event**, PSC_TriggerEventEdge **Edge**);
- ◆ PSC_TriggerEventEdge PSC_GetTriggerEventEdge(uint32_t **Event**);
- ◆ PSC_OverRunFlag PSC_GetOverRunFlag(void);
- ◆ PSC_StartupFlag PSC_GetStartupFlag(void);
- ◆ void PSC_ClearEventOverrunFlag(uint32_t **Event**);
- ◆ void PSC_SetStartup(uint32_t **Event**);
- ◆ void PSC_SetOutputData(uint8_t **Data**);
- ◆ void PSC_SetOutputDataBit(uint8_t **Bit_x**, uint8_t **BitValue**);
- ◆ void PSC_SetDataOutputFUNC(uint8_t **Bit_x**, FunctionalState **NewState**);
- ◆ FunctionalState PSC_GetDataOutputState(uint8_t **Bit_x**);
- ◆ uint8_t PSC_GetInputData(void);
- ◆ uint8_t PSC_GetInputDataBit(uint8_t **Bit_x**);
- ◆ PSC_ExecutionState PSC_GetExecutionState(void);

23.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) Setting the Register of PSC.

PSC_SetAccumulator(), PSC_SetMultiplReg(), PSC_SetShiftCountReg(),
PSC_SetUpperLimitReg(), PSC_SetLowerLimitReg(), PSC_SetAddSubReg0(),
PSC_SetAddSubReg1(), PSC_SetArithmeticParameter(), PSC_SetAddrPointer0(),
PSC_SetAddrPointer1(), PSC_SetAddrPointer2(), PSC_SetAddrPointer3(),
PSC_SetBreakPointer(), PSC_SetProgramPointer(), PSC_SetRepetProcVectPointer(),
PSC_SetArithmeticResult()

2) Getting the Register of PSC.

PSC_GetAccumulator(), PSC_GetMultiplReg(), PSC_GetShiftCountReg(),
PSC_GetUpperLimitReg(), PSC_GetLowerLimitReg(), PSC_GetAddSubReg0(),

PSC_GetAddSubReg1(), PSC_GetAddrPointer0(), PSC_GetAddrPointer1(),
PSC_GetAddrPointer2(), PSC_GetAddrPointer3(), PSC_GetBreakPointer(),
PSC_GetProgramPointer(), PSC_GetRepetProcVectPointer(),
PSC_GetArithmeticParameter(), PSC_GetArithmeticResult()

3) Others

PSC_SetBreakFUNC(), PSC_SetStepFUNC(), PSC_SetExecutionFUNC(),
PSC_SetTriggerEventFUNC(), PSC_GetTriggerEventState(),
PSC_SetTriggerEventEdge(), PSC_GetTriggerEventEdge(), PSC_GetOverRunFlag(),
PSC_GetStartupFlag(), PSC_ClearEventOverrunFlag(), PSC_SetStartup(),
PSC_SetOutputData(), PSC_SetOutputDataBit(), PSC_SetDataOutputFUNC(),
PSC_GetDataOutputState(), PSC_GetInputData(), PSC_GetInputDataBit(),
PSC_GetExecutionState().

23.2.3 Function Documentation

23.2.3.1 PSC_SetAccumulator

Set to Accumulator

Prototype:

void

PSC_SetAccumulator (uint32_t **UA0**)

Parameters:

UA0: Value of 32bit

Description:

Set to Accumulator.

Return:

None

23.2.3.2 PSC_GetAccumulator

Get Accumulator

Prototype:

uint32_t

PSC_GetAccumulator (void)

Parameters:

None

Description:

Get Accumulator.

Return:

Value of 32bit

23.2.3.3 PSC_SetMultiplReg

Set to Multiplier register

Prototype:

void

PSC_SetMultiplReg (uint32_t *UM0*)

Parameters:

UM0: Value of 32bit

Description:

Set to Multiplier register.

Return:

None

23.2.3.4 PSC_GetMultiplReg

Get Multiplier register

Prototype:

uint32_t

PSC_GetMultiplReg (void)

Parameters:

None

Description:

Get Multiplier register.

Return:

Value of 32bit

23.2.3.5 PSC_SetShiftCountReg

Set to Shift count register

Prototype:

void

PSC_SetShiftCountReg (uint32_t *UM1*)

Parameters:

UM1: Value of 32bit

Description:

Set to Shift count register.

Return:

None

23.2.3.6 PSC_GetShiftCountReg

Get Shift count register

Prototype:

uint32_t

PSC_GetShiftCountReg (void)

Parameters:

None

Description:

Get Shift count register.

Return:

Value of 32bit

23.2.3.7 PSC_SetUpperLimitReg

Set to Upper limit value register

Prototype:

void

PSC_SetUpperLimitReg (uint32_t *UL0*)

Parameters:

UL0: Value of 32bit

Description:

Set to Upper limit value register.

Return:

None

23.2.3.8 PSC_GetUpperLimitReg

Get Upper limit value register

Prototype:

uint32_t

PSC_GetUpperLimitReg (void)

Parameters:

None

Description:

Get Upper limit value register.

Return:

Value of 32bit

23.2.3.9 PSC_SetLowerLimitReg

Set to Lower limit value register

Prototype:

void

PSC_SetLowerLimitReg (uint32_t *UL1*)

Parameters:

UL1: Value of 32bit

Description:

Set to Lower limit value register.

Return:

None

23.2.3.10 PSC_GetLowerLimitReg

Get Lower limit value register

Prototype:

uint32_t

PSC_GetLowerLimitReg (void)

Parameters:

None

Description:

Get Lower limit value register.

Return:

Value of 32bit

23.2.3.11 PSC_SetAddSubReg0

Set to Add sub value register0

Prototype:

void

PSC_SetAddSubReg0 (uint32_t *UR0*)

Parameters:

UR0: Value of 32bit

Description:

Set to Add sub value register.

Return:

None

23.2.3.12 PSC_GetAddSubReg0

Get Add sub value register0

Prototype:

uint32_t

PSC_GetAddSubReg0 (void)

Parameters:

None

Description:

Get Add sub value register.

Return:

Value of 32bit

23.2.3.13 PSC_SetAddSubReg1

Set to Add sub value register1

Prototype:

void

PSC_SetAddSubReg1 (uint32_t *UR1*)

Parameters:

UR1: Value of 32bit

Description:

Set to Add sub value register.

Return:

None

23.2.3.14 PSC_GetAddSubReg1

Get Add sub value register1

Prototype:

uint32_t

PSC_GetAddSubReg1 (void)

Parameters:

None

Description:

Get Add sub value register.

Return:

Value of 32bit

23.2.3.15 PSC_SetArithmeticParameter

Set to Arithmetic parameter signed register

Prototype:

void

PSC_SetArithmeticParameter (PSC_ArithmeticPARAM *Param*)

Parameters:

Param: The union that indicates arithmetic parameter signed register.
(Refer to “Data Structure Description” for details).

Description:

Set to Arithmetic parameter signed register.

Return:

None

23.2.3.16 PSC_GetArithmeticParameter

Get Arithmetic parameter signed register

Prototype:

PSC_ArithmeticPARAM

PSC_GetArithmeticParameter (void)

Parameters:

None

Description:

Get Arithmetic parameter signed register.

Return:

The union that indicates arithmetic parameter signed register.
(Refer to “Data Structure Description” for details).

23.2.3.17 PSC_SetAddrPointer0

Set to Address pointer0

Prototype:

void

PSC_SetAddrPointer0 (uint32_t *AP0*)

Parameters:

AP0: Value of 32bit

Description:

Set to Address pointer0.

Return:

None

23.2.3.18 PSC_GetAddrPointer0

Get Address pointer0

Prototype:

uint32_t

PSC_GetAddrPointer0 (void)

Parameters:

None

Description:

Get Address pointer0.

Return:

Value of 32bit

23.2.3.19 PSC_SetAddrPointer1

Set to Address pointer1

Prototype:

void

PSC_SetAddrPointer1 (uint32_t **AP1**)

Parameters:

AP1: Value of 32bit

Description:

Set to Address pointer1.

Return:

None

23.2.3.20 PSC_GetAddrPointer1

Get Address pointer1

Prototype:

uint32_t
PSC_GetAddrPointer1 (void)

Parameters:

None

Description:

Get Address pointer1.

Return:

Value of 32bit

23.2.3.21 PSC_SetAddrPointer2

Set to Address pointer2

Prototype:

void
PSC_SetAddrPointer2 (uint32_t **AP2**)

Parameters:

AP2: Value of 32bit

Description:

Set to Address pointer2.

Return:

None

23.2.3.22 PSC_GetAddrPointer2

Get Address pointer2

Prototype:

uint32_t
PSC_GetAddrPointer2 (void)

Parameters:

None

Description:

Get Address pointer2.

Return:

Value of 32bit

23.2.3.23 PSC_SetAddrPointer3

Set to Address pointer3

Prototype:

void

PSC_SetAddrPointer3 (uint32_t **AP3**)

Parameters:

AP3: Value of 32bit

Description:

Set to Address pointer3.

Return:

None

23.2.3.24 PSC_GetAddrPointer3

Get Address pointer3

Prototype:

uint32_t

PSC_GetAddrPointer3 (void)

Parameters:

None

Description:

Get Address pointer3.

Return:

Value of 32bit

23.2.3.25 PSC_SetBreakPointer

Set to Break pointer

Prototype:

void

PSC_SetBreakPointer (uint32_t **BR0**)

Parameters:

BR0: Value of 32bit

Description:

Set to Break pointer.

Return:

None

23.2.3.26 PSC_GetBreakPointer

Get Break pointer

Prototype:

uint32_t

PSC_GetBreakPointer (void)

Parameters:

None

Description:

Get Break pointer.

Return:

Value of 32bit

23.2.3.27 PSC_SetProgramPointer

Set to Program pointer

Prototype:

void

PSC_SetProgramPointer (uint32_t **PG0**)

Parameters:

PG0: Value of 32bit

Description:

Set to Program pointer.

Return:

None

23.2.3.28 PSC_GetProgramPointer

Get Program pointer

Prototype:

uint32_t

PSC_GetProgramPointer (void)

Parameters:

None

Description:

Get Program pointer.

Return:

Value of 32bit

23.2.3.29 PSC_SetRepetProcVectPointer

Set to Repeat process vector pointer

Prototype:

void

PSC_SetRepetProcVectPointer (uint32_t **VG0**)

Parameters:

VG0: Value of 32bit

Description:

Set to Repeat process vector pointer.

Return:

None

23.2.3.30 PSC_GetRepetProcVectPointer

Get Repeat process vector pointer

Prototype:

uint32_t

PSC_GetRepetProcVectPointer (void)

Parameters:

None

Description:

Get Repeat process vector pointer.

Return:

Value of 32bit

23.2.3.31 PSC_SetBreakFUNC

Enable or disable Break function

Prototype:

void

PSC_SetBreakFUNC (FunctionalState **NewState**)

Parameters:

NewState: Specify Break function state.

This parameter can be one of the following values:

- **DISABLE:** <BRK>=0
- **ENABLE:** <BRK>=1

Description:

Enable or disable Break function.

Return:

None

23.2.3.32 PSC_SetStepFUNC

Enable or disable Step function

Prototype:

void

PSC_SetStepFUNC (FunctionalState **NewState**)

Parameters:

NewState: Specify Step function state.

This parameter can be one of the following values:

- **DISABLE:** <STEP>=0
- **ENABLE:** <STEP>=1

Description:

Enable or disable Step function.

Return:

None

23.2.3.33 PSC_SetExecutionFUNC

Enable or disable Program execution

Prototype:

void

PSC_SetExecutionFUNC (FunctionalState **NewState**)

Parameters:

NewState: Specify Program execution state.

This parameter can be one of the following values:

- **DISABLE:** <START>=0
- **ENABLE:** <START>=1

Description:

Enable or disable Program execution.

Return:

None

23.2.3.34 PSC_SetArithmeticResult

Set to Arithmetic result register

Prototype:

void

PSC_SetArithmeticResult (PSC_ArithmeticResult **ArithmeticResult**)

Parameters:

ArithmeticResult. The union that indicates arithmetic result register.
(Refer to “Data Structure Description” for details).

Description:

Set to Arithmetic result register.

Return:

None

23.2.3.35 PSC_GetArithmeticResult

Get Arithmetic result register

Prototype:

PSC_ArithmeticResult
PSC_GetArithmeticResult (void)

Parameters:

None

Description:

Get Arithmetic result register.

Return:

The union that indicates arithmetic result register.
(Refer to “Data Structure Description” for details).

23.2.3.36 PSC_SetTriggerEventFUNC

Enable or disable the startup event trigger

Prototype:

void
PSC_SetTriggerEventFUNC (uint32_t **Event**, FunctionalState **NewState**)

Parameters:

Event. Select the number of startup trigger event.

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

- **PSC_TRIGGER_EVENT_0:** No.0 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_1:** No.1 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_2:** No.2 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_3:** No.3 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_4:** No.4 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_5:** No.5 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_6:** No.6 peripheral startup trigger is selected

- **PSC_TRIGGER_EVENT_7:** No.7 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_8:** No.8 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_9:** No.9 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_10:** No.10 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_11:** No.11 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_12:** No.12 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_13:** No.13 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_14:** No.14 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_15:** No.15 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_ALL:** All the peripheral startup triggers is selected

NewState: Specify startup event trigger state.

- **ENABLE:** Enable selected peripheral startup trigger execution.
- **DISABLE:** Disable selected peripheral startup trigger execution.

Description:

Enable or disable the startup event trigger.

Return:

None

23.2.3.37 PSC_GetTriggerEventState

Get the startup event trigger state

Prototype:

FunctionalState

PSC_GetTriggerEventState (uint32_t **Event**)

Parameters:

Event: Select the number of startup trigger event.

This parameter can be one of the following values:

- **PSC_TRIGGER_EVENT_0:** No.0 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_1:** No.1 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_2:** No.2 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_3:** No.3 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_4:** No.4 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_5:** No.5 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_6:** No.6 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_7:** No.7 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_8:** No.8 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_9:** No.9 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_10:** No.10 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_11:** No.11 peripheral startup trigger is selected

- **PSC_TRIGGER_EVENT_12:** No.12 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_13:** No.13 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_14:** No.14 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_15:** No.15 peripheral startup trigger is selected

Description:

Get the startup event trigger state.

Return:

ENABLE: Selected peripheral startup trigger execution is enabled.

DISABLE: Selected peripheral startup trigger execution is disabled.

23.2.3.38 PSC_SetTriggerEventEdge

Select the edge of startup trigger event 8 to 11

Prototype:

void

PSC_SetTriggerEventEdge (uint32_t **Event**, PSC_TriggerEventEdge **Edge**)

Parameters:

Event: Select the number of startup trigger event.

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

- **PSC_TRIGGER_EVENT_8:** No.8 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_9:** No.9 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_10:** No.10 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_11:** No.11 peripheral startup trigger is selected

Edge: Specify the edge of startup trigger event 8 to 11.

This parameter can be one of the following values:

- **PSC_START_TRIGGER_FALLING:** The startup trigger event is falling edge.
- **PSC_START_TRIGGER_RISING:** The startup trigger event is rising edge.

Description:

Select the edge of startup trigger event 8 to 11.

Return:

None

23.2.3.39 PSC_GetTriggerEventEdge

Get the edge of startup trigger event 8 to 11

Prototype:

PSC_TriggerEventEdge

PSC_GetTriggerEventEdge (uint32_t **Event**)

Parameters:

Event: Select the number of startup trigger event.

This parameter can be one of the following values:

- **PSC_TRIGGER_EVENT_8:** No.8 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_9:** No.9 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_10:** No.10 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_11:** No.11 peripheral startup trigger is selected

Description:

Get the edge of startup trigger event 8 to 11.

Return:

PSC_START_TRIGGER_FALLING: The startup trigger event is falling edge.

PSC_START_TRIGGER_RISING: The startup trigger event is rising edge.

23.2.3.40 PSC_GetOverRunFlag

Get overrun flag caused by the startup trigger event No.15 to 0.

Prototype:

PSC_OverRunFlag

PSC_GetOverRunFlag (void)

Parameters:

None

Description:

Get overrun flag caused by the startup trigger event No.15 to 0.

Return:

The union that indicates overrun flag.

(Refer to “Data Structure Description” for details).

23.2.3.41 PSC_GetStartupFlag

Get startup event occurrence flag caused by the startup trigger event No.15 to 0.

Prototype:

PSC_StartupFlag
PSC_GetStartupFlag (void)

Parameters:

None

Description:

Get startup event occurrence flag caused by the startup trigger event No.15 to 0.

Return:

The union that indicates startup event occurrence flag.
(Refer to “Data Structure Description” for details).

23.2.3.42 PSC_ClearEventOverrunFlag

Clear the startup event tag/overrun flag caused by No.15 to 0.

Prototype:

void
PSC_ClearEventOverrunFlag (uint32_t *Event*)

Parameters:

Event: Select the number of startup trigger event.

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

- **PSC_TRIGGER_EVENT_0:** No.0 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_1:** No.1 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_2:** No.2 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_3:** No.3 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_4:** No.4 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_5:** No.5 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_6:** No.6 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_7:** No.7 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_8:** No.8 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_9:** No.9 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_10:** No.10 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_11:** No.11 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_12:** No.12 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_13:** No.13 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_14:** No.14 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_15:** No.15 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_ALL:** All the peripheral startup triggers is selected

Description:

Clear the startup event tag/overrun flag caused by No.15 to 0.

Return:

None

23.2.3.43 PSC_SetStartup

Set the startup trigger events of peripheral functions by CPU

Prototype:

void

PSC_SetStartup (uint32_t *Event*)

Parameters:

Event: Select the number of startup trigger event.

This parameter can be one of the following values:

- **PSC_TRIGGER_EVENT_0**: No.0 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_1**: No.1 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_2**: No.2 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_3**: No.3 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_4**: No.4 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_5**: No.5 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_6**: No.6 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_7**: No.7 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_8**: No.8 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_9**: No.9 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_10**: No.10 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_11**: No.11 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_12**: No.12 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_13**: No.13 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_14**: No.14 peripheral startup trigger is selected
- **PSC_TRIGGER_EVENT_15**: No.15 peripheral startup trigger is selected

Description:

Set the startup trigger events of peripheral functions by CPU.

Return:

None

23.2.3.44 PSC_SetOutputData

Set output data for PSC.

Prototype:

void

PSC_SetOutputData (uint8_t **Data**)

Parameters:

Data: Specified value will be written to PSCxPTOUT register.

Description:

Set output data for PSC.

Return:

None

23.2.3.45 PSC_SetOutputDataBit

Set output data bit for PSC

Prototype:

void

PSC_SetOutputDataBit (uint8_t **Bit_x**, uint8_t **BitValue**)

Parameters:

Bit_x: Select PSC output bit.

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

- **PSC_BIT_0:** PSC port bit 0
- **PSC_BIT_1:** PSC port bit 1
- **PSC_BIT_2:** PSC port bit 2
- **PSC_BIT_3:** PSC port bit 3
- **PSC_BIT_4:** PSC port bit 4
- **PSC_BIT_5:** PSC port bit 5
- **PSC_BIT_6:** PSC port bit 6
- **PSC_BIT_7:** PSC port bit 7
- **PSC_BIT_ALL:** PSC port

BitValue: The value of specified Bit will be set.

This parameter can be one of the following values:

- **PSC_BIT_VALUE_0:** Clear PSC pin,
- **PSC_BIT_VALUE_1:** Set PSC pin.

Description:

Set output data bit for PSC.

Return:

None

23.2.3.46 PSC_SetDataOutputFUNC

Enable or disable data output for PSC

Prototype:

void

PSC_SetDataOutputFUNC (uint8_t **Bit_x**, FunctionalState **NewState**)

Parameters:

Bit_x: Select PSC output bit.

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

- **PSC_BIT_0**: PSC port bit 0
- **PSC_BIT_1**: PSC port bit 1
- **PSC_BIT_2**: PSC port bit 2
- **PSC_BIT_3**: PSC port bit 3
- **PSC_BIT_4**: PSC port bit 4
- **PSC_BIT_5**: PSC port bit 5
- **PSC_BIT_6**: PSC port bit 6
- **PSC_BIT_7**: PSC port bit 7
- **PSC_BIT_ALL**: PSC port

NewState: Specify data output state.

This parameter can be one of the following values:

- **ENABLE**: Enable output ,
- **DISABLE**: Disable output.

Description:

Enable or disable data output for PSC.

Return:

None

23.2.3.47 PSC_GetDataOutputState

Get the state of data output for PSC

Prototype:

FunctionalState

PSC_GetDataOutputState (uint8_t **Bit_x**)

Parameters:

Bit_x: Select PSC output bit.

This parameter can be one of the following values:

- **PSC_BIT_0:** PSC port bit 0
- **PSC_BIT_1:** PSC port bit 1
- **PSC_BIT_2:** PSC port bit 2
- **PSC_BIT_3:** PSC port bit 3
- **PSC_BIT_4:** PSC port bit 4
- **PSC_BIT_5:** PSC port bit 5
- **PSC_BIT_6:** PSC port bit 6
- **PSC_BIT_7:** PSC port bit 7

Description:

Get the state of data output for PSC.

Return:

ENABLE: The selected PSC port bit output is enabled,

DISABLE: The selected PSC port bit output is disabled,

23.2.3.48 PSC_GetInputData

Get input data for PSC

Prototype:

uint8_t

PSC_GetInputData (void)

Parameters:

None

Description:

Get input data for PSC.

Return:

Input data

23.2.3.49 PSC_GetInputDataBit

Get input data bit for PSC

Prototype:

uint8_t

PSC_GetInputDataBit (uint8_t **Bit_x**)

Parameters:

Bit_x: Select PSC input bit.

This parameter can be one of the following values:

- **PSC_BIT_0:** PSC port bit 0
- **PSC_BIT_1:** PSC port bit 1
- **PSC_BIT_2:** PSC port bit 2
- **PSC_BIT_3:** PSC port bit 3
- **PSC_BIT_4:** PSC port bit 4
- **PSC_BIT_5:** PSC port bit 5
- **PSC_BIT_6:** PSC port bit 6
- **PSC_BIT_7:** PSC port bit 7

Description:

Get input data bit for PSC.

Return:

PSC_BIT_VALUE_0: The selected PSC port bit input is 0,

PSC_BIT_VALUE_1: The selected PSC port bit input is 1.

23.2.3.50 PSC_GetExecutionState

Get the state of PSC program execution

Prototype:

PSC_ExecutionState

PSC_GetExecutionState (void)

Parameters:

None

Description:

Get the state of PSC program execution.

Return:

PSC_PROGRAM_STOP: Program stopping.

PSC_PROGRAM_EXECUTE: Program execution.

23.2.4 Data Structure Description

23.3.4.1 PSC_ArithmeticPARAM

Data Fields:

uint32_t

All Arithmetic parameter signed register.

Bit Fields:

uint32_t

Reserved0 (Bit 0 to Bit15)

Reserved

uint32_t

SignUA0 (Bit 16)

Signed bit of A0.

uint32_t

SignUM0 (Bit 17)

Signed bit of M0.

uint32_t

SignUM1 (Bit 18)

Signed bit of M1.

uint32_t

SignUL0 (Bit 19)

Signed bit of L0.

uint32_t

SignUL1 (Bit 20)

Signed bit of L1.

uint32_t

SignUR0 (Bit 21)

Signed bit of R0.

uint32_t

SignUR1 (Bit 22)

Signed bit of R1.

uint32_t

Reserved1 (Bit 23 to Bit 31)

Reserved

23.3.4.2 PSC_ArithmeticResult

Data Fields:

uint32_t

All Arithmetic result register.

Bit Fields:

uint32_t

Reserved0 (Bit 0 to Bit15)

Reserved

uint32_t

OverFlow (Bit 16)

OverFlow flag.

uint32_t

UnderFlow (Bit 17)

Underflow flag.

uint32_t

Reserved1 (Bit 18 to Bit 23)

Reserved

uint32_t

Zero (Bit 24)

Zero flag.

uint32_t

Reserved1 (Bit 25 to Bit 31)

Reserved

23.3.4.3 PSC_OverRunFlag

Data Fields:

uint32_t

All Overrun flag.

Bit Fields:

uint32_t

Reserved (Bit 0 to Bit15)

Reserved

uint32_t

INTOVRF0 (Bit 16)

No.0 overrun flag

uint32_t

INTOVRF1 (Bit 17)

No.1 overrun flag

uint32_t

INTOVRF2 (Bit 18)

No.2 overrun flag

uint32_t

INTOVRF3 (Bit 19)

No.3 overrun flag

uint32_t

INTOVRF4 (Bit 20)

No.4 overrun flag

uint32_t

INTOVRF5 (Bit 21)

No.5 overrun flag

uint32_t

INTOVRF6 (Bit 22)

No.6 overrun flag

uint32_t

INTOVRF7 (Bit 23)

No.7 overrun flag

uint32_t

INTOVRF8 (Bit 24)

No.8 overrun flag

uint32_t

INTOVRF9 (Bit 25)

No.9 overrun flag

uint32_t

INTOVRF10 (Bit 26)

No.10 overrun flag

uint32_t

INTOVRF11 (Bit 27)

No.11 overrun flag

uint32_t

INTOVRF12 (Bit 28)

No.12 overrun flag

uint32_t

INTOVRF13 (Bit 29)

No.13 overrun flag

uint32_t

INTOVRF14 (Bit 30)

No.14 overrun flag

uint32_t

INTOVRF15 (Bit 31)

No.15 overrun flag

23.3.4.4 PSC_StartupFlag

Data Fields:

uint32_t

All Startup event occurrence flag.

Bit Fields:

uint32_t

INTFLG0	(Bit 0)	No.0 startup flag
----------------	---------	-------------------

uint32_t

INTFLG1	(Bit 1)	No.1 startup flag
----------------	---------	-------------------

uint32_t

INTFLG2	(Bit 2)	No.2 startup flag
----------------	---------	-------------------

uint32_t

INTFLG3	(Bit 3)	No.3 startup flag
----------------	---------	-------------------

uint32_t

INTFLG4	(Bit 4)	No.4 startup flag
----------------	---------	-------------------

uint32_t

INTFLG5	(Bit 5)	No.5 startup flag
----------------	---------	-------------------

uint32_t

INTFLG6	(Bit 6)	No.6 startup flag
----------------	---------	-------------------

uint32_t

INTFLG7	(Bit 7)	No.7 startup flag
----------------	---------	-------------------

uint32_t

INTFLG8	(Bit 8)	No.8 startup flag
----------------	---------	-------------------

uint32_t

INTFLG9	(Bit 9)	No.9 startup flag
----------------	---------	-------------------

uint32_t

INTFLG10	(Bit 10)	No.10 startup flag
-----------------	----------	--------------------

uint32_t

INTFLG11	(Bit 11)	No.11 startup flag
-----------------	----------	--------------------

uint32_t

INTFLG12	(Bit 12)	No.12 startup flag
-----------------	----------	--------------------

uint32_t

INTFLG13	(Bit 13)	No.13 startup flag
-----------------	----------	--------------------

uint32_t

INTFLG14	(Bit 14)	No.14 startup flag
-----------------	----------	--------------------

uint32_t

INTFLG15	(Bit 15)	No.15 startup flag
-----------------	----------	--------------------

uint32_t

Reserved	(Bit 16 to Bit31)	Reserved
-----------------	-------------------	----------