TOSHIBA

TOSHIBA TX04 Peripheral Driver User Guide (TMPM46B)

Ver 1.000
Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M46BUG-00xE

RESTRICTIONS ON PRODUCT USE

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

Index

1. Introduction

TOSHIBA TX04 Peripheral Driver is a set of drivers for all peripherals found on the TOSHIBA TX04 series microcontrollers. TMPM46B Peripheral Driver is an important part of TOSHIBA TX04 Peripheral Driver, which is designed for TMPM46B series MCUs.

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

The design goals of TOSHIBA TMPM46B 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 TMPM46B Peripheral Drivers.

/Libraries/TX04_CMSIS
This folder contains the device peripheral access layer of TMPM46B CMSIS files.

/Libraries/TX04_Periph_Driver
This folder contains all the source code of the drivers, the core of TOSHIBA TMPM46B Peripheral Driver.

/Libraries/TX04_Periph_Driver/inc
This folder contains all the header files of TMPM46B Peripheral Drivers for each peripheral.

/Libraries/TX04_Periph_Driver/src
This folder contains all the source files of TMPM46B Peripheral Drivers for each peripheral.

/Project
This folder contains template project and examples for using TMPM46B Peripheral Driver.

/Project/Template
This folder contains template project of TOSHIBA TMPM46B Peripheral Driver.

/Project/Examples
This folder contains a set of examples for using TMPM46B Peripheral Driver

/Project/Examples/Utilities/TMPM46B-EVAL
This folder contains the configuration and driver files for hardware resources (e.g. led, key) on TMPM46B boards.

3. ADC

3.1 Overview

TMPM46B contain one unit of 12-bit sequential-conversion analog/digital converters (ADC) with normal 8 analog input (AIN0 to AIN7) channels.

The 12-bit AD converter has the following features:

1. Start normal AD conversion and top-priority AD conversion by software activation, internal triggers or an external trigger (ADTRG).

2. Operation 4 different modes of Normal AD conversion:
    Fixed-channel single conversion mode
    Channel scan single conversion mode
    Fixed-channel repeat conversion mode
    Channel scan repeat conversion mode

3. Operation modes of top-priority AD conversion:
    Fixed-channel single conversion mode

4. Normal / Top-priority AD conversion completion interrupt
5. Normal / Top-priority AD conversion completion/busy flag
6. AD monitor function
    When the AD monitor function is enabled, an interrupt is generated if any comparison result is matched.
7. AD conversion clock is controllable from fc to fc/16.
8. Current reduction function of VREF reference is supported.

The ADC API provides a set of functions for using the TMPM46B ADC modules. It includes ADC channel set, mode set, monitor function set, interrupt set, ADC status read, ADC result value read and so on.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_adc.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_adc.h containing the macros, data types, structures and API definitions for use by applications.

Note:
To use the Port J as an analog input of the AD converter, disable input on PJIE and disable pull-up on PJPUP

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***, uint32_t ***INTMode***)
- void ADC_SetInputChannel(TSB_AD_TypeDef * ***ADx***, ADC_AINx ***InputChannel***)

- void ADC_SetScanChannel(TSB_AD_TypeDef * ***ADx***,
  ADC_AINx ***StartChannel***,
  uint32_t ***Range***)
- void ADC_SetVrefCut(TSB_AD_TypeDef * ***ADx***, uint32_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***,
  ADC_AINx ***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***,
  uint32_t ***HWSrc***,
  FunctionalState ***NewState***)
- void ADC_SetHWTrgTop(TSB_AD_TypeDef * ***ADx***,
  uint32_t ***HWSrc***,
  FunctionalState ***NewState***)
- ADC_State ADC_GetConvertState(TSB_AD_TypeDef * ***ADx***)
- ADC_Result ADC_GetConvertResult(TSB_AD_TypeDef * ***ADx***,
  ADC_REGx ***ADREGx***)
- void ADC_EnableTrigger(void)
- void ADC_DisableTrigger(void)
- ADC_SetTriggerStartup(ADC_TRGx ***TriggerStartup***)
- ADC_SetTriggerStartupTop(ADC_TRGx ***TopTriggerStartup***)

3.2.2 Detailed Description

Functions listed above can be divided into five 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() and ADC_SetIdleMode() handle other specified functions.
5) ADC_EnableTrigger(), ADC_DisableTrigger(), ADC_SetTriggerStartup(), ADC_SetTriggerStartupTop().

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 the following value:

- **TSB_AD**

Description:
This function will software reset ADC.

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

Return:
None

3.2.3.2 ADC_SetClk

Set ADC sample hold time and prescaler output.

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

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

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

- **ADC_CONVERSION_CLK_10:** 10 x <ADCLK>
- **ADC_CONVERSION_CLK_20:** 20 x <ADCLK>
- **ADC_CONVERSION_CLK_30:** 30 x <ADCLK>
- **ADC_CONVERSION_CLK_40:** 40 x <ADCLK>
- **ADC_CONVERSION_CLK_80:** 80 x <ADCLK>
- **ADC_CONVERSION_CLK_160:** 160 x <ADCLK>
- **ADC_CONVERSION_CLK_320:** 320 x <ADCLK>

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

- **ADC_FC_DIVIDE_LEVEL_1:** fc
- **ADC_FC_DIVIDE_LEVEL_2:** fc / 2
- **ADC_FC_DIVIDE_LEVEL_4:** fc / 4
- **ADC_FC_DIVIDE_LEVEL_8:** fc / 8
- **ADC_FC_DIVIDE_LEVEL_16:** fc / 16

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

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

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 the following value:

- **TSB_AD**

Description:
This function will start normal AD conversion.

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

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

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

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

Return:
None

3.2.3.4 ADC_SetScanMode

Enable or disable ADC scan mode.

Prototype:
void
ADC_SetScanMode(TSB_AD_TypeDef * ***ADx***,
FunctionalState ***NewState***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

NewState: Specify ADC scan mode state
This parameter can be one of the following values:

- **ENABLE :** Enable scan mode
- **DISABLE :** Disable scan mode

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 the following value:

- **TSB_AD**

NewState: Specify ADC repeat mode state
This parameter can be one of the following values:

- **ENABLE :** Enable repeat mode
- **DISABLE :** Disable repeat mode

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***,
uint32_t ***INTMode***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

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

- **ADC_INT_SINGLE:** Generate 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.

***Note:**
This function is valid only in fixed channel repeat conversion mode.
Examples for setting fixed channel repeat conversion mode:

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

Return:
None

3.2.3.7 ADC_SetInputChannel

Set ADC input channel.

Prototype:
void
ADC_SetInputChannel(TSB_AD_TypeDef * ***ADx***,
ADC_AINx ***InputChannel***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

InputChannel: Analog input channel.
This parameter can be one of the following values:

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

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

Note:
Only one channel of **ADC_AN_00**~**ADC_AN_07** 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***,
ADC_AINx ***StartChannel***,
uint32_t ***Range***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

StartChannel: Specify the start channel to be scanned.

This parameter can be one of the following values:

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

Range: Specify the range of assignable channel scan value.
This parameter can be one of the following values:

- **1, 2, 3, 4, 5, 6, 7, 8. (note: StartChannel + Range <= 8)**

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

***Note:**
Valid channel scan setting values are shown as follows:

StartChannel	***Range*** ***(***The range of assignable channel scan value***)***
ADC_AN_00	1 to 8
ADC_AN_01	1 to 7
ADC_AN_02	1 to 6
ADC_AN_03	1 to 5
ADC_AN_04	1 to 4
ADC_AN_05	1 to 3
ADC_AN_06	1 to 2
ADC_AN_07	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***,
uint32_t ***VrefCtrl***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

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 the following value:
- **TSB_AD**

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

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_AD**

NewState: Specify AD conversion Vref application control.
This parameter can be one of the following values:
- **ENABLE :** Enable reference voltage(Vref)
- **DISABLE :** Disable reference voltage(Vref)

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

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

Return:
None

3.2.3.12 ADC_SetInputChannelTop

Select ADC top-priority conversion analog input channel.

Prototype:
void
ADC_SetInputChannelTop(TSB_AD_TypeDef * ***ADx***,
ADC_AINx ***TopInputChannel***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

TopInputChannel: Analog input channel for top-priority conversion.
This parameter can be one of the following values:

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

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

Note:
Only one channel of **ADC_AN_00**~**ADC_AN_07** 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 the following value:

- **TSB_AD**

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

***Note:**
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 the following value:

- **TSB_AD**

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 :** Enable ADC monitor
- **DISABLE :** Disable ADC monitor

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 the following value:

- **TSB_AD**

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 two AD monitor modules which are controlled by two compare control registers.
This function will specify compare control register by ***ADCMPx*** setting and specify ADC monitor configuration ***Monitor*** setting.

***Note:** 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***,
uint32_t ***HWSrc***,
FunctionalState ***NewState***)

Parameters:

ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

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

- **ADC_EXTERADTRG:** A$\overline{\text{DTRG}}$ pin
- **ADC_INTERTRIGGER:** Internal trigger (selected by ADILVTRGSEL <TRGSEL>)

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

- **ENABLE :** Enable hardware trigger source
- **DISABLE :** Disable hardware trigger source

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

***Note:**
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***,
uint32_t ***HWSrc***,
FunctionalState ***NewState***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:
- **TSB_AD**

HWSrc: Hardware source for activating top-priority AD conversion.
This parameter can be one of the following values:
- **ADC_EXTERADTRG:** A$\overline{\text{DTRG}}$ pin
- **ADC_INTERTRIGGER:** Internal trigger (selected by ADILVTRGSEL <HPTRGSEL>)

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

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

***Note:**
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 / busy flag (normal and top-priority).

Prototype:
ADC_State
ADC_GetConvertState(TSB_AD_TypeDef * ***ADx***)

Parameters:
ADx: Select ADC unit.
This parameter can be the following value:
- **TSB_AD**

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

Return:
A union with the state of AD conversion:
NormalBusy(Bit 0) : ‘1’ means normal AD is converting

NormalComplete (Bit 1) : ‘1’ means normal AD conversion is complete.
TopBusy(Bit 2) : ‘1’ means top-priority AD is converting
TopComplete (Bit 3) : ‘1’ 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***,
ADC_REGx ***ADREGx***)

Parameters:

ADx: Select ADC unit.
This parameter can be the following value:

- **TSB_AD**

ADREGx: Select ADC result register.
This 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.

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

Fixed channel single mode		
Unit	Channel	Storage register
TSB_AD	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			
Unit	Start channel	Scan channel range	Storage register
TSB_AD	ADC_AN_00	8 channels	ADC_REG_00 to ADC_REG_07
	ADC_AN_01	7 channels	ADC_REG_01 to ADC_REG_07
	ADC_AN_02	6 channels	ADC_REG_02 to ADC_REG_07
	ADC_AN_03	5 channels	ADC_REG_03 to ADC_REG_07
	ADC_AN_04	4 channels	ADC_REG_04 to ADC_REG_07
	ADC_AN_05	3 channels	ADC_REG_05 to ADC_REG_07
	ADC_AN_06	2 channels	ADC_REG_06 to ADC_REG_07
	ADC_AN_07	1 channels	ADC_REG_07 to ADC_REG_07

About the ADC mode setting, please refer to relate APIs.

***Note:**
1 For top-priority AD conversion, the result is stored in “ADC_REG_SP”.

Return: AD conversion result:

ADResult (Bit 0 to Bit 11) :	store AD result value.
Stored (Bit 12) :	‘1’ means AD result has been stored. It will be cleared if this register is read.
OverRun (Bit 13)	‘1’ means new AD result overwrote the old one. It will be cleared if this register is read.

3.2.3.20 ADC_EnableTrigger

Enable the trigger.

Prototype:
void
ADC_EnableTrigger(void)

Parameters:
None

Description:
This function will enable the trigger

Return:
None

3.2.3.21 ADC_DisableTrigger

Disable the trigger.

Prototype:
void

ADC_DisableTrigger(void)

Parameters:
None

Description:
This function will disable the trigger

Return:
None

3.2.3.22 ADC_SetTriggerStartup

Selects a trigger for startup of normal AD conversion

Prototype:
void
ADC_SetTriggerStartup(ADC_TRGx ***TriggerStartup***)

Parameters:
TriggerStartup: trigger for startup of normal AD conversion
This parameter can be one of the following values:

- **ADC_TRG_00**, **ADC_TRG_01**, **ADC_TRG_02**, **ADC_TRG_03, ADC_TRG_04**, **ADC_TRG_05**, **ADC_TRG_06**, **ADC_TRG_07, ADC_TRG_08**, **ADC_TRG_09**

Description:
This function will select a trigger for startup of normal AD conversion

Return:
None

3.2.3.23 ADC_SetTriggerStartupTop

Selects a trigger for startup of top-priority AD conversion

Prototype:
void
ADC_SetTriggerStartupTop(ADC_TRGx ***TopTriggerStartup***)

Parameters:
TopTriggerStartup: trigger for startup of top-priority AD conversion
This parameter can be one of the following values:

- **ADC_TRG_00**, **ADC_TRG_01**, **ADC_TRG_02**, **ADC_TRG_03, ADC_TRG_04**, **ADC_TRG_05**, **ADC_TRG_06**, **ADC_TRG_07, ADC_TRG_08**, **ADC_TRG_09**

Description:
This function will select a trigger for startup of top-priority AD conversion

Return:
None

3.2.4 Data Structure Description

3.2.4.1 ADC_MonitorTypeDef

Data Fields:

ADC_AINx
CmpChannel : Select which ADC channel will be used.
It can be:
ADC_AN_00 to ADC_AN_07 (8 channels)

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

ADC_CmpCondition
Condition Condition to compare ADC channel with Compare Register ,
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 ADC channel with Compare Register,
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**

(Note: please refer to part “AD monitor function” in datasheet for more detail usage information)

3.2.4.2 ADC_State

Data Fields for this union:

uint32_t
All specifies AD conversion state.

Bit Fields:
uint32_t
NormalBusy(Bit 0) Normal A/D conversion busy flag (ADBF).
‘1’ means conversion is busy

uint32_t
NormalComplete (Bit 1) Normal AD conversion complete flag (EOCF).
‘1’ means conversion is completed

uint32_t
TopBusy(Bit 2) Top-priority A/D conversion busy flag (HPADBF).
‘1’ means conversion is busy

uint32_t

TopComplete (Bit 3) Top-priority AD conversion complete flag (HPEOCF).
1' means conversion is completed

uint32_t
Reserved (Bit 4 to Bit 31) reserved.

3.2.4.3 ADC_Result

Data Fields for this union:

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) '1' means AD result has been stored.

uint32_t
OverRun (Bit 13) '1' means new AD result overwrote the old one.

uint32_t
Reserved (Bit 14 to Bit 31) reserved.

4. AES

4.1 Overview

TOSHIBA TMPM46B contains an AES processor (AES: Advanced Encryption Standard). The AES processor encrypts/decrypts data in units of 128-bit block.

The AES processor has the following features:

- Supports 3 algorithms.
  ECB mode, CBC mode, and CTR mode
- Supports 3 key lengths
  128-bit length, 192-bit length, and 256-bit length
- Supports 2 transfer modes
  CPU transfer and DMA transfer
- Provides 4-word FIFOs
  Provides two 4-word FIFOs for input data and output data.

The AES drivers API provide a set of functions to configure AES, including such parameters as plaintext/encrypted text data, arithmetic result data, input key data, output key data, algorithm setting, key length setting, DMA transfer, operation setting, FIFO status, arithmetic status and so on.

This driver is contained in \Libraries\TX04_Periph_Driver\src\tmpm46b_aes.c, with \Libraries/TX04_Periph_Driver\inc\tmpm46b_aes.h containing the API definitions for use by applications.

4.2 API Functions

4.2.1 Function List

- ◆ Result AES_SetData(uint32_t ***Data***);
- ◆ uint32_t AES_GetResult(void);
- ◆ Result AES_SetKey(AES_KeyLength ***KeyLength***, uint32_t ***Key[]***);
- ◆ uint32_t AES_GetKey(uint32_t ***KeyNum***);
- ◆ Result AES_SetCntInit(uint32_t ***CNT[4]***);
- ◆ Result AES_SetVectorInit(uint32_t ***IV[4]***);
- ◆ Result AES_ClrFIFO(void);
- ◆ void AES_Init(AES_InitTypeDef * ***InitStruct***);
- ◆ Result AES_SetOperationMode(AES_OperationMode ***OperationMode***);
- ◆ AES_OperationMode AES_GetOperationMode(void);
- ◆ Result AES_SetDMAState(FunctionalState ***DMATransfer***);
- ◆ FunctionalState AES_GetDMAState(void);
- ◆ Result AES_SetKeyLength(AES_KeyLength ***KeyLength***);
- ◆ AES_KeyLength AES_GetKeyLength(void);
- ◆ Result AES_SetAlgorithmMode(AES_AlgorithmMode ***AlgorithmMode***);
- ◆ AES_AlgorithmMode AES_GetAlgorithmMode(void);
- ◆ AES_ArithmeticStatus AES_GetArithmeticStatus(void);
- ◆ AES_FIFOStatus AES_GetWFIFOStatus(void);
- ◆ AES_FIFOStatus AES_GetRFIFOStatus(void);
- ◆ void AES_IPReset(void);

4.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) The AES basic configuration is handled by the AES_SetData(),AES_SetKey(), AES_SetCntInit(), AES_SetVectorInit(), AES_Init(), AES_SetOperationMode(), AES_SetDMAState(),AES_SetKeyLength(), and AES_SetAlgorithmMode() functions.

2) The AES operation result and status are got by the AES_GetResult (),AES_GetKey(), AES_GetOperationMode(),AES_GetDMAState(),AES_GetKeyLength(), AES_GetAlgorithmMode(),AES_GetArithmeticStatus(),AES_GetWFIFOStatus(), and AES_GetRFIFOStatus() functions.
3) The AES FIFO clear and peripheral function reset is handled by the AES_ClrFIFO () and AES_IPReset() functions.

4.2.3 Function Documentation

4.2.3.1 AES_SetData

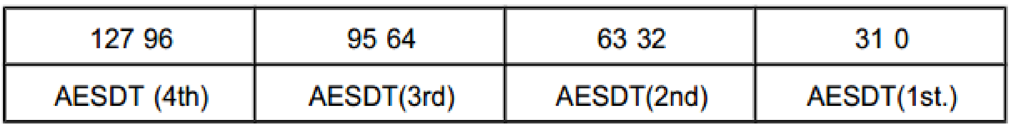
Set plaintext/encrypted data.

Prototype:
Result
AES_SetData(uint32_t ***Data***)

Parameters:
Data: Plaintext/encrypted data

Description:
This function will set plaintext/encrypted data..

***Note:**
Plaintext/Encrypted data register has a 4-word FIFO. The FIFO is required to write data four times per calculation.
Write data is allocated from the lower side shown as shown below:



127 96	95 64	63 32	31 0
AESDT (4th)	AESDT(3rd)	AESDT(2nd)	AESDT(1st.)

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.2 AES_GetResult

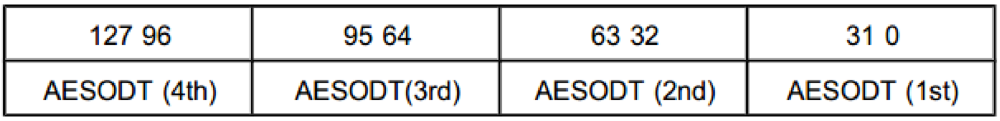
Get the calculation result.

Prototype:
uint32_t
AES_GetResult(void)

Parameters:
None

Description:
This function will get the calculation result.

***Note:**
Calculation result store register has 4-word FIFO. The FIFO is required to write data four times per calculation.
The arithmetic result can be read from the lower side and it is stored as shown below:



127 96	95 64	63 32	31 0
AESODT (4th)	AESODT(3rd)	AESODT (2nd)	AESODT (1st)

Return:
The calculation result.

4.2.3.3 AES_SetKey

Set key data.

Prototype:
Result
AES_SetKey(AES_KeyLength ***KeyLength***, uint32_t ***Key[]***)

Parameters:
KeyLength: The key length.
This parameter can be one of the following values:

- **AES_KEY_LENGTH_128**: 128-bit key will be set.
- **AES_KEY_LENGTH_192**: 192-bit key will be set.
- **AES_KEY_LENGTH_256**: 256-bit key will be set

Key[]: The key data that varies depending on the key length.

Description:
This function will set key data.

***Note:**
The input key data registers to be used vary depending on the key length specified with AESMOD<KEYLEN[1:0]>.

bit	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0
128-bit key length					AESKEY4	AESKEY5	AESKEY6	AESKEY7
192-bit key length			AESKEY2	AESKEY3	AESKEY4	AESKEY5	AESKEY6	AESKEY7
256-bit key length	AESKEY0	AESKEY1	AESKEY2	AESKEY3	AESKEY4	AESKEY5	AESKEY6	AESKEY7

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.4 AES_GetKey

Get the output key data.

Prototype:
uint32_t
AES_GetKey(uint32_t ***KeyNum***)

Parameters:
KeyNum: Specify the key to be got
This parameter can be one of the following values:

- **AES_KEY_NUM_0**: Output key store register 0 will be got.
- **AES_KEY_NUM_1**: Output key store register 1 will be got.
- **AES_KEY_NUM_2**: Output key store register 2 will be got.
- **AES_KEY_NUM_3**: Output key store register 3 will be got.
- **AES_KEY_NUM_4**: Output key store register 4 will be got.
- **AES_KEY_NUM_5**: Output key store register 5 will be got.
- **AES_KEY_NUM_6**: Output key store register 6 will be got.

- **AES_KEY_NUM_7**: Output key store register 7 will be got.

Description:
This function will get the output key data.

***Note:**
The output key store registers to be used vary depending on the key length specified with AESMOD<KEYLEN[1:0]>.

Bit	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0
128-bit key length					AESRKEY4	AESRKEY5	AESRKEY6	AESRKEY7
192-bit key length			AESRKEY2	AESRKEY3	AESRKEY4	AESRKEY5	AESRKEY6	AESRKEY7
256-bit key length	AESRKEY0	AESRKEY1	AESRKEY2	AESRKEY3	AESRKEY4	AESRKEY5	AESRKEY6	AESRKEY7

Return:
The output key data.

4.2.3.5 AES_SetCntInit

Set the counter initial value in CTR mode.

Prototype:
Result
AES_SetCntInit(uint32_t ***CNT[4U]***)

Parameters:
CNT[4U]: The counter initial value.

Description:
This function will set the counter initial value in CTR mode.

***Note:**
Data allocation is as shown below

Bit	127 96	95 64	63 32	31 0
Register	AESCNT0	AESCNT1	AESCNT2	AESCNT3

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.6 AES_SetVectorInit

Set the initial vector in CBC mode.

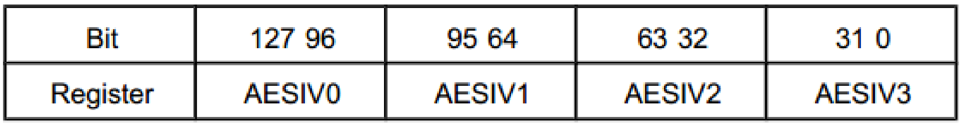
Prototype:
Result
AES_SetVectorInit(uint32_t ***IV[4U]***)

Parameters:
IV[4U]: The initial vector.

Description:

This function will set the initial vector in CBC mode.

***Note:**
Data allocation is as shown below:



Bit	127 96	95 64	63 32	31 0
Register	AESIV0	AESIV1	AESIV2	AESIV3

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.7 AES_ClrFIFO

Clear both the write FIFO and the read FIFO.

Prototype:
Result
AES_ClrFIFO(void)

Parameters:
None

Description:
This function will clear both the write FIFO and the read FIFO.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.8 AES_Init

Initialize the AES.

Prototype:
void
AES_Init(AES_InitTypeDef * ***InitStruct***)

Parameters:
InitStruct: The structure containing basic AES configuration. (Refer to Data structure Description for details)

Description:
This function will initialize the AES.

Return:
None

4.2.3.9 AES_SetOperationMode

Set the operation mode.

Prototype:
Result
AES_SetOperationMode(AES_OperationMode ***OperationMode***)

Parameters:
***OperationMode*:** Specify the operation mode.
This parameter can be one of the following values:

- **AES_ENCRYPTION_MODE:** AES encrypts plaintext to encrypted text.
- **AES_DECRYPTION_MODE:** AES decrypts encrypted text to plaintext.

Description:
This function will set the operation mode.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.10 AES_GetOperationMode

Get the operation mode.

Prototype:
AES_OperationMode
AES_GetOperationMode(void)

Parameters:
None

Description:
This function will get the operation mode.

Return:
The operation mode:
AES_ENCRYPTION_MODE: AES encrypts plaintext to encrypted text.
AES_DECRYPTION_MODE: AES decrypts encrypted text to plaintext.

4.2.3.11 AES_SetDMAState

Enable or disable the DMA transfer.

Prototype:
Result
AES_SetDMAState(FunctionalState ***DMATransfer***)

Parameters:
DMATransfer: Specify the DMA transfer.
This parameter can be one of the following values:

- **ENABLE**: Enable DMA transfer.
- **DISABLE**: Disable DMA transfer.

Description:
This function will enable or disable the DMA transfer.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.12 AES_GetDMAState

Get the DMA transfer state.

Prototype:
FunctionalState
AES_GetDMAState(void)

Parameters:
None

Description:
This function will get the DMA transfer state.

Return:
The DMA transfer state:
ENABLE: DMA transfer is being enabled.
DISABLE: DMA transfer is being disabled.

4.2.3.13 AES_SetKeyLength

Set the key length.

Prototype:
Result
AES_SetKeyLength(AES_KeyLength ***KeyLength***)

Parameters:
KeyLength: Specify the key length.
This parameter can be one of the following values:

- **AES_KEY_LENGTH_128**: Key length will be set to 128-bit.
- **AES_KEY_LENGTH_192**: Key length will be set to 192-bit.
- **AES_KEY_LENGTH_256**: Key length will be set to 256-bit.

Description:
This function will set the key length.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.14 AES_GetKeyLength

Get the key length.

Prototype:
AES_KeyLength
AES_GetKeyLength(void)

Parameters:
None

Description:
This function will get the key length.

Return:
The key length:
AES_KEY_LENGTH_128: Key length is 128-bit.
AES_KEY_LENGTH_192: Key length is 192-bit.
AES_KEY_LENGTH_256: Key length is 256-bit.

AES_KEY_UNKNOWN_LENGTH: Key length is unknown and maybe error exists.

4.2.3.15 AES_SetAlgorithmMode

Set the algorithm mode.

Prototype:
Result
AES_SetAlgorithmMode(AES_AlgorithmMode ***AlgorithmMode***)

Parameters:
AlgorithmMode: Specify the algorithm mode.
This parameter can be one of the following values:

- **AES_ECB_MODE**: Algorithm will be set to ECB mode.
- **AES_CBC_MODE**: Algorithm will be set to CBC mode.
- **AES_CTR_MODE**: Algorithm will be set to CTR mode.

Description:
This function will set the algorithm mode.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

4.2.3.16 AES_GetAlgorithmMode

Get the algorithm mode.

Prototype:
AES_AlgorithmMode
AES_GetAlgorithmMode(void)

Parameters:
None

Description:
This function will get the algorithm mode.

Return:
The algorithm mode:
AES_ECB_MODE: Algorithm mode is ECB mode.
AES_CBC_MODE: Algorithm mode is CBC mode.
AES_CTR_MODE: Algorithm mode is CTR mode.
AES_UNKNOWN_MODE: Algorithm mode is unknown and maybe error exists.

4.2.3.17 AES_GetArithmeticStatus

Get the arithmetic status.

Prototype:
AES_ArithmeticStatus
AES_GetArithmeticStatus(void)

Parameters:
None

Description:
This function will get the arithmetic status.

***Note:**
Do not write any value to AES registers when calculation is in process.

Return:
The arithmetic status:
AES_CALCULATION_COMPLETE: Calculation is complete.
AES_CALCULATION_PROCESS: Calculation is in process.

4.2.3.18 AES_GetWFIFOStatus

Get the writing FIFO status.

Prototype:
AES_FIFOStatus
AES_GetWFIFOStatus(void)

Parameters:
None

Description:
This function will get the writing FIFO status.

Return:
The writing FIFO status:
AES_FIFO_NO_DATA: No data in writing FIFO.
AES_FIFO_EXIST_DATA: Data exists in writing FIFO.

4.2.3.19 AES_GetRFIFOStatus

Get the reading FIFO status.

Prototype:
AES_FIFOStatus
AES_GetRFIFOStatus(void)

Parameters:
None

Description:
This function will get the reading FIFO status.

Return:
The reading FIFO status:
AES_FIFO_NO_DATA: No data in reading FIFO.
AES_FIFO_EXIST_DATA: Data exists in reading FIFO.

4.2.3.20 AES_IPReset

Reset AES by peripheral function.

Prototype:
void
AES_IPReset(void)

Parameters:
None

Description:
This function will reset AES by peripheral function.

Return:
None

4.2.4 Data Structure Description

4.2.4.1 AES_InitTypeDef

Data Fields:
AES_OperationMode
OperationMode Set AES operation mode, which can be set as:

- **AES_ENCRYPTION_MODE:** AES encrypts plaintext to encrypted text
- **AES_DECRYPTION_MODE:** AES decrypts encrypted text to plaintext

AES_KeyLength
KeyLength Specify the key length, which can be set as:

- **AES_KEY_LENGTH_128**: Key length will be set to 128-bit.
- **AES_KEY_LENGTH_192**: Key length will be set to 192-bit.
- **AES_KEY_LENGTH_256**: Key length will be set to 256-bit.

AES_AlgorithmMode
AlgorithmMode Specify the algorithm mode, which can be set as:

- **AES_ECB_MODE**: Algorithm will be set to ECB mode.
- **AES_CBC_MODE**: Algorithm will be set to CBC mode.
- **AES_CTR_MODE**: Algorithm will be set to CTR mode.

5. CG

5.1 Overview

The CG API provides a set of functions for using the TMPM46B 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\tmpm46b_cg.c, with TX04_Periph_Driver\inc\tmpm46b_cg.h containing the API definitions for use by applications.

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

EHCLKIN : Clock input from the X1 pins
EHOSC : Output clock from the external high-speed oscillator
ELOSC : Output clock from the external Low-speed oscillator
IHOSC : Output clock from the internal high-speed oscillator.(for SYS)
FOSCHI : Clock specified by CGOSCCR<HOSCON>
fosc : Clock specified by CGOSCCR<OSCSEL>
fpll : Clock multiplied by PLL.
fc : Clock specified by CGPLLSEL<PLLSEL> (high-speed clock).
fgear : Clock specified by CGSYSCR<GEAR[2:0]>.
fsys : Clock specified by CGSYSCR<GEAR[2:0]>.(system clock)
fperiph : Clock specified by CGSYSCR<FPSEL[2:0]>.
ΦT0 : Clock specified by CGSYSCR<PRCK[2:0]> (prescaler clock).

5.2 API Functions

5.2.1 Function List

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

- CG_FoscSrc CG_GetFoscSrc(void)
- FunctionalState CG_GetFoscState(CG_FoscSrc ***Source***)
- void CG_SetSTBYMode(CG_STBYMode ***Mode***)
- CG_STBYMode CG_GetSTBYMode(void)
- void CG_SetPortKeepInStop2Mode(FunctionalState ***NewState***)
- FunctionalState CG_GetPortKeepInStop2Mode(void)
- Result CG_SetFcSrc(CG_FcSrc ***Source***)
- CG_FcSrc CG_GetFcSrc(void)
- void CG_SetProtectCtrl(FunctionalState ***NewState***)
- void CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,
  CG_INTActiveState ***ActiveState***,
  FunctionalState ***NewState***)
- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***)
- void CG_ClearINTReq(CG_INTSrc ***INTSource***)
- CG_NMIFactor CG_GetNMIFlag(void)
- FunctionalState CG_GetIOSCFlashFlag(void)
- CG_ResetFlag CG_GetResetFlag(void)
- void CG_SetADCClkSupply(FunctionalState NewState)
- void CG_SetFcPeriphA(uint32_t ***Periph***, FunctionalState ***NewState***)
- void CG_SetFcPeriphB(uint32_t ***Periph***, FunctionalState ***NewState***)
- void CG_SetFs(FunctionalState ***NewState***)

5.2.2 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_GetFPLLValue(),CG_SetPLL(), CG_GetPLLState(), CG_SetFosc(),CG_SetFoscSrc(), CG_GetFoscSrc(), CG_GetFoscState(),CG_SetFcSrc(),CG_GetFcSrc(),CG_SetProtectCtrl().
2) The 2nd group of APIs handle settings of standby modes:
   CG_SetSTBYMode(), CG_GetSTBYMode( ), CG_SetPortKeepInStop2Mode(), CG_GetPortKeepInStop2Mode().
3) The 3rd group of APIs handle settings of interrupts:
   CG_SetSTBYReleaseINTSrc(), CG_GetSTBYReleaseINTState(), CG_ClearINTReq(), CG_GetNMIFlag(), CG_GetResetFlag().
4) The other APIs control clock supply for peripherals:
   CG_SetADCClkSupply(), CG_SetFcPeriphA(), CG_SetFcPeriphB(), CG_GetIOSCFlashFlag(), CG_SetFs().

5.2.3 Function Documentation

5.2.3.1 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

5.2.3.2 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

5.2.3.3 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

5.2.3.4 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.

5.2.3.5 CG_SetPhiT0Level

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

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

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

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

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

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

5.2.3.6 CG_GetPhiT0Level

Get the dividing level between clock ΦT0 and fc.

Prototype:

CG_DivideLevel
CG_GetPhiT0Level(void)

Parameters:
None

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

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

5.2.3.7 CG_SetSCOUTSrc

Set the clock source of SCOUT output.

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

Parameters:
Source: select clock source of SCOUT.
This parameter can be one of the following values:
- **CG_SCOUT_SRC_FS**: SCOUT source is set to fs.
- **CG_SCOUT_SRC_FSYS_DIVIDE_8**: SCOUT source is set to fsys/8.
- **CG_SCOUT_SRC_FSYS_DIVIDE_4**: SCOUT source is set to fsys/4.
- **CG_SCOUT_SRC_FOSC**: SCOUT source is set to fosc.

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

Return:
None

5.2.3.8 CG_GetSCOUTSrc

Get the clock source of SCOUT output.

Prototype:
SCOUTSrc
CG_GetSCOUTSrc(void)

Parameters:
None

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

Return:
The clock source of SCOUT output:

- **CG_SCOUT_SRC_FS**: SCOUT source is fs.
- **CG_SCOUT_SRC_FSYS_DIVIDE_8**: SCOUT source is set to fsys/8.
- **CG_SCOUT_SRC_FSYS_DIVIDE_4**: SCOUT source is set to fsys/4.
- **CG_SCOUT_SRC_FSYS**: SCOUT source is set to fsys.

5.2.3.9 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_HIGH**: internal high-speed oscillator is selected as timer source.
- **CG_WARM_UP_SRC_OSC_EXT_HIGH**: external high-speed oscillator is selected as timer source.
- **CG_WARM_UP_SRC_OSC_EXT_LOW**: external low-speed oscillator is selected as timer source.

Time:
Number of warm-up cycle. It is between 0x0000 and 0xFFFFU.

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

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

Example of calculating register value for warm-up time:

> /* When using high-speed oscillator 10MHz, and set warm-up time 5ms. */
> So value = (warm-up time to set) / (input frequency cycle(s)) = 5ms / (1/10MHz) = 5000cycle = 0xC350.
> Round lower 4 bit off, set 0xC35 to CGOSCCR<WUPT[11:0]>

Return:
None.

5.2.3.10 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

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

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

5.2.3.12 CG_SetFPLLValue

Set PLL multiplying value

Prototype:
Result
CG_SetFPLLValue(uint32_t ***NewValue***)

Parameters:
NewValue:

- **CG_8M_MUL_4_FPLL:**
  Input clock 8MHz, output clock 32MHz (4 multiplying)
- **CG_8M_MUL_5_FPLL:**
  Input clock 8MHz, output clock 40MHz (5 multiplying)
- **CG_8M_MUL_6_FPLL:**
  Input clock 8MHz, output clock 48MHz (6 multiplying)
- **CG_8M_MUL_8_FPLL:**

Input clock 8MHz, output clock 64MHz (8 multiplying)

- **CG_8M_MUL_10_FPLL:**
  Input clock 8MHz, output clock 80MHz (10 multiplying)
- **CG_8M_MUL_12_FPLL:**
  Input clock 8MHz, output clock 96MHz (12 multiplying)
- **CG_10M_MUL_4_FPLL:**
  Input clock 10MHz, output clock 40MHz (4 multiplying)
- **CG_10M_MUL_5_FPLL:**
  Input clock 10MHz, output clock 50MHz (5 multiplying)
- **CG_10M_MUL_6_FPLL:**
  Input clock 10MHz, output clock 60MHz (6 multiplying)
- **CG_10M_MUL_8_FPLL:**
  Input clock 10MHz, output clock 80MHz (8 multiplying)
- **CG_10M_MUL_10_FPLL:**
  Input clock 10MHz, output clock 100MHz (10 multiplying)
- **CG_10M_MUL_12_FPLL:**
  Input clock 10MHz, output clock 120MHz (12 multiplying)
- **CG_12M_MUL_4_FPLL:**
  Input clock 12MHz, output clock 48MHz (4 multiplying)
- **CG_12M_MUL_5_FPLL:**
  Input clock 12MHz, output clock 60MHz (5 multiplying)
- **CG_12M_MUL_6_FPLL:**
  Input clock 16MHz, output clock 72MHz (6 multiplying)
- **CG_12M_MUL_8_FPLL:**
  Input clock 12MHz, output clock 96MHz (8 multiplying)
- **CG_12M_MUL_10_FPLL:**
  Input clock 12MHz, output clock 120MHz (10 multiplying)
- **CG_16M_MUL_4_FPLL:**
  Input clock 16MHz, output clock 64MHz (4 multiplying)
- **CG_16M_MUL_5_FPLL:**
  Input clock 16MHz, output clock 90MHz (5 multiplying)

Description:
This function sets PLL multiplying value.

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

5.2.3.13 CG_GetFPLLValue

Get the value of PLL setting.

Prototype:
uint32_t
CG_GetFPLLValue(void)

Parameters:
None

Description:
This function will get the PLL multiplying value.
If the other value is read from the register, it means the value is reserved.

Return:

The source of PLL multiplying value

- **CG_8M_MUL_4_FPLL:**
  Input clock 8MHz, output clock 32MHz (4 multiplying)
- **CG_8M_MUL_5_FPLL:**
  Input clock 8MHz, output clock 40MHz (5 multiplying)
- **CG_8M_MUL_6_FPLL:**
  Input clock 8MHz, output clock 48MHz (6 multiplying)
- **CG_8M_MUL_6_FPLL:**
  Input clock 8MHz, output clock 48MHz (6 multiplying)
- **CG_8M_MUL_8_FPLL:**
  Input clock 8MHz, output clock 64MHz (8 multiplying)
- **CG_8M_MUL_10_FPLL:**
  Input clock 8MHz, output clock 80MHz (10 multiplying)
- **CG_8M_MUL_12_FPLL:**
  Input clock 8MHz, output clock 96MHz (12 multiplying)
- **CG_10M_MUL_4_FPLL:**
  Input clock 10MHz, output clock 40MHz (4 multiplying)
- **CG_10M_MUL_5_FPLL:**
  Input clock 10MHz, output clock 50MHz (5 multiplying)
- **CG_10M_MUL_6_FPLL:**
  Input clock 10MHz, output clock 60MHz (6 multiplying)
- **CG_10M_MUL_8_FPLL:**
  Input clock 10MHz, output clock 80MHz (8 multiplying)
- **CG_10M_MUL_10_FPLL:**
  Input clock 10MHz, output clock 100MHz (10 multiplying)
- **CG_10M_MUL_12_FPLL:**
  Input clock 10MHz, output clock 120MHz (12 multiplying)
- **CG_12M_MUL_4_FPLL:**
  Input clock 12MHz, output clock 48MHz (4 multiplying)
- **CG_12M_MUL_5_FPLL:**
  Input clock 12MHz, output clock 60MHz (5 multiplying)
- **CG_12M_MUL_6_FPLL:**
  Input clock 16MHz, output clock 72MHz (6 multiplying)
- **CG_12M_MUL_8_FPLL:**
  Input clock 12MHz, output clock 96MHz (8 multiplying)
- **CG_12M_MUL_10_FPLL:**
  Input clock 12MHz, output clock 120MHz (10 multiplying)
- **CG_16M_MUL_4_FPLL:**
  Input clock 16MHz, output clock 64MHz (4 multiplying)
- **CG_16M_MUL_5_FPLL:**
  Input clock 16MHz, output clock 90MHz (5 multiplying)

5.2.3.14 CG_SetPLL

Enable or disable the PLL circuit.

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

Parameters:
NewState:

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

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

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

5.2.3.15 CG_GetPLLState

Get the state of PLL circuit.

Prototype:
FunctionalState
CG_GetPLLState(void)

Parameters:
None

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

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

5.2.3.16 CG_SetFosc

Enable or disable high-speed oscillator (fosc).

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

Parameters:
Source: select clock source of fosc.
This parameter can be one of the following values:

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

NewState

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

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

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

5.2.3.17 CG_SetFoscSrc

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

Prototype:
void
CG_SetFoscSrc(CG_FoscSrc Source)

Parameters:
Source: select source for fosc.
This parameter can be one of the following values:

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

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

Return:
None

5.2.3.18 CG_GetFoscSrc

Get the source of the high-speed oscillator.

Prototype:
CG_FoscSrc
CG_GetFoscSrc(void)

Parameters:
None

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

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

5.2.3.19 CG_GetFoscState

Get the state of the high-speed oscillator.

Prototype:
FunctionalState
CG_GetFoscState(CG_FoscSrc Source)

Parameters:
Source: select source for fosc.

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

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

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

5.2.3.20 CG_SetSTBYMode

Set the standby mode.

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

Parameters:
Mode: the low power consumption mode, the description of each value is as the following:

- **CG_STBY_MODE_STOP1**: STOP1 mode. All the internal circuits including the internal oscillator are brought to a stop.
- **CG_STBY_MODE_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

5.2.3.21 CG_GetSTBYMode

Get the standby mode.

Prototype:
CG_STBYMode
CG_GetSTBYMode(void)

Parameters:
None

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

Return:
The low power mode:
CG_STBY_MODE_STOP1: STOP1 mode.
CG_STBY_MODE_STOP2: STOP2 mode
CG_STBY_MODE_IDLE: IDLE mode
CG_STBY_MODE_UNKNOWN: Invalid data is read.

5.2.3.22 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

5.2.3.23 CG_GetPortKeepInStop2Mode

Get the pin status in stop2 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 keeps in stop2 mode
DISABLE: <PTKEEP>=0
ENABLE: <PTKEEP>=1

5.2.3.24 CG_SetFcSrc

Set the clock source of fc

Prototype:
Result
CG_SetFcSrc(CG_FcSrc ***Source***)

Parameters:
Source: the source for fc
This parameter can be one of the following values:

- **CG_FC_SRC_FOSC** : fc source will be set to fosc
- **CG_FC_SRC_FPLL**: fc source will be set to fpll

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

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

5.2.3.25 CG_GetFcSrc

Get the clock source of fc.

Prototype:
CG_FcSrc
CG_GetFosc(void)

Parameters:
None

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

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

5.2.3.26 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

5.2.3.27 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_1** : INT1
- **CG_INT_SRC_2** : INT2
- **CG_INT_SRC_7** : INT7
- **CG_INT_SRC_8** : INT8
- **CG_INT_SRC_D** : INTD
- **CG_INT_SRC_E:** INTE
- **CG_INT_SRC_F:** INTF
- **CG_INT_SRC_RTC** : INTRTC

ActiveState: select the active state for release trigger.
For ***CG_INT_SRC_RTC***, this parameter can only be

- **CG_INT_ACTIVE_STATE_FALLING**: active on falling edge

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

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

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

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

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

Return:
None

5.2.3.28 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_1, CG_INT_SRC_2, CG_INT_SRC_7, CG_INT_SRC_8, CG_INT_SRC_D, CG_INT_SRC_E, CG_INT_SRC_F, CG_INT_SRC_RTC.

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

Return:
Active state of the input INT

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

5.2.3.29 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_1, CG_INT_SRC_2, CG_INT_SRC_7, CG_INT_SRC_8, CG_INT_SRC_D, CG_INT_SRC_E, CG_INT_SRC_F, CG_INT_SRC_RTC.

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

Return:
None

5.2.3.30 CG_GetNMIFlag

Get the NMI flag that shows who triggered NMI

Prototype:
CG_NMI_Factor
CG_GetNMIFlag (void)

Parameters:
None

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

Return:
NMI value:
WDT (Bit 0) means generated from WDT.
DetectLowVoltage (Bit 2) only lower than the setting voltage when voltage decreasing.

5.2.3.31 CG_GetIOSCFlashFlag

Get the flag for stopping of the internal high-speed oscillator or writing to the flash memory.

Prototype:
FunctionalState

CG_GetIOSCFlashFlag(void)

Parameters:
None

Description:
This function gets the flag for stopping of the internal high-speed oscillator or writing to the flash memory.

***Note:**
For programing into the Flash memory after entering into Normal mode from Stop2 mode, it is required to confirm that CGRSTFLG<OSCFLF> is read as "1".

Return:
Flag for stopping of the internal high-speed oscillator or writing to the flash memory:

ENABLE: Can stop the internal high-speed oscillator and write to the flash memory.

DISABLE: Can’t stop the internal high-speed oscillator and write to the flash memory.

5.2.3.32 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:
PinReset (Bit0) Reset by power on reset
WDTReset (Bit 2) means reset from WDT.
STOP2Reset(Bit3) means reset flag by STOP2 mode release
DebugReset (Bit 4) means reset from SYSRESETREQ.
LVDReset (Bit6) Rest by LVD

5.2.3.33 CG_SetADCClkSupply

Enable or disable supplying clock fsys for ADC.

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

Parameters:

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

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

Description:

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

Return:

None

5.2.3.34 CG_SetFcPeriphA

Enable or disable supplying clock fsys to peripheries.
Prototype:
void
CG_SetFcPeriphA(uint32_t ***Periph***,
FunctionalState ***NewState***)

Parameters:

Periph: The target peripheral that CG supplies clock fc for
This parameter can be one of the following values or their combination:

- **CG_FC_PERIPH_PORTA:** Clock control for PORT A
- **CG_FC_PERIPH_PORTB:** Clock control for PORT B
- **CG_FC_PERIPH_PORTC:** Clock control for PORT C
- **CG_FC_PERIPH_PORTD:** Clock control for PORT D
- **CG_FC_PERIPH_PORTE:** Clock control for PORT E
- **CG_FC_PERIPH_PORTF:** Clock control for PORT F
- **CG_FC_PERIPH_PORTG:** Clock control for PORT G
- **CG_FC_PERIPH_PORTH:** Clock control for PORT H
- **CG_FC_PERIPH_PORTJ:** Clock control for PORT J
- **CG_FC_PERIPH_PORTK:** Clock control for PORT K
- **CG_FC_PERIPH_PORTL:** Clock control for PORT L
- **CG_FC_PERIPH_TMRB0:** Clock control for TMRB0
- **CG_FC_PERIPH_TMRB1:** Clock control for TMRB1
- **CG_FC_PERIPH_TMRB2:** Clock control for TMRB2
- **CG_FC_PERIPH_TMRB3:** Clock control for TMRB3
- **CG_FC_PERIPH_TMRB4:** Clock control for TMRB4
- **CG_FC_PERIPH_TMRB5:** Clock control for TMRB5
- **CG_FC_PERIPH_TMRB6:** Clock control for TMRB6
- **CG_FC_PERIPH_TMRB7:** Clock control for TMRB7
- **CG_FC_PERIPH_MPT0:** Clock control for MPT0
- **CG_FC_PERIPH_MPT1:** Clock control for MPT1
- **CG_FC_PERIPH_MPT2:** Clock control for MPT2
- **CG_FC_PERIPH_MPT3:** Clock control for MPT3
- **CG_FC_PERIPH_TRACE:** Clock control for TRACE
- **CG_FC_PERIPHA_ALL:** ALL clock control

NewState

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

Description:

This function enables or disables supplying clock fsys to peripheries

Return:
None

5.2.3.35 CG_SetFcPeriphB

Enable or disable supplying clock fsys to peripheries.
Prototype:
void
CG_SetFcPeriphB(uint32_t ***Periph***,
FunctionalState ***NewState***)

Parameters:
Periph: The target peripheral that CG supplies clock fc for
This parameter can be one of the following values or their combination:

- **CG_FC_PERIPH_SIO_UART0:** Clock control for SIO/UART0
- **CG_FC_PERIPH_SIO_UART1:** Clock control for SIO/UART1
- **CG_FC_PERIPH_SIO_UART2:** Clock control for SIO/UART2
- **CG_FC_PERIPH_SIO_UART3:** Clock control for SIO/UART3
- **CG_FC_PERIPH_UART0:** Clock control for UART0
- **CG_FC_PERIPH_UART1:** Clock control for UART1
- **CG_FC_PERIPH_I2C0:** Clock control for I2C0
- **CG_FC_PERIPH_I2C1:** Clock control for I2C1
- **CG_FC_PERIPH_I2C2:** Clock control for I2C2
- **CG_FC_PERIPH_SSP0:** Clock control for SSP0
- **CG_FC_PERIPH_SSP1:** Clock control for SSP1
- **CG_FC_PERIPH_SSP2:** Clock control for SSP2
- **CG_FC_PERIPH_EBIF:** Clock control for EBIF
- **CG_FC_PERIPH_DMACA:** Clock control for DMAC A
- **CG_FC_PERIPH_DMACB:** Clock control for DMAC B
- **CG_FC_PERIPH_DMACC:** Clock control for DMAC C
- **CG_FC_PERIPH_DMAIF:** Clock control for DMACIF
- **CG_FC_PERIPH_ADC:** Clock control for ADC
- **CG_FC_PERIPH_WDT:** Clock control for WDT
- **CG_FC_PERIPH_MLA**: Supplies the clock to MLA
- **CG_FC_PERIPH_ESG:** Supplies the clock to ESG
- **CG_FC_PERIPH_SHA:** Supplies the clock to SHA
- **CG_FC_PERIPH_AES:** Supplies the clock to AES
- **CG_FC_PERIPHB_ALL:** ALL clock control

NewState

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

Description:
This function enables or disables supplying clock fsys to peripheries

Return:
None

5.2.3.36 CG_SetFs

Enable or disable external low-speed oscillator (fs) for RTC.

Prototype:
void

CG_SetFs(FunctionalState ***NewState***)

Parameters:
NewState
- **ENABLE**: to enable external low-speed oscillator for RTC.
- **DISABLE**: to disable external low-speed oscillator for RTC.

Description:
This enables or disables external low-speed oscillator (fs) for RTC.

Return:
None

5.2.4 Data Structure Description

5.2.4.1 CG_NMIFactor

Data Fields:
uint32_t
All specifies CGNMI source generation state.

Bit Fields:
uint32_t
WDT(Bit 0) means generated from WDT.
uint32_t
Reserved0 (Bit 1) Reserved
uint32_t
DetectLowVoltage(Bit 2) means generated when detect low voltage by LVD.
uint32_t
Reserved1 (Bit3~bit31) Reserved

5.2.4.2 CG_ResetFlag

Data Fields:
uint32_t
All specifies CG reset source.

Bit Fields:
uint32_t
ResetPin(Bit0) Reset from RESET pin

uint32_t
OSCFLF(Bit1) Flag for stopping of the internal high-speed oscillator or writing to the flash memory

uint32_t
WDTReset(Bit2) Reset from WDT

uint32_t
STOP2Reset(Bit3) Reset flag by STOP2 mode release

uint32_t
DebugReset(Bit4) Reset from SYSRESETREQ

uint32_t

Reserved0(Bit5)   Reserved

uint32_t
LVDReset(Bit6)   Rest by LVD

uint32_t
Reserved1(Bit7~Bit31)   Reserved

6. ESG

6.1 Overview

TOSHIBA TMPM46B has the entropy seed generator (ESG).
The entropy seed generator (ESG) is a circuit that generates a 512-bit entropy seed using a ring oscillator.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_esg.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_esg.h containing the macros, data types, structures and API definitions for use by applications.

6.2 API Functions

6.2.1 Function List

- Result ESG_Startup(void);
- Result ESG_SetLatchTiming(ESG_LatchTiming Value);
- uint32_t ESG_GetLatchTiming(void);
- Result ESG_SetFintiming(uint16_t Fintming);
- uint16_t ESG_GetFintiming(void);
- Result ESG_ClrInt(void);
- FunctionalState ESG_GetIntStatus(void);
- void ESG_IPReset(void);
- ESG_CalculationStatus ESG_GetCalculationStatus(void);
- void ESG_GetResult(uint32_t Seed[16U]);

6.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) ESG setting by ESG_SetLatchTiming(), ESG_GetLatchTiming(), ESG_SetFintiming(), ESG_GetFintiming(), ESG_ClrInt(), ESG_GetInStatus(), ESG_IPReset(), ESG_GetCalculationStatus() functions.
2) ESG module work or stop ESG_Startup() funcition.
3) Read data form register by ESG_GetResult() function.

6.2.3 Function Documentation

6.2.3.1 ESG_Startup

Startup ESG operation.

Prototype:
Result
ESG_Startup(void)

Parameters:
None

Description:
This function will startup ESG operation.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

6.2.3.2 ESG_SetLatchTiming

Set Entropy seed latch timing.

Prototype:
Result
ESG_SetLatchTiming(ESG_LatchTiming **Value**)

Parameters:
Value: The latch timing for ESG

Description:
This function will set Entropy seed latch timing.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

6.2.3.3 ESG_GetLatchTiming

Get Entropy seed latch timing.

Prototype:
uint32_t
ESG_GetLatchTiming(void)

Parameters:
none

Description:
This function will get Entropy seed latch timing.

Return:
The value of entropy seed latch timing.

6.2.3.4 ESG_SetFintiming

Set Entropy seed output timing.

Prototype:
Result
ESG_SetFintiming(uint16_t **Fintming**)

Parameters:
Fintming: the value of entropy seed output timing

Description:
This function will set Entropy seed output timing.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

6.2.3.5 ESG_GetFintiming

Get Entropy seed output timing.

Prototype:
uint16_t
ESG_GetFintiming(void)

Parameters:
none

Description:
This function will get Entropy seed output timing.

Return:
The value of entropy seed output timing.

6.2.3.6 ESG_ClrInt

Clear the ESG interrupt.

Prototype:
Result
ESG_ClrInt(void)

Parameters:
None

Description:
This function will clear the ESG interrupt.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

6.2.3.7 ESG_GetIntStatus

Get the ESG interrupt status.

Prototype:
FunctionalState
ESG_GetIntStatus(void)

Parameters:
None

Description:
This function will get the ESG interrupt status.

Return:
DISABLE means no interrupt.
ENABLE means interrupt is occurred.

6.2.3.8 ESG_IPReset

Reset ESG by peripheral function.

Prototype:
void
ESG_IPReset(void)

Parameters:
None

Description:
This function will reset ESG by peripheral function.

Return:
None

6.2.3.9 ESG_GetCalculationStatus

Get the calculation status.

Prototype:
ESG_CalculationStatus
ESG_GetCalculationStatus(void)

Parameters:
None

Description:
This function will get the calculation status.

Return:
ESG_CALCULATION_COMPLETE means calculation has completed.
ESG_CALCULATION_PROCESS means calculation is in process.

6.2.3.10 ESG_GetResult

Get the calculation result.

Prototype:
void
ESG_GetResult(uint32_t **Seed[16U]**)

Parameters:
Seed[16U]: A point that points to the value of calculation result.

Description:
This function will get the calculation result.

Return:
none

6.2.4 Data Structure Description

None

7. EXB

7.1 Overview

The TMPM46B 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/tmpm46b_exb.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_exb.h containing the macros, data types, structures and API definitions for use by applications.

7.2 API Functions

7.2.1 Function List

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

7.2.2 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 of EXB.
   EXB_Enable(), EXB_Disable().

7.2.3 Function Documentation

7.2.3.1 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_MULTIPLEX** for multiplex bus mode.

Description:
This function sets the bus mode for the external bus.

When ***BusMode*** is **EXB_BUS_MULTIPLEX**, the bus mode will be multiplex mode.

Return:
None

7.2.3.2 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

7.2.3.3 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
- **EXB_CS2**: for chip 2
- **EXB_CS3**: for chip 3

Description:
This function will enable the access to the specified chip.

Return:
None

7.2.3.4 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
- **EXB_CS2**: for chip 2
- **EXB_CS3**: for chip 3

Description:
This function will disable the access to the specified chip.

Return:
None

7.2.3.5 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
- **EXB_CS2**: for chip 2
- **EXB_CS3**: for chip 3

InitStruct is the structure containing basic EXB configuration including address space size, chip start address, data bus width , wait signal, wait function 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

7.2.4 Data Structure Description

7.2.4.1 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.

uint8_t
StartAddr Set the start address. The max value is 0xFF.

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)

7.2.4.2 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.

8. FC

8.1 Overview

TMPM46B device contains flash memory.
The size of flash is 1024Kbytes.

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\tmpm46b_fc.c with \Libraries\TX04_Periph_Driver\inc\tmpm46b_fc.h containing the API definitions for use by applications.

8.2 API Functions

8.2.1 Function List

- void FC_SetSecurityBit(FunctionalState ***NewState***)
- FunctionalState FC_GetSecurityBit(void)
- WorkState FC_GetBusyState(void)
- FunctionalState FC_GetBlockProtectState(uint8_t ***BlockNum***)
- FunctionalState FC_GetPageProtectState(uint8_t ***PageNum***)
- FunctionalState FC_GetAbortState(void)
- uint32_t FC_GetSwapSize(void);
- uint32_t FC_GetSwapState(void);
- void FC_SelectArea(uint8_t ***AreaNum***, FunctionalState ***NewState***);
- void FC_SetAbortion(void);
- void FC_ClearAbortion(void);
- void FC_SetClkDiv(uint8_t ***ClkDiv***);
- void FC_SetProgramCount(uint8_t ***ProgramCount***);
- void FC_SetEraseCounter(uint8_t ***EraseCounter***);
- FC_Result FC_ProgramBlockProtectState(uint8_t ***BlockNum***);
- FC_Result FC_ProgramPageProtectState(uint8_t ***PageNum***);
- FC_Result FC_EraseProtectState(void);
- FC_Result FC_WritePage(uint32_t ***PageAddr***, uint32_t * ***Data***);
- FC_Result FC_EraseBlock(uint32_t ***BlockAddr***);
- FC_Result FC_EraseArea(uint32_t ***AreaAddr***);
- FC_Result FC_ErasePage(uint32_t ***PageAddr***);
- FC_Result FC_EraseChip(void);
- FC_Result FC_SetSwpsrBit(uint8_t ***BitNum***);
- uint32_t FC_GetSwpsrBitValue(uint8_t ***BitNum***);

8.2.2 Detailed Description

Functions listed above can be divided into five 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(), FC_GetPageProtectState().
3) The functions change the protection status of each block:

FC_ProgramBlockProtectState(), FC_ProgramPageProtectState(), FC_EraseProtectState().

4) Use automatic operation command of flash.
FC_WritePage(), FC_EraseBlock(), FC_EraseChip(), FC_EraseArea(), FC_ErasePage(), FC_SetSwpsrBit().

5) Others:
FC_GetAbortState(), FC_GetSwapSize(), FC_GetSwapState(), FC_SelectArea(), FC_SetAbortion(), FC_ClearAbortion(), FC_SetClkDiv(), FC_SetProgramCount(), FC_SetEraseCounter(), FC_GetSwpsrBitValue().

8.2.3 Function Documentation

8.2.3.1 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 FCPSRx register) used for the write/erase-protection function are set to “1”.
2) The FCSECBIT <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 FCSECBIT<SEBIT> to “1”.

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

Return:
None

8.2.3.2 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.

8.2.3.3 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.

8.2.3.4 FC_GetBlockProtectState

Get the specified block protection state

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

Parameters:
BlockNum:The flash block number
FC_BLOCK_1 to FC_BLOCK_31

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

8.2.3.5 FC_GetPageProtectState

Get the page protection status.

Prototype:
FunctionalState
FC_GetPageProtectState(uint8_t ***PageNum***)

Parameters:
PageNum:The flash page number

- **FC_PAGE_0** to **FC_PAGE_7**

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

Return:
Page protection status.
DISABLE: Page is unprotected
ENABLE: Page is protected

8.2.3.6 FC_GetAbortState

Get the status of the auto operation is aborted or not

Prototype:
FunctionalState
FC_GetAbortState(void)

Parameters:
None

Description:
Once the auto operation is aborted by FCCR<WEABORT>, "1" is set to this bit.

Return:
The status of the auto operation is aborted or not:
DISABLE: The aborted is disabled
DONE: The aborted is enabled

8.2.3.7 FC_GetSwapSize

Get the swap size.

Prototype:
uint32_t
FC_GetSwapSize(void)

Parameters:
None

Description:
Get the swap size.

Return:
The swap size.

FC_SWAP_SIZE_4K: The swap size is 4K bytes.
FC_SWAP_SIZE_8K: The swap size is 8K bytes.
FC_SWAP_SIZE_16K: The swap size is 16K bytes.
FC_SWAP_SIZE_32K: The swap size is 32K bytes.
FC_SWAP_SIZE_512: Area 0 swaps with 512K bytes.

8.2.3.8 FC_GetSwapState

Get the swap state.

Prototype:
uint32_t
FC_GetSwapState(void)

Parameters:
None

Description:
Get the swap state.

Return:
The swap state.
FC_SWAP_RELEASE: Release the swap.
FC_SWAP_PROHIBIT: Setting is prohibited..
FC_SWAPPING: In swapping.
FC_SWAP_INITIAL: Release the swap (Initial state).

8.2.3.9 FC_SelectArea

Specifies an "area" in the Flash memory.

Prototype:
void
FC_SelectArea(uint8_t ***AreaNum*** ,
FunctionalState ***NewState***)

Parameters:
AreaNum:The flash area number
- **FC_AREA_0, FC_AREA_1, FC_AREA_ALL**

NewState: Specify area state.
This parameter can be one of the following values:
- **ENABLE**: Select the area.
- **DISABLE**: Unselect the area.

Description:
Specifies an "area" in the Flash memory that is targeted by Flash memory operation command

Return:
None

8.2.3.10 FC_SetAbortion

Set abortion of auto operation command.

Prototype:

void
FC_SetAbortion(void)

Parameters:
None

Description:
Set abortion of auto operation command.

Return:
None

8.2.3.11 FC_ClearAbortion

Clear FCSR<WEABORT> to "0" command.

Prototype:
void
FC_ClearAbortion(void)

Parameters:
None

Description:
Clear FCSR<WEABORT> to "0" command.

Return:
None

8.2.3.12 FC_SetClkDiv

Set Frequency division ratio to change the clock.

Prototype:
void
FC_SetClkDiv(uint8_t ***ClkDiv***)

Parameters:
ClkDiv: The divisor of the system clock

- **FC_Clk_Div_1** to **FC_Clk_Div_32**

Description:
Set Frequency division ratio to change the clock (WCLK: fsys/(DIV+1))n automatic operation to 8 to 12MHz.

Return:
None

8.2.3.13 FC_SetProgramCount

Set the number of counts that makes a programming time (CNT/WCLK) by automatic program execution command

Prototype:

void
FC_SetProgramCount(uint8_t ***ProgramCount***)

Parameters:
ProgramCount: the counter of the divided system clock for flash program

- **FC_PROG_CNT_250, FC_PROG_CNT_300, FC_PROG_CNT_350.**

Description:
Set the number of counts that makes a programming time (CNT/WCLK) by automatic program execution command be within the range of 20 to 40 μsec.

Return:
None

8.2.3.14 FC_SetEraseCounter

Set the number of counts until erase time (CNT/WCLK) will be 100 ~ 130msec using each auto erase command.

Prototype:
void
FC_SetEraseCounter (uint8_t ***EraseCounter***)

Parameters:
EraseCounter: the number of counts until erase time (CNT/WCLK) will be 100~130msec using each auto erase command

- **FC_ERAS_CNT_85, FC_ERAS_CNT_90,**
- **FC_ERAS_CNT_95, FC_ERAS_CNT_100,**
- **FC_ERAS_CNT_105, FC_ERAS_CNT_110**
- **FC_ERAS_CNT_115, FC_ERAS_CNT_120,**
- **FC_ERAS_CNT_125, FC_ERAS_CNT_130,**
- **FC_ERAS_CNT_135, FC_ERAS_CNT_140.**

Description:
Set the number of counts until erase time (CNT/WCLK) will be 100 ~ 130msec using each auto erase command.

Return:
None

8.2.3.15 FC_ProgramBlockProtectState

Program the protection bits.

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

Parameters:

- **FC_BLOCK_1** to **FC_BLOCK_31**

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.

8.2.3.16 FC_ProgramPageProtectState

Program the protection bits.

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

Parameters:
- **FC_PAGE_0** to **FC_PAGE_7**

Description:
This API is used to set the protection bit to “1” so that the corresponding page 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 page protection bit operation over time error.

8.2.3.17 FC_EraseProtectState

Erase the protection bits.

Prototype:
FC_Result
FC_EraseBlockProtectState(void)

Parameters:
None

Description:
This API is used to erase the protection bits (clear them to”0”) so that the whole flash will not be protected.
The whole flash protection bit 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 page protection bits operation over time error.

8.2.3.18 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.

Description:
This API is used to write data to specified page.
It contains 1024 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**:
1 An attempt to rewrite a page two or more times without erasing the content can cause damages to the device.
2 For programing into the Flash memory after entering into Normal mode from Stop2 mode, it is required to confirm that CGRSTFLG<OSCFLF> is read as "1".

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 or page is protected. The write operation can’t be executed.
FC_ERROR_OVER_TIME: Write operation over time error.

8.2.3.19 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.

8.2.3.20 FC_EraseArea

Erase the content of specified area.

Prototype:
FC_Result
FC_EraseArea(uint32_t ***AreaAddr***)

Parameters:
AreaAddr: The block starts address.

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

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

8.2.3.21 FC_ErasePage

Erase the content of specified page.

Prototype:
FC_Result
FC_ErasePage(uint32_t ***PageAddr***)

Parameters:
PageAddr: The page starts address.

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

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

8.2.3.22 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.

8.2.3.23 FC_SetSwpsrBit

Setting values of FCSWPSR[10:0] by memory swap command.

Prototype:
FC_Result
FC_SetSwpsrBit(uint8_t ***BitNum***)

Parameters:
BitNum: The FCSWPSR bit number to be set
This parameter can be one of the following values:
FC_SWPSR_BIT_0 to FC_SWPSR_BIT_10

Description:
Setting values of FCSWPSR[10:0] by memory swap command. Automatic memory swap is a command to write "1" to each bit of FCSWPSR[10:0] in the units of 1-bit. A bit cannot be set to "0" independently. All bits should be cleared to "0" using automatic protect bit erase command.

Return:
Result of the operation to set the FCSWPSR bit.
FC_SUCCESS: Set the FCSWPSR bit successfully.
FC_ERROR_OVER_TIME: Set the FCSWPSR bit operation over time error.

8.2.3.24 FC_GetSwpsrBitValue

Get the value of the special bit of FCSWPSR[10:0].

Prototype:
uint32_t
FC_GetSwpsrBitValue(uint8_t ***BitNum***)

Parameters:
BitNum: The special bit of SWPSR.
This parameter can be one of the following values:
FC_SWPSR_BIT_0 to FC_SWPSR_BIT_10

Description:
Get the value of the special bit of FCSWPSR[10:0].

Return:
The value returned can be one of the following values:

FC_BIT_VALUE_0, FC_BIT_VALUE_1.

8.2.4 Data Structure Description

None

9. FUART

9.1 Overview

TOSHIBA TMPM46B contains the Asynchronous serial channel (Full UART) with Modem control.
TOSHIBA TMPM46B 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/tmpm46b_fuart.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_fuart.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 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***, FUART_Direction ***Direction***)
- void FUART_SetIrDADivisor(TSB_FUART_TypeDef * ***FUARTx***, uint32_t ***Divisor***)
- 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_SetIrDAEncodeMode(
  TSB_FUART_TypeDef * ***FUARTx***, uint32_t ***Mode***)
- Result FUART_EnableIrDA(TSB_FUART_TypeDef * ***FUARTx***)
- void FUART_DisableIrDA(TSB_FUART_TypeDef * ***FUARTx***)
- 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***)
- void FUART_SetRTSStatus(
  TSB_FUART_TypeDef * ***FUARTx***, FUART_ModemStatus ***Status***)
- void FUART_SetDTRStatus(
  TSB_FUART_TypeDef * ***FUARTx***, FUART_ModemStatus ***Status***)

9.2.2 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(), FUART_SetRTSStatus(), FUART_SetDTRStatus().
5) Configure IrDA.
   FUART_EnableIrDA(), FUART_DisableIrDA(), FUART_SetIrDAEncodeMode(), FUART_SetIrDADivisor().

9.2.3 Function Documentation

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

9.2.3.1 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

9.2.3.2 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

9.2.3.3 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

9.2.3.4 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

9.2.3.5 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.

9.2.3.6 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

9.2.3.7 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

9.2.3.8 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.

9.2.3.9 FUART_SetIrDADivisor

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

Prototype:
void
FUART_SetIrDADivisor(TSB_FUART_TypeDef * ***FUARTx***,
uint32_t ***Divisor***)

Parameters:
FUARTx: The specified Full UART channel.
Divisor: The IrDA Low-power divisor (from 0x01 to 0xFF)

Description:
This API will set IrDA Low-power divisor to generate the IrLPBaud16 signal by dividing down of UARTCLK.
This API must be executed before the IrDA circuit is enabled.

Return:
None

9.2.3.10 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

9.2.3.11 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

9.2.3.12 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

9.2.3.13 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

9.2.3.14 FUART_SetIrDAEncodeMode

Select IrDA encoding mode for transmitting 0 bits.

Prototype:
void
FUART_SetIrDAEncodeMode(TSB_FUART_TypeDef * ***FUARTx***,
uint32_t ***Mode***)

Parameters:
FUARTx: The specified Full UART channel.
Mode: IrDA encoding mode select for transmitting 0 bits.

- **FUART_IRDA_3_16_BIT_PERIOD_MODE**: 0 bits are transmitted as an active high pulse of 3/16th of the bit period.
- **FUART_IRDA_3_TIMES_IRLPBAUD16_MODE**: 0 bits are transmitted with a pulse width that is 3 times the period of the IrLPBaud16 input signal.

Description:
This API selects IrDA encoding mode. Change IrDA encoding mode to **FUART_IRDA_3_TIMES_IRLPBAUD16_MODE** can reduce power consumption but might decrease transmission distance.

Return:
None

9.2.3.15 FUART_EnableIrDA

Enable the IrDA circuit.

Prototype:
Result
FUART_EnableIrDA(TSB_FUART_TypeDef * ***FUARTx***)

Parameters:
FUARTx: The specified Full UART channel.

Description:
If Full UART is enabled, this API will enable IrDA circuit. If Full UART is disabled, this API will do nothing and return an error.

Return:
SUCCESS: Enable IrDA circuit successfully.
ERROR: The UART channel is disabled, cannot enable IrDA circuit.

9.2.3.16 FUART_DisableIrDA

Disable the IrDA circuit.

Prototype:
void
FUART_DisableIrDA(TSB_FUART_TypeDef * ***FUARTx***)

Parameters:
FUARTx: The specified Full UART channel.

Description:
If Full UART is enabled, this API will disable IrDA circuit. If Full UART is disabled, this API will do nothing, IrDA circuit doesn't work originally.

Return:
None

9.2.3.17 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 UARTTXINTR and UARTRXINTR are generated. The interrupts are generated based on a transition through a level rather than based on the level.

Return:
None

9.2.3.18 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_RIN_MODEM_INT_MASK**: Enable RIN interrupt
- **FUART_CTS_MODEM_INT_MASK**: Enable CTS modem interrupt
- **FUART_DCD_MODEM_INT_MASK**: Enable DCD modem interrupt
- **FUART_DSR_MODEM_INT_MASK**: Enable DSR 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

9.2.3.19 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).

9.2.3.20 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).

9.2.3.21 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).

9.2.3.22 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

9.2.3.23 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

9.2.3.24 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 tranmit/ receive FIFO by using DMAC.

Return:
None

9.2.3.25 FUART_GetModemStatus

Get all the Modem Status, include: CTS, DSR, DCD, RIN, DTR, and RTS.

Prototype:
FUART_AllModemStatus
FUART_GetModemStatus(TSB_FUART_TypeDef * ***FUARTx***)

Parameters:
FUARTx: The specified Full UART channel.

Description:
This API will get all the Modem Status, include: CTS, DSR, DCD, RIN, DTR, and RTS.

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

9.2.3.26 FUART_SetRTSStatus

Set the Full UART RTS(Request To Send) modem status output.

Prototype:
void
FUART_SetRTSStatus(TSB_FUART_TypeDef * ***FUARTx***,
FUART_ModemStatus ***Status***)

Parameters:
FUARTx: The specified Full UART channel.
Status: RTS modem status output.

- **FUART_MODEM_STATUS_0**: The modem status output is 0.
- **FUART_MODEM_STATUS_1**: The modem status output is 1.

Description:
This API will set the Full UART RTS(Request To Send) modem status output.

Return:
None

9.2.3.27 FUART_SetDTRStatus

Set the Full UART DTR(Data Transmit Ready) modem status output.

Prototype:
void
FUART_SetDTRStatus(TSB_FUART_TypeDef * ***FUARTx***,
FUART_ModemStatus ***Status***)

Parameters:
FUARTx: The specified Full UART channel.
Status: DTR modem status output.

- **FUART_MODEM_STATUS_0**: The modem status output is 0.
- **FUART_MODEM_STATUS_1**: The modem status output is 1.

Description:
This API will set the Full UART DTR(Data Transmit Ready) modem status output.

Return:
None

9.2.4 Data Structure Description

9.2.4.1 FUART_InitTypeDef

Data Fields:
uint32_t
BaudRate configures the Full UART communication baud rate, it can’t be 0(bsp) and must be smaller than 2950000(bps).
uint32_t
DataBits specifies data bits per transfer, which can be set as:

- **FUART_DATA_BITS_5** for 5-bit mode
- **FUART_DATA_BITS_6** for 6-bit mode
- **FUART_DATA_BITS_7** for 7-bit mode

- **FUART_DATA_BITS_8** for 8-bit mode

uint32_t
StopBits specifies the length of stop bit transmission, which can be set as:

- **FUART_STOP_BITS_1** for 1 stop bit
- **FUART_STOP_BITS_2** for 2 stop bits

uint32_t
Parity specifies the parity mode, which can be set as:

- **FUART_NO_PARITY** for no parity
- **FUART_0_PARITY** for 0 parity
- **FUART_1_PARITY** for 1 parity
- **FUART_EVEN_PARITY** for even parity
- **FUART_ODD_PARITY** for odd parity

uint32_t
Mode enables or disables reception, transmission or both, which can be set as:

- **FUART_ENABLE_TX** for enabling transmission
- **FUART_ENABLE_RX** for enabling reception
- **FUART_ENABLE_TX | FUART_ENABLE_RX** for enabling both reception and transmition

uint32_t
FlowCtrl Enable or disable the hardware flow control, which can be set as:

- **FUART_NONE_FLOW_CTRL** for no flow control
- **FUART_CTS_FLOW_CTRL** for enabling CTS flow control
- **FUART_RTS_FLOW_CTRL** for enabling RTS flow control
- **FUART_CTS_FLOW_CTRL | FUART_RTS_FLOW_CTRL** for enabling both CTS and RTS flow control

9.2.4.2 FUART_INTStatus

Data Fields:

uint32_t
All: Full UART interrupt status or mask.

Bit

uint32_t ***RIN***: 1	RIN modem interrupt
uint32_t ***CTS***: 1	CTS modem interrupt
uint32_t ***DCD***: 1	DCD modem interrupt
uint32_t ***DSR***: 1	DSR modem interrupt
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

9.2.4.3 FUART_AllModemStatus

Data Fields:
uint32_t
All: Full UART All Modem Status
Bit
uint32_t
CTS: 1 CTS modem status
uint32_t
DSR: 1 DSR modem status
uint32_t
DCD: 1 DCD modem status
uint32_t
Reserved1: 5 Reserved
uint32_t
RI: 1 RIN modem status
uint32_t
Reserved2: 1 Reserved
uint32_t
DTR: 1 DTR modem status
uint32_t
RTS: 1 RTS modem status
uint32_t
Reserved3: 20 Reserved

10. GPIO

10.1 Overview

For TOSHIBA TMPM46B 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/tmpm46b_gpio.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_gpio.h containing the macros, data types, structures and API definitions for use by applications.

10.2 API Functions

10.2.1 Function List

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

10.2.2 Detailed Description

Functions listed above can be divided into three parts:

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

10.2.3 Function Documentation

10.2.3.1 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.

Description:
This function will read specified GPIO Data register.

Return:
The value read from DATA register.

10.2.3.2 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.

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.

10.2.3.3 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.

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

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

Return:
None

10.2.3.4 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.

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

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

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

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

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

Return:
None

10.2.3.5 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.

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

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

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

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

Return:
None

10.2.3.6 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.

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

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

- Combination of the effective bits.

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

Return:
None

10.2.3.7 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.

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

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

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

Note: To use the Port J as an analog input of the AD converter, disable input on PJIE and disable pull-up on PJPUP.

Return:
None

10.2.3.8 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.

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

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

NewState:

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

Description:

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

Return:

None

10.2.3.9 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.

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

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

NewState:

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

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

Return:
None

10.2.3.10 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.

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

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

NewState:

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

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

Return:
None

10.2.3.11 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_PA:** GPIO port A.

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

- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_ALL:** GPIO pin[0:1],
- Combination of the effective bits.

NewState:

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

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

Return:

None

10.2.3.12 GPIO_SetOpenDrain

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

Prototype:

void
GPIO_SetOpenDrain(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***,
FunctionalState ***NewState***)

Parameters:

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

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PK:** GPIO port K.
- **GPIO_PL:** GPIO port L.

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

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

NewState:

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

Description:

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

Return:

None

10.2.3.13 GPIO_EnableFuncReg

Enable specified GPIO function.

Prototype:

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

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

- **GPIO_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_PJ:** GPIO port J.
- **GPIO_PK :** GPIO port K.
- **GPIO_PL:** GPIO port L.

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

- **GPIO_FUNC_REG_1** for GPIO function register 1,
- **GPIO_FUNC_REG_2** for GPIO function register 2,
- **GPIO_FUNC_REG_3** for GPIO function register 3,
- **GPIO_FUNC_REG_4** for GPIO function register 4,
- **GPIO_FUNC_REG_5** for GPIO function register 5,
- **GPIO_FUNC_REG_6** for GPIO function register 6

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

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

Description:
This function will enable GPIO pin specified function.

Return:
None

10.2.3.14 GPIO_DisableFuncReg

Disable specified GPIO function.
Prototype:
void
GPIO_DisableFuncReg(GPIO_Port ***GPIO_x***,
uint8_t ***FuncReg_x***,
uint8_t ***Bit_x***)

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

- **GPIO_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_PJ:** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PL:** GPIO port L.

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

- **GPIO_FUNC_REG_1** for GPIO function register 1,
- **GPIO_FUNC_REG_2** for GPIO function register 2,
- **GPIO_FUNC_REG_3** for GPIO function register 3,
- **GPIO_FUNC_REG_4** for GPIO function register 4,
- **GPIO_FUNC_REG_5** for GPIO function register 5,
- **GPIO_FUNC_REG_6** for GPIO function register 6

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

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

Description:

This function will disable GPIO pin specified function.

Return:

None

10.2.4 Data Structure Description

10.2.4.1 GPIO_InitTypeDef

Data Fields:

uint8_t

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

- **GPIO_INPUT:** Set GPIO pin as input port
- **GPIO_OUTPUT:** Set GPIO pin as output port
- **GPIO_IO_MODE_NONE:** Don’t change GPIO pin I/O mode.

uint8_t

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

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

uint8_t

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

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

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

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

10.2.4.2 GPIO_RegTypeDef

Data Fields:

uint8_t
PinDATA Port x data register, port data read and write by this variable.

uint8_t
PinCR Port x output control register.

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

uint8_t
PinFR[FRMAX] Function setting register. You will be able to use the functions assigned by setting "1"

uint8_t
PinOD Port x open drain control register.

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

uint8_t
PinPUP Port x pull-up control register:

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

uint8_t
PinPDN Port x pull-down control register :

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

uint8_t
PinPIE Port x input control register:

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

10.2.4.3 TSB_Port_TypeDef

Data Fields:

__IO uint32_t
DATA The "DATA" can be read and written

__IO uint32_t
PinCR The "CR" can be read and written.

__IO uint32_t
PinFR[FRMAX] The "FR[FRMAX]" can be read and written

uint32_t
RESERVED0[RESER] Reserved

__IO uint32_t
PinOD The "OD" can be read and written

__IO uint32_t
PinPUP The "PUP" can be read and written

__IO uint32_t

PinPDN The “PDN” can be read and written
uint32_t
RESERVED1[RESER] Reserved
__IO uint32_t
PinPIE Port x input control register

11. I2C

11.1 Overview

The TMPM46B contains I2C Bus Interface with 3 channels (I2C0~2).
The I2C bus is connected to external devices via SCL and SDA, and it can communicate with multiple devices.
Data can be transferred in free data format by the I2C channels. In free data format, data is always sent by master-transmitter and received by slave-receiver.

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

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_i2c.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_i2c.h containing the macros, data types, structures and API definitions for use by applications.

11.2 API Functions

11.2.1 Function List

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

11.2.2 Detailed Description

Functions listed above can be divided into four parts:

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

11.2.3 Function Documentation

Note**: in all of the following APIs, parameter “TSB_I2C_TypeDef* ***I2Cx” can be one of the following values:
TSB_I2C0, TSB_I2C1, TSB_I2C2

11.2.3.1 I2C_SetACK

Enable or disable the generation of ACK clock.

Prototype:
void
I2C_SetACK(TSB_I2C_TypeDef* ***I2Cx***,
FunctionalState ***NewState***)

Parameters:
I2Cx is the specified I2C channel.
NewState sets the generation of ACK clock, which can be:

- **ENABLE** for generating of ACK clock
- **DISABLE** for no ACK clock

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

Return:
None

11.2.3.2 I2C_Init

Initialize the specified I2C channel in I2C mode.

Prototype:
void
I2C_Init(TSB_I2C_TypeDef* ***I2Cx***,
I2C_InitTypeDef* ***InitI2CStruct***)

Parameters:
I2Cx is the specified I2C channel.
InitI2CStruct is the structure containing I2C configuration (refer to Data Structure Description for details).

Description:
This function will initialize and configure the self-address, bit length of transfer data, clock division, the generation of ACK clock and the operation mode of I2C transfer for the specified I2C channel selected by ***I2Cx***.

Return:
None

11.2.3.3 I2C_SetBitNum

Specify the number of bits per transfer.

Prototype:

void
I2C_SetBitNum(TSB_I2C_TypeDef* ***I2Cx***,
uint32_t ***I2CBitNum***)

Parameters:
I2Cx is the specified I2C channel.
I2CBitNum specifies the number of bits per transfer, max. 8.
This parameter can be one of the following values:

- **I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;
- **I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;
- **I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;
- **I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;
- **I2C_DATA_LEN_7**, which means that the data length number of bits per transfer is 7.

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

Return:
None

11.2.3.4 I2C_SWReset

Reset the state of the specified I2C channel.

Prototype:
void
I2C_SWReset(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

Return:
None

11.2.3.5 I2C_ClearINTReq

Clear I2C interrupt request in I2C bus mode.

Prototype:
void
I2C_ClearINTReq(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

Return:
None

11.2.3.6 I2C_GenerateStart

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

Prototype:
void
I2C_GenerateStart(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

Return:
None

11.2.3.7 I2C_GenerateStop

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

Prototype:
void
I2C_GenerateStop(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

Return:
None

11.2.3.8 I2C_GetState

Get the I2C channel state in I2C bus mode.

Prototype:
I2C_State
I2C_GetState(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

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

11.2.3.9 I2C_SetSendData

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

Prototype:
void
I2C_SetSendData(TSB_I2C_TypeDef* ***I2Cx***,
uint32_t ***Data***)

Parameters:
I2Cx is the specified I2C channel.
Data is a byte-data to be sent. The maximum value is 0xFF.

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

Return:
None

11.2.3.10 I2C_GetReceiveData

Get data received from the specified I2C channel.

Prototype:
uint32_t
I2C_GetReceiveData(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

Return:

Data which has been received

11.2.3.11 I2C_SetFreeDataMode

Set I2C channel working in I2C free data mode.

Prototype:
void
I2C_SetFreeDataMode(TSB_I2C_TypeDef* ***I2Cx***,
FunctionalState ***NewState***)

Parameters:
I2Cx is the specified I2C channel.
NewState specifies the state of the I2C when system is idle mode, which can be

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

Description:
The specified I2C channel can transfer data in free data format by calling this function. In free data format, master device always transmits data while slave device always receives data. If the I2C is needed to shift to transfer data in normal I2C format, call **I2C_Init()**.

Return:
None

11.2.3.12 I2C_GetSlaveAddrMatchState

Get slave address match detection state.

Prototype:
FunctionalState
I2C_ GetSlaveAddrMatchState(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

Description:
Get slave address match detection state.

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

11.2.3.13 I2C_SetPrescalerClock

Set prescaler clock of the specified I2C channel.

Prototype:
void
I2C_SetPrescalerClock(TSB_I2C_TypeDef* ***I2Cx***,
uint32_t ***PrescalerClock***)

Parameters:
I2Cx is the specified I2C channel.
PrescalerClock is the prescaler clock value.
This parameter can be one of the following values:

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

Description:
This function will set prescaler clock of the specified I2C channel,
The system clock(fsys) is divided according to ***PrescalerClock*** as the prescaler clock(fprsck), and the prescaler clock is further divided by ***I2CClkDiv*** (refer to Data Structure Description for details).and used as the serial clock for I2C transfer,
Make sure the prescaler clock in the range between 50ns and 150ns.

Return:
None

11.2.3.14 I2C_SetINTReq

Enable or disable interrupt request of the I2C channel.

Prototype:
void
I2C _SetINTReq(TSB_I2C_TypeDef* ***I2Cx***,
FunctionalState ***NewState***)

Parameters:
I2Cx is the specified I2C channel.
NewState: Specify I2C interrupt setting
This parameter can be one of the following values:

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

Description:
This function will enable or disable I2C interrupt request.

Return:
None

11.2.3.15 I2C_GetINTStatus

Get interrupt generation state.

Prototype:
FunctionalState
I2C_GetINTStatus(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

Description:
This function will get the state of I2C interrupt generation.

Return:
The state of interrupt generation.

ENABLE: I2C interrupt has been generated.
DISABLE: I2C has not interrupted.

11.2.3.16 I2C_ClearINTOutput

Clear the I2C interrupt output.

Prototype:
void
I2C_ClearINTOutput(TSB_I2C_TypeDef* ***I2Cx***)

Parameters:
I2Cx is the specified I2C channel.

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

Return:
None

11.2.4 Data Structure Description

11.2.4.1 I2C_InitTypeDef

Data Fields:
uint32_t
I2CSelfAddr specifies self-address of the I2C channel in I2C mode, the last bit of which can not be 1 and max. 0xFE.
uint32_t
I2CDataLen Specify data length of the I2C channel in I2C mode, which can be set as:

- **I2C_DATA_LEN_8**, which means that the data length number of bits per transfer is 8;
- **I2C_DATA_LEN_1**, which means that the data length number of bits per transfer is 1;
- **I2C_DATA_LEN_2**, which means that the data length number of bits per transfer is 2;
- **I2C_DATA_LEN_3**, which means that the data length number of bits per transfer is 3;
- **I2C_DATA_LEN_4**, which means that the data length number of bits per transfer is 4;
- **I2C_DATA_LEN_5**, which means that the data length number of bits per transfer is 5;
- **I2C_DATA_LEN_6**, which means that the data length number of bits per transfer is 6;
- **I2C_DATA_LEN_7**, which means that the data length number of bits per transfer is 7.

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

- **I2C_SCK_CLK_DIV_20**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 20;
- **I2C_SCK_CLK_DIV_24**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 24;

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

uint32_t
PrescalerClkDiv specifies the division of the system clock for generating the fprsck, which can be set as:

- **I2C_PRESCALER_DIV_1**, which means that the frequency of the prescaler clock is quotient of fsys divided by 1
- **I2C_PRESCALER_DIV_2**, which means that the frequency of the prescaler clock is quotient of fsys divided by 2
- **I2C_PRESCALER_DIV_3**, which means that the frequency of the prescaler clock is quotient of fsys divided by 3
- **I2C_PRESCALER_DIV_4**, which means that the frequency of the prescaler clock is quotient of fsys divided by 4
- **I2C_PRESCALER_DIV_5**, which means that the frequency of the prescaler clock is quotient of fsys divided by 5
- **I2C_PRESCALER_DIV_6**, which means that the frequency of the prescaler clock is quotient of fsys divided by 6
- **I2C_PRESCALER_DIV_7**, which means that the frequency of the prescaler clock is quotient of fsys divided by 7
- **I2C_PRESCALER_DIV_8**, which means that the frequency of the prescaler clock is quotient of fsys divided by 8
- **I2C_PRESCALER_DIV_9**, which means that the frequency of the prescaler clock is quotient of fsys divided by 9
- **I2C_PRESCALER_DIV_10**, which means that the frequency of the prescaler clock is quotient of fsys divided by 10
- **I2C_PRESCALER_DIV_11**, which means that the frequency of the prescaler clock is quotient of fsys divided by 11
- **I2C_PRESCALER_DIV_12**, which means that the frequency of the prescaler clock is quotient of fsys divided by 12
- **I2C_PRESCALER_DIV_13**, which means that the frequency of the prescaler clock is quotient of fsys divided by 13
- **I2C_PRESCALER_DIV_14**, which means that the frequency of the prescaler clock is quotient of fsys divided by 14
- **I2C_PRESCALER_DIV_15**, which means that the frequency of the prescaler clock is quotient of fsys divided by 15
- **I2C_PRESCALER_DIV_16**, which means that the frequency of the prescaler clock is quotient of fsys divided by 16
- **I2C_PRESCALER_DIV_17**, which means that the frequency of the prescaler clock is quotient of fsys divided by 17
- **I2C_PRESCALER_DIV_18**, which means that the frequency of the prescaler clock is quotient of fsys divided by 18
- **I2C_PRESCALER_DIV_19**, which means that the frequency of the prescaler clock is quotient of fsys divided by 19
- **I2C_PRESCALER_DIV_20**, which means that the frequency of the prescaler clock is quotient of fsys divided by 20

- **I2C_PRESCALER_DIV_21**, which means that the frequency of the prescaler clock is quotient of fsys divided by 21
- **I2C_PRESCALER_DIV_22**, which means that the frequency of the prescaler clock is quotient of fsys divided by 22
- **I2C_PRESCALER_DIV_23**, which means that the frequency of the prescaler clock is quotient of fsys divided by 23
- **I2C_PRESCALER_DIV_24**, which means that the frequency of the prescaler clock is quotient of fsys divided by 24
- **I2C_PRESCALER_DIV_25**, which means that the frequency of the prescaler clock is quotient of fsys divided by 25
- **I2C_PRESCALER_DIV_26**, which means that the frequency of the prescaler clock is quotient of fsys divided by 26
- **I2C_PRESCALER_DIV_27**, which means that the frequency of the prescaler clock is quotient of fsys divided by 27
- **I2C_PRESCALER_DIV_28**, which means that the frequency of the prescaler clock is quotient of fsys divided by 28
- **I2C_PRESCALER_DIV_29**, which means that the frequency of the prescaler clock is quotient of fsys divided by 29
- **I2C_PRESCALER_DIV_30**, which means that the frequency of the prescaler clock is quotient of fsys divided by 30
- **I2C_PRESCALER_DIV_31**, which means that the frequency of the prescaler clock is quotient of fsys divided by 31
- **I2C_PRESCALER_DIV_32**, which means that the frequency of the prescaler clock is quotient of fsys divided by 32

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

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

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

11.2.4.2 I2C_State

Data Fields:
uint32_t
All specifies state data in I2C mode

Bit Fields:
uint32_t
LastRxBit specifies last received bit monitor.

uint32_t
GeneralCall specifies general call detected monitor.

uint32_t
SlaveAddrMatch specifies slave address match monitor.

uint32_t
ArbitrationLost specifies arbitration last detected monitor.

uint32_t
INTReq specifies Interrupt request monitor.

uint32_t
BusState specifies bus busy flag.

uint32_t
TRx specifies transfer or Receive selection monitor.

uint32_t
MasterSlave specifies master or slave selection monitor.

12. IGBT

12.1 Overview

TMPM46B contains 4 channels multi-purpose timer (MPT). MPT can operate in IGBT mode.

There are the following functions in IGBT mode:

1) 16-bit programmable square-wave output mode (PPG, two waves)
2) External trigger starting
3) Period matching detection function
4) Emergency stop function
5) Synchronous start mode

The IGBT driver APIs provide a set of functions to control IGBT module, including setting start mode, operation mode, counter state, source clock division, initial output level, trigger/EMG noise elimination division, changing output active/inactive timing, output wave period, EMG output and so on.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_igbt.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_igbt.h containing the macros, data types, structures and API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- void IGBT_Enable(TSB_MT_TypeDef* ***IGBTx***)
- void IGBT_Disable(TSB_MT_TypeDef* ***IGBTx***)
- void IGBT_SetClkInCoreHalt(TSB_MT_TypeDef* ***IGBTx***, uint8_t ***ClkState***)
- void IGBT_SetSWRunState(TSB_MT_TypeDef* ***IGBTx***, uint8_t ***Cmd***)
- uint16_t IGBT_GetCaptureValue(TSB_MT_TypeDef* ***IGBTx***, uint8_t ***CapReg***)
- void IGBT_Init(TSB_MT_TypeDef* ***IGBTx***, IGBT_InitTypeDef* ***InitStruct***)
- void IGBT_Recount(TSB_MT_TypeDef* ***IGBTx***)
- void IGBT_ChangeOutputActiveTiming(TSB_MT_TypeDef* ***IGBTx***, uint8_t ***Output,*** uint16_t ***Timing***)
- void IGBT_ChangeOutputInactiveTiming(TSB_MT_TypeDef* ***IGBTx***, uint8_t ***Output,*** uint16_t ***Timing***)
- void IGBT_ChangePeriod(TSB_MT_TypeDef* ***IGBTx***, uint16_t ***Period***)
- WorkState IGBT_GetCntState(TSB_MT_TypeDef* ***IGBTx***)
- Result IGBT_CancelEMGState(TSB_MT_TypeDef* ***IGBTx***)
- IGBT_EMGStateTypeDef IGBT_GetEMGState(TSB_MT_TypeDef * ***IGBTx***)
- void IGBT_ChangeTrgValue(TSB_MT_TypeDef****IGBTx***, uint16_t ***uTrgCnt***)
- void IGBT_ClSynSlaveChCounter(TSB_MT_TypeDef * ***IGBTx***)

12.2.2 Detailed Description

Functions listed above can be divided into five parts:

1) Initialize and configure the common functions of each IGBT channel are handled by IGBT_Enable(), IGBT_Disable() and IGBT_Init().
2) The counter state and control of each IGBT channel are handled by IGBT_SetClkInCoreHalt(), IGBT_SetSWRunState(), IGBT_Recount() and IGBT_GetCntState().
3) Changing Operation parameters and getting captured value of each IGBT channel are handled by IGBT_GetCaptureValue(), IGBT_ChangeOutputActiveTiming(), IGBT_ChangeOutputInactiveTiming() and IGBT_ChangePeriod().

4) Getting and cancel the EMG protection state is handled by IGBT_GetEMGState () and IGBT_CancelEMGState().
5) Change the Trigger value counter and synchronous counter clearing setting are handled by IGBT_ChangeTrgValue() and IGBT_SetSynCounterClearConfig().

12.2.3 Function Documentation

Note**: in all of the following APIs, parameter "TSB_MT_TypeDef* ***IGBTx" can be one of the following values:
IGBT0, IGBT1, IGBT2 or **IGBT3**.

12.2.3.1 IGBT_Enable

Enable the specified MPT channel in IGBT mode.

Prototype:
void
IGBT_Enable(TSB_MT_TypeDef* ***IGBTx***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.

Description:
This function will enable the specified MPT channel selected by ***IGBTx***. After calling this API, the MPT operates in IGBT mode and the specified channel should be initialized and configured by **IGBT_Init()**.

Return:
None

12.2.3.2 IGBT_Disable

Disable the specified MPT channel operating in IGBT mode.

Prototype:
void
IGBT_Disable(TSB_MT_TypeDef* ***IGBTx***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.

Description:
This function will disable the specified MPT channel selected by ***IGBTx***.

Return:
None

12.2.3.3 IGBT_SetClkInCoreHalt

Set the clock stop or not in Core Halt when the specified MPT channel operates in IGBT mode.

Prototype:
void
IGBT_SetClkInCoreHalt(TSB_MT_TypeDef* ***IGBTx***,
uint8_t ***ClkState***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.
ClkState specify the control in Core Halt during debug mode, which can be one of the values below:

- **IGBT_RUNNING_IN_CORE_HALT**, clock does not stop and outputs are not controlled,
- **IGBT_STOP_IN_CORE_HALT**, clock stops and output are controlled in accordance with configuration.

Description:
In Core Halt during debug mode, the clock of IGBT mode can stop or keep running, which depends on ***ClkState.*** It's highly recommended to select **IGBT_STOP_IN_CORE_HALT** as ***ClkState.***

Return:
None

12.2.3.4 IGBT_SetSWRunState

Start or stop the counter in IGBT mode by software command.

Prototype:
void
IGBT_SetSWRunState(TSB_MT_TypeDef* ***IGBTx,***
uint8_t ***Cmd***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.
Cmd is the command of controlling the counter, which can be one of the values below,

- **IGBT_RUN**, the command of starting the counter,
- **IGBT_STOP**, the command of stopping the counter.

Description:
This function can start or stop the counter by software.

Return:
None

***Note:**
1) The actual timing of counter starting or stopping depends on the configuration of IGBT mode. If **IGBT_CMD_START** or **IGBT_CMD_START_NO_START_INT** is selected as ***StartMode*** (refer to Data Structure Description for details), this API can fully control the counter. If ***StartMode*** is set as other values, trigger can also control the counter.
2) If **IGBT_FALLING_TRG_START** or **IGBT_RISING_TRG_START** is selected as ***StartMode*** (refer to Data Structure Description for details), after the initialization and configuration is complete, the software start command, **IGBT_SetSWRunState(*IGBTx*, IGBT_RUN)**, must be issued before the start trigger. Then the counter can be started by the trigger.
3) When EMG input is low level and EMG interrupt occurs, please use **IGBT_SetSWRunState(*IGBTx*, IGBT_STOP)** to stop the counter.

12.2.3.5 IGBT_GetCaptureValue

Get the captured counter value in IGBT mode.

Prototype:
uint16_t
IGBT_GetCaptureValue(TSB_MT_TypeDef* ***IGBTx,***
uint8_t ***CapReg***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.
CapReg selects the capture register, which can be one of the values below,

- **IGBT_CAPTURE_0**, capture register 0,
- **IGBT_CAPTURE_1**, capture register 1.

Description:
This function returns the value captured and stored in the specified capture register.

Return:
The value captured

***Note:**
Only when **IGBT_CMD_START** or **IGBT_CMD_START_NO_START_INT** is selected as ***StartMode*** (refer to Data Structure Description for details), the counter value can be captured and be got by calling this function. The timing of the first input edge will be captured and stored in capture register 0. And the timing of the second input edge will be captured and stored in the capture register 1.

12.2.3.6 IGBT_Init

Initialize and configure the specified MPT channel in IGBT mode.

Prototype:
void
IGBT_Init(TSB_MT_TypeDef* ***IGBTx***,
IGBT_InitTypeDef* ***InitStruct***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.
InitStruct is the structure containing IGBT configuration including start mode, operation mode, output in stop state, start trigger acceptance mode, interrupt period, source clock division, initialization of output0/1, noise elimination time division for trigger input and EMG input, active and inactive timing of output0/1, the period of IGBT output wave and EMG function setting (refer to Data Structure Description for details).

Description:
After enabling the IGBT mode by calling **IGBT_Enable()**, this function can be used to initialize and configure the specified MPT channel in IGBT mode.

Return:
None

***Note:**

The corresponding I/O ports must be set as EMG input pins when MPTs operate in IGBT mode.
Call this function when **IGBT_CancelEMGState()** returns **SUCCESS** only, otherwise the initialization and configuration will not take effect.

12.2.3.7 IGBT_Recount

Clear and restart the counter.

Prototype:
void
IGBT_Recount(TSB_MT_TypeDef* ***IGBTx***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.

Description:
While the counter is running, call this function will make counter restart counting from 0.

Return:
None

12.2.3.8 IGBT_ChangeOutputActiveTiming

Change the active timing of IGBT output 0 or output 1.

Prototype:
void
IGBT_ChangeOutputActiveTiming(TSB_MT_TypeDef* ***IGBTx***,
uint8_t ***Output,***
uint16_t ***Timing***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.
Output selects the IGBT output port, which can be

- **IGBT_OUTPUT_0**, IGBT output port 0,
- **IGBT_OUTPUT_1**, IGBT output port 1.

Timing specifies the new output active timing. The value must be set between 0 and the output inactive timing.

Description:
This function is used to change the active timing of output. But the new active timing will take effect after the counter matches the period value.

Return:
None

12.2.3.9 IGBT_ChangeOutputInactiveTiming

Change the inactive timing of IGBT output 0 or output 1.

Prototype:
void

```
IGBT_ChangeOutputInactiveTiming(TSB_MT_TypeDef* IGBTx,
                                uint8_t Output,
                                uint16_t Timing)
```

Parameters:

IGBTx is the specified MPT channel in IGBT mode.

Output selects the IGBT output port, which can be

- **IGBT_OUTPUT_0**, IGBT output port 0,
- **IGBT_OUTPUT_1**, IGBT output port 1.

Timing specifies the new output inactive timing. The value must be set between the output active timing and the ***Period***.

Description:

This function is used to change the inactive timing of output. But the new inactive timing will take effect after the counter matches the period value.

Return:

None

12.2.3.10 IGBT_ChangePeriod

Change the period of IGBT output.

Prototype:

```
void
IGBT_ChangePeriod(TSB_MT_TypeDef* IGBTx,
                  uint16_t Period)
```

Parameters:

IGBTx is the specified MPT channel in IGBT mode.

Period specifies the new output period. The value must be set between the output inactive timing and 0xFFFF.

Description:

This function is used to change the period of output. But the new period will take effect after the counter matches the previous period value.

Return:

None

12.2.3.11 IGBT_GetCntState

Get the counter state.

Prototype:

```
WorkState
IGBT_GetCntState(TSB_MT_TypeDef* IGBTx)
```

Parameters:

IGBTx is the specified MPT channel in IGBT mode.

Description:

This function is used to get the counter state.

Return:

The counter state, which can be:
BUSY, the counter is running.
DONE, the counter stops.

12.2.3.12 IGBT_CancelEMGState

Cancel the EMG state of IGBT.

Prototype:
Result
IGBT_CancelEMGState(TSB_MT_TypeDef* ***IGBTx***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.

Description:
This function is used to cancel the EMG state of IGBT. Before canceling the EMG state, call **IGBT_GetCntState()** to check the state of the counter and make sure that the EMG input level is H.
If the counter is running (**IGBT_GetCntState()** returns **BUSY**) or the EMG input is driven to L, it returns **ERROR** and the EMG state is not cancelled.
If the counter stops (**IGBT_GetCntState()** returns **DONE**) and the EMG input is driven to H, it cancels the EMG state and returns **SUCCESS**.

Return:
The result of EMG state canceling, which can be
SUCCESS, EMG state of IGBT is cancelled.
ERROR, EMG state of IGBT is not cancelled.

12.2.3.13 IGBT_GetEMGState

Get the EMG state of IGBT.

Prototype:
IGBT_EMGStateTypeDef
IGBT_GetEMGState(TSB_MT_TypeDef * ***IGBTx***)

Parameters:
IGBTx is the specified MPT channel in IGBT mode.

Description:
This function is used to get the EMG state of IGBT, which includes EMG input pin status after noise elimination and EMG protection status.

Return:
The EMG status enumeration structure ***IGBT_EMGStateTypeDef*** of EMG state (refer to Data Structure Description for details).

12.2.3.14 IGBT_ChangeTrgValue

Change the Trigger value of IGBT output.

Prototype:
void

IGBT_ChangeTrgValue(TSB_MT_TypeDef****IGBTx***, uint16_t ***uTrgCnt***)

Parameters:

IGBTx is the specified MPT channel in IGBT mode.

uTrgCnt is the new IGBT up-counter.

Description:

Change the Trigger value of IGBT output.

Return:

None

12.2.3.15 IGBT_SetSynCounterClearConfig

Set synchronous counter clearing setting.

Prototype:

void
IGBT_SetSynCounterClearConfig(TSB_MT_TypeDef * ***IGBTx,***
uint8_t ***SynClrMode***)

Parameters:

IGBTx is the specified MPT channel in IGBT mode.

SynClrMode specify the synchronous counter clearing mode, which can be one of the values below:

- **IGBT_SYNCLR_UPCN_ENABLE**, the up-counters on slave channel are cleared synchronously with those of the master channel,
- **IGBT_SYNCLR_UPCN_DISABLE**, the up-counter of each channel are not cleared synchronously.

Description:

This function is used to set synchronous counter clearing setting of slave channels in the synchronous start mode.

Return:

None

12.2.4 Data Structure Description

12.2.4.1 IGBT_InitTypeDef

Data Fields:

uint8_t

StartMode selects start mode of counter, which could be

- **IGBT_CMD_START**, counter is controlled by software command and the timing of input edge can be captured.
- **IGBT_CMD_START_NO_START_INT**, counter is controlled by software command and the timing of input edge can be captured. No interrupt occurs when counter starts.
- **IGBT_CMD_FALLING_TRG_START**. There are 2 ways to start the counter. One is to issue the software start command during the trigger driven to low level. The other is a falling edge input to the trigger after the software start command is issued.
- **IGBT_CMD_FALLING_TRG_START_NO_START_INT**, the ways to start the counter is same as **IGBT_CMD_FALLING_TRG_START**, but no interrupt occurs when counter is started by software command.

- **IGBT_CMD_RISING_TRG_START**, There are 2 ways to start the counter. One is to issue the software start command during the trigger driven to high level. The other is a rising edge input to the trigger after the software start command is issued.
- **IGBT_CMD_RISING_TRG_START_NO_START_INT**, the ways to start the counter is same as **IGBT_CMD_RISING_TRG_START**, but no interrupt occurs when counter is started by software command.
- **IGBT_FALLING_TRG_START**, only falling trigger edge can start the counter. On the other hand a rising trigger edge can stop the counter (*See Note).
- **IGBT_RISING_TRG_START**, only rising trigger edge can start the counter. On the other hand a falling trigger edge can stop the counter (*See Note).
- **IGBT_SYNSLAVE_CHNL_START**, Synchronous start (sets only slave channels (*See Note).

uint8_t

OperationMode selects IGBT operation mode, which can be set as:

- **IGBT_CONTINUOUS_OUTPUT**, IGBT operates in continuous output mode.
- **IGBT_ONE_TIME_OUTPUT**, IGBT operates in one-time output mode.

uint8_t

CntStopState specifies the output state when counter stops, which can be set as:

- **IGBT_OUTPUT_INACTIVE**, IGBT outputs inactive level.
- **IGBT_OUTPUT_MAINTAINED**, IGBT outputs do not change.
- **IGBT_OUTPUT_NORMAL**, counter does not stop until the end of the period, except a stop trigger, and outputs shift to inactive level when the counter stops.

FunctionalState

ActiveAcceptTrg selects whether the start trigger is accepted when output is active level. This parameter can be set as:

- **ENABLE**, trigger will always be accepted.
- **DISABLE**, trigger will be ignored during active output.

uint8_t

INTPeriod specifies the interrupt occurrence period, which can be set as:

- **IGBT_INT_PERIOD_1**, interrupt occurs every one IGBT output period.
- **IGBT_INT_PERIOD_2**, interrupt occurs every two IGBT output periods.
- **IGBT_INT_PERIOD_4**, interrupt occurs every four IGBT output periods.

uint8_t

ClkDiv selects the division of IGBT source clock, which can be set as:

- **IGBT_CLK_DIV_1**, the frequency of IGBT source clock equals to φT0.
- **IGBT_CLK_DIV_2**, the frequency of IGBT source clock is quotient of φT0 divided by 2.
- **IGBT_CLK_DIV_4**, the frequency of IGBT source clock is quotient of φT0 divided by 4.
- **IGBT_CLK_DIV_8**, the frequency of IGBT source clock is quotient of φT0 divided by 8.

uint8_t

Output0Init, initialize the IGBT output 0, which can be set as:

- **IGBT_OUTPUT_DISABLE**, disable IGBT output.
- **IGBT_OUTPUT_HIGH_ACTIVE**, initial output is low level and high level is the active output.
- **IGBT_OUTPUT_LOW_ACTIVE**, initial output is high level and low level is the active output.

uint8_t

Output1Init, initialize the IGBT output 1, which can be set as:

- **IGBT_OUTPUT_DISABLE**, disable IGBT output.

- **IGBT_OUTPUT_HIGH_ACTIVE**, initial output is low level and high level is the active output.
- **IGBT_OUTPUT_LOW_ACTIVE**, initial output is high level and low level is the active output.

uint8_t
TrgDenoiseDiv selects the division of noise elimination time for trigger input in IGBT mode. This parameter can be set as:

- **IGBT_NO_DENOISE**, no noise elimination.
- **IGBT_DENOISE_DIV_16**, eliminate pulses shorter than 16 / fsys.
- **IGBT_DENOISE_DIV_32**, eliminate pulses shorter than 32 / fsys.
- **IGBT_DENOISE_DIV_48**, eliminate pulses shorter than 48 / fsys.
- **IGBT_DENOISE_DIV_64**, eliminate pulses shorter than 64 / fsys.
- **IGBT_DENOISE_DIV_80**, eliminate pulses shorter than 80 / fsys.
- **IGBT_DENOISE_DIV_96**, eliminate pulses shorter than 96 / fsys.
- **IGBT_DENOISE_DIV_112**, eliminate pulses shorter than 112 / fsys.
- **IGBT_DENOISE_DIV_128**, eliminate pulses shorter than 128 / fsys.
- **IGBT_DENOISE_DIV_144**, eliminate pulses shorter than 144 / fsys.
- **IGBT_DENOISE_DIV_160**, eliminate pulses shorter than 160 / fsys.
- **IGBT_DENOISE_DIV_176**, eliminate pulses shorter than 176 / fsys.
- **IGBT_DENOISE_DIV_192**, eliminate pulses shorter than 192 / fsys.
- **IGBT_DENOISE_DIV_208**, eliminate pulses shorter than 208 / fsys.
- **IGBT_DENOISE_DIV_224**, eliminate pulses shorter than 224 / fsys.
- **IGBT_DENOISE_DIV_240**, eliminate pulses shorter than 240 / fsys.

uint16_t
Output0ActiveTiming specifies the active timing of output 0. The value must be set between 0 and Output0InactiveTiming.

uint16_t
Output0InactiveTiming specifies the active timing of output 0. The value must be set between Output0ActiveTiming and Period.

uint16_t
Output1ActiveTiming specifies the active timing of output 1. The value must be set between 0 and Output1InactiveTiming.

uint16_t
Output1InactiveTiming specifies the active timing of output 1. The value must be set between Output1ActiveTiming and Period.

uint16_t
Period specifies the IGBT output period, max. 0xFFFF.

uint8_t
EMGFunction specifies the EMG stop function. This parameter can be set as:

- **IGBT_DISABLE_EMG**, disable IGBT EMG stop function.
- **IGBT_EMG_OUTPUT_INACTIVE**, IGBT outputs inactive level during EMG state.
- **IGBT_EMG_OUTPUT_HIZ**, IGBT outputs Hi-z during EMG state.

uint8_t
EMGDenoiseDiv selects the division of noise elimination time for EMG input in IGBT mode. This parameter can be set as:

- **IGBT_NO_DENOISE**, no noise elimination.
- **IGBT_DENOISE_DIV_16**, eliminate pulses shorter than 16 / fsys.
- **IGBT_DENOISE_DIV_32**, eliminate pulses shorter than 32 / fsys.
- **IGBT_DENOISE_DIV_48**, eliminate pulses shorter than 48 / fsys.
- **IGBT_DENOISE_DIV_64**, eliminate pulses shorter than 64 / fsys.
- **IGBT_DENOISE_DIV_80**, eliminate pulses shorter than 80 / fsys.
- **IGBT_DENOISE_DIV_96**, eliminate pulses shorter than 96 / fsys.
- **IGBT_DENOISE_DIV_112**, eliminate pulses shorter than 112 / fsys.
- **IGBT_DENOISE_DIV_128**, eliminate pulses shorter than 128 / fsys.
- **IGBT_DENOISE_DIV_144**, eliminate pulses shorter than 144 / fsys.

- **IGBT_DENOISE_DIV_160**, eliminate pulses shorter than 160 / fsys.
- **IGBT_DENOISE_DIV_176**, eliminate pulses shorter than 176 / fsys.
- **IGBT_DENOISE_DIV_192**, eliminate pulses shorter than 192 / fsys.
- **IGBT_DENOISE_DIV_208**, eliminate pulses shorter than 208 / fsys.
- **IGBT_DENOISE_DIV_224**, eliminate pulses shorter than 224 / fsys.
- **IGBT_DENOISE_DIV_240**, eliminate pulses shorter than 240 / fsys.

***Note:**
To use trigger to start the counter, a software start command must be issued at first.
To use the synchronous start mode, set "11" to MTxIGCR<IGSTA[1:0]> on the slave channels (**IGBT1**, **IGBT2** or **IGBT3**) and set other than "11" to the master channel (**IGBT0**).

12.2.4.2 IGBT_EMGStateTypeDef

Data Fields:
enum
IGBT_EMGInputState indicates the EMG input pin status after noise elimination, which could be

- **IGBT_EMG_INPUT_LOW**, EMG input pin after noise elimination is low.
- **IGBT_EMG_INPUT_HIGH**, EMG input pin after noise elimination is high.

enum
IGBT_EMGProtectState indicates the EMG protection status, which could be

- **IGBT_EMG_NORMAL**, EMG protection status is in normal operation.
- **IGBT_EMG_PROTECT**, EMG protection status is during in protection.

13. LVD

13.1 Overview

TMPM46B has Low voltage detection circuit (LVD). The voltage detection circuit generates a reset signal or an interrupt signal by detecting a decreasing/increasing voltage.

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

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_lvd.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_lvd.h containing the macros, data types, structures and API definitions for use by applications.

13.2 API Functions

13.2.1 Function List

- void LVD_EnableVD(void)
- void LVD_DisableVD(void)
- void LVD_SetVDLevel(uint32_t ***VDLevel***)
- LVD_VDStatus LVD_GetVDStatus(void)
- void LVD_SetVDResetOutput(FunctionalState ***NewState***)
- void LVD_SetVDINTOutput(FunctionalState ***NewState***)

13.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) Configure LVD are handled by LVD_EnableVD(), LVD_DisableVD(), LVD_SetVDLevel(), LVD_SetVDResetOutput(), LVD_SetVD1INTOutput().
2) Get the power supply voltage detection status info by LVD_GetVDStatus().

13.2.3 Function Documentation

13.2.3.1 LVD_EnableVD

Enable the operation of voltage detection.

Prototype:
void
LVD_EnableVD(void)

Parameters:
None.

Description:
This function will enable the voltage detection operation.

Return:
None.

13.2.3.2 LVD_DisableVD

Disable the operation of voltage detection.

Prototype:

void
LVD_DisableVD(void)

Parameters:
None.

Description:
This function will disable the voltage detection operation.

Return:
None.

13.2.3.3 LVD_SetVDLevel

Select the detection voltage level.

Prototype:
void
LVD_SetVDLevel(uint32_t ***VDLevel***)

Parameters:
VDLevel is the voltage detection level.
This parameter can be one of the following values:

- **LVD_VDLVL_280:** Voltage detection level is from 2.80 ± 0.1V.
- **LVD_VDLVL_285:** Voltage detection level is from 2.85 ± 0.1V.
- **LVD_VDLVL_290:** Voltage detection level is from 2.90 ± 0.1V.
- **LVD_VDLVL_295:** Voltage detection level is from 2.95 ± 0.1V.
- **LVD_VDLVL_300:** Voltage detection level is from 3.00 ± 0.1V.
- **LVD_VDLVL_305:** Voltage detection level is from 3.05 ± 0.1V.
- **LVD_VDLVL_310:** Voltage detection level is from 3.10 ± 0.1V.
- **LVD_VDLVL_315:** Voltage detection level is from 3.15 ± 0.1V.

Description:
This function will set the level of voltage detection.

Return:
None.

13.2.3.4 LVD_GetVDStatus

Get voltage detection status.

Prototype:
LVD_VDStatus
LVD_GetVDStatus(void)

Parameters:
None.

Description:
This function will get voltage detection status.

Return:
LVD_VDStatus: The voltage detection status, which can be one of:
LVD_VD_UPPER: Power supply voltage is upper than the detection voltage.

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

13.2.3.5 LVD_SetVDResetOutput

Enable or disable LVD reset output of voltage detection.

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

Parameters:
NewState: new state of LVD reset output.
This parameter can be one of the following values:
ENABLE or **DISABLE**

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

Return:
None.

13.2.3.6 LVD_SetVDINTOutput

Enable or disable LVD interrupt output of voltage detection.

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

Parameters:
NewState: new state of LVD interrupt output.
This parameter can be one of the following values:
ENABLE or **DISABLE**

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

Return:
None.

13.2.4 Data Structure Description

None

14. MLA

14.1 Overview

TOSHIBA TMPM46B contains an MLA processor (MLA: Multiple Length Arithmetic Coprocessor). The Multiple Length Arithmetic coprocessor (MLA) performs calculation for Elliptic Curve Cryptography (ECC) with 256-bit key length.

The MLA supports 3 algorithms below:
- Montgomery multiplication (256bit)
- Multiple length addition
- Multiple length subtraction

The MLA drivers API provide a set of functions to configure MLA, including such parameters as data block number, calculation result data, input data, output data, calculation mode setting, Montgomery parameter setting, operation setting, calculation status, carry and borrow flag status and so on.

This driver is contained in \Libraries\TX04_Periph_Driver\src\tmpm46b_mla.c, with \Libraries\TX04_Periph_Driver\inc\tmpm46b_mla.h containing the API definitions for use by applications.

14.2 API Functions

14.2.1 Function List

- void MLA_SetCalculationMode(uint32_t ***CalculationMode***);
- MLA_CalculationMode MLA_GetCalculationMode(void);
- void MLA_SetADataBlkNum(uint8_t ***BlkNum***);
- uint8_t MLA_GetADataBlkNum(void);
- void MLA_SetBDataBlkNum(uint8_t ***BlkNum***);
- uint8_t MLA_GetBDataBlkNum(void);
- void MLA_SetWDataBlkNum(uint8_t ***BlkNum***);
- uint8_t MLA_GetWDataBlkNum(void);
- MLA_CarryBorrowFlag MLA_GetCarryBorrowFlag(void);
- MLA_CalculationStatus MLA_GetCalculationStatus(void);
- Result MLA_SetMontgomeryParameter(uint32_t ***Data***);
- uint32_t MLA_GetMontgomeryParameter(void);
- Result MLA_WriteDataBlkNum(uint8_t ***BlkNum***, uint32_t ***Data[8U]***);
- void MLA_ReadDataBlkNum(uint8_t ***BlkNum***);
- void MLA_IPReset(void)

14.2.2 Detailed Description

Functions listed above can be divided into three parts:
1) The MLA basic configuration is handled by the MLA_SetCalculationMode(), MLA_SetADataBlkNum(), MLA_SetBDataBlkNum(), MLA_SetBDataWlkNum(), MLA_SetMontgomeryParameter() and MLA_WriteDataBlkNum() functions.
2) The MLA operation result and status are got by the MLA_GetCalculationMode(), MLA_GetADataBlkNum(), MLA_GetBDataBlkNum(), MLA_GetWDataBlkNum(), MLA_GetCarryBorrowFlag(), MLA_GetCalculationStatus(), MLA_GetMontgomeryParameter() and MLA_ReadDataBlkNum() functions.
3) The MLA peripheral function reset is handled by the MLA_IPReset() functions.

14.2.3 Function Documentation

14.2.3.1 MLA_SetCalculationMode

Set the calculation mode.

Prototype:
void
MLA_SetCalculationMode(uint32_t ***CalculationMode***)

Parameters:
CalculationMode**:** Specify the calculation mode
This parameter can be one of the following values:

- **MLA_COM_MODE_MUL**: Montgomery multiplication (256bit).
- **MLA_COM_MODE_ADD**: Multiple length addition.
- **MLA_COM_MODE_SUB**: Multiple length subtraction.

Description:
This function will set the calculation mode.

Return:
None

14.2.3.2 MLA_GetCalculationMode

Get the calculation mode.

Prototype:
MLA_CalculationMode
MLA_GetCalculationMode(void)

Parameters:
None

Description:
This function will get the calculation mode.

Return:
calculation mode.
MLA_CalculationMode_MUL: Montgomery multiplication (256bit).
MLA_CalculationMode_MUL: Multiple length addition.
MLA_CalculationMode_SUB: Multiple length subtraction.

14.2.3.3 MLA_SetADataBlkNum

Set a data block number that is substituted into "a".

Prototype:
void
MLA_SetADataBlkNum(uint8_t ***BlkNum***)

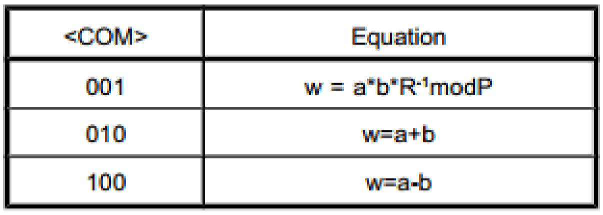
Parameters:
BlkNum: Data block number.
This parameter can be one of the following values:

- **MLA_BLK_0** to **MLA_BLK_31**.

Description:
This function will set a data block number that is substituted into "a".

***Note:**
Set data block numbers, which are substituted into to "a", "b", and "w" in the

following equations, to <SRC1>, <SRC2>, and <RDB> respectively.



<COM>	Equation
001	$w = a*b*R^{-1}modP$
010	w=a+b
100	w=a-b

Return:
None

14.2.3.4 MLA_GetADataBlkNum

Get the data block number that is substituted into "a".

Prototype:
uint8_t
MLA_GetADataBlkNum(void)

Parameters:
None

Description:
This function will get the data block number that is substituted into "a".

Return:
Data block number.
MLA_BLK_0 to MLA_BLK_31, or MLA_BLK_UNKNOWN.

14.2.3.5 MLA_SetBDataBlkNum

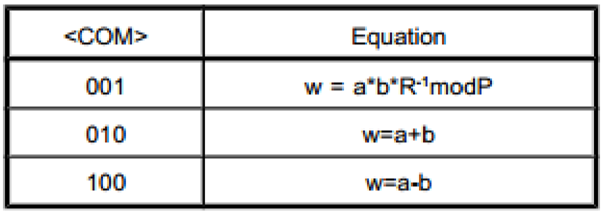
Set a data block number that is substituted into "b".

Prototype:
void
MLA_SetBDataBlkNum(uint8_t ***BlkNum***)

Parameters:
BlkNum: Data block number.
This parameter can be one of the following values:
➢ **MLA_BLK_0** to **MLA_BLK_31**.

Description:
This function will set a data block number that is substituted into "b".

***Note:**
Set data block numbers, which are substituted into to "a", "b", and "w" in the following equations, to <SRC1>, <SRC2>, and <RDB> respectively.



<COM>	Equation
001	$w = a*b*R^{-1}modP$
010	w=a+b
100	w=a-b

Return:

None

14.2.3.6 MLA_GetBDataBlkNum

Get the data block number that is substituted into "b".

Prototype:
uint8_t
MLA_GetBDataBlkNum(void)

Parameters:
None

Description:
This function will get the data block number that is substituted into "b".

Return:
Data block number.
MLA_BLK_0 to MLA_BLK_31, or MLA_BLK_UNKNOWN.

14.2.3.7 MLA_SetWDataBlkNum

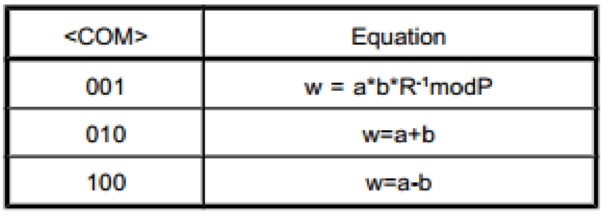
Set a data block number that is substituted into "w".

Prototype:
void
MLA_SetWDataBlkNum(uint8_t ***BlkNum***)

Parameters:
BlkNum: Data block number.
This parameter can be one of the following values:
- **MLA_BLK_0** to **MLA_BLK_31, or MLA_BLK_UNKNOWN**.

Description:
This function will set a data block number that is substituted into "w".

***Note:**
Set data block numbers, which are substituted into to "a", "b", and "w" in the following equations, to <SRC1>, <SRC2>, and <RDB> respectively.



<COM>	Equation
001	$w = a*b*R^{-1}modP$
010	$w=a+b$
100	$w=a-b$

Return:
None

14.2.3.8 MLA_GetWDataBlkNum

Get the data block number that is substituted into "w".

Prototype:
uint8_t

MLA_GetWDataBlkNum(void)

Parameters:
None

Description:
This function will get the data block number that is substituted into "w".

Return:
Data block number.
MLA_BLK_0 to MLA_BLK_31, or MLA_BLK_UNKNOWN.

14.2.3.9 MLA_GetCarryBorrowFlag

Get carry and borrow flag status.

Prototype:
MLA_CarryBorrowFlag
MLA_GetCarryBorrowFlag(void)

Parameters:
None

Description:
This function will get carry and borrow flag status.

Return:
MLA_CARRYBORROW_NO: No carry or borrow flag
MLA_CARRYBORROW_OCCURS: A carry or borrow flag occurs.

14.2.3.10 MLA_GetCalculationStatus

Get the status of the calculation.

Prototype:
MLA_CalculationStatus
MLA_GetCalculationStatus(void)

Parameters:
None

Description:
This function will get the status of the calculation.

Return:
MLA_CALCULATION_STOP: Stop.
MLA_CALCULATION_PROGRESS: Calculation in progress.

14.2.3.11 MLA_SetMontgomeryParameter

Set montgomery parameter.

Prototype:
Result
MLA_SetMontgomeryParameter(uint32_t ***Data***)

Parameters:
Data**:** Montgomery parameter, max 0xFFFFFFFF.

Description:
This function will set montgomery parameter.

Return:
None.

14.2.3.12 MLA_GetMontgomeryParameter

Get montgomery parameter.

Prototype:
uint32_t
MLA_GetMontgomeryParameter(void)

Parameters:
None

Description:
This function will get montgomery parameter.

Return:
Montgomery parameter.

14.2.3.13 MLA_WriteDataBlkNum

Write data to the specified data block number.

Prototype:
Result
MLA_WriteDataBlkNum(uint8_t ***BlkNum,*** uint32_t ***Data[8U]***)

Parameters:
BlkNum: Data block number.
This parameter can be one of the following values:

- **MLA_BLK_0** to **MLA_BLK_31**.

Data[8U]: Calculation input data.

Description:
This function will write data to the specified data block number.

Return:
SUCCESS means write successful.
ERROR means write failed.

14.2.3.14 MLA_ReadDataBlkNum

Read data from the specified data block number.

Prototype:
void
MLA_ReadDataBlkNum(uint8_t ***BlkNum***, uint32_t ***Result[8U]***)

Parameters:
BlkNum: Data block number.

This parameter can be one of the following values:

- **MLA_BLK_0** to **MLA_BLK_31**.

Description:
This function will get data from the specified data block number..

Return:
Output data.

14.2.3.15 MLA_IPReset

Reset MLA by peripheral function.

Prototype:
void
MLA_IPReset(void)

Parameters:
None

Description:
This function will reset MLA by peripheral function.

Return:
None

14.2.4 Data Structure Description

None

15. RTC

15.1 Overview

The Real Time Clock (RTC) in the TMPM46B 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 (alarm output)
- Alarm interrupt
- Clock correction function
- 1 Hz clock output

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/tmpm46b_rtc.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_rtc.h containing the macros, data types, structures and API definitions for use by applications.

15.2 API Functions

15.2.1 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_ResetAlarm(void);
- void RTC_ResetClockSec(void);
- RTC_ReqState RTC_GetResetClockSecReq(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***);
- void RTC_SetProtectCtrl(FunctionalState ***NewState***)
- void RTC_EnableCorrection(void)
- void RTC_DisableCorrection(void)
- void RTC_SetCorrectionTime(uint8_t ***Time***)
- void RTC_SetCorrectionValue(RTC_CorrectionMode ***Mode***,uint16_t ***Cnt***)

15.2.2 Detailed Description

Functions listed above can be divided into six 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_GetResetClockSecReq(), RTC_SetClockValue() and RTC_GetClockValue() handle for RTC clock function only.
4) RTC_EnableAlarm(), RTC_DisableAlarm(), RTC_SetAlarmValue(),RTC_ResetAlarm() and RTC_GetAlarmValue() handle for RTC alarm function only.
5) RTC_EnableCorrection(),RTC_DisableCorrection(),RTC_SetCorrectionTime() and RTC_SetCorrectionValue() handle for RTC clock correction function.
6) RTC_SetAlarmOutput(),RTC_SetProtectCtrl() and RTC_SetRTCINT() handle other specified functions.

15.2.3 Function Documentation

15.2.3.1 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.

15.2.3.2 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

15.2.3.3 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

15.2.3.4 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

15.2.3.5 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.

15.2.3.6 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

15.2.3.7 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 bet 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

15.2.3.8 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

15.2.3.9 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

15.2.3.10 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

15.2.3.11 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

15.2.3.12 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

15.2.3.13 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

15.2.3.14 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

15.2.3.15 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

15.2.3.16 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

15.2.3.17 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

15.2.3.18 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.

15.2.3.19 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

15.2.3.20 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.

15.2.3.21 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

15.2.3.22 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.

15.2.3.23 RTC_EnableClock

Enable RTC clock function.

Prototype:
void
RTC_EnableClock(void);

Parameters:
None

Description:
This function will enable clock function.

Return:
None

15.2.3.24 RTC_DisableClock

Disable RTC clock function.

Prototype:
void
RTC_DisableClock(void);

Parameters:
None

Description:
This function will disable clock function.

Return:
None

15.2.3.25 RTC_EnableAlarm

Enable RTC alarm function.

Prototype:
void
RTC_EnableAlarm(void);

Parameters:
None

Description:
This function will enable alarm function.

Return:
None

15.2.3.26 RTC_DisableAlarm

Disable RTC alarm function.

Prototype:
void
RTC_DisableAlarm(void);

Parameters:
None

Description:
This function will disable alarm function.

Return:
None

15.2.3.27 RTC_SetRTCINT

Enable or disable INTRTC.

Prototype:
void
RTC_SetRTCINT(FunctionalState ***NewState***);

Parameters:
NewState: New state of INTRTC.
- **ENABLE**: Enable INTRTC.
- **DISABLE**: Disable INTRTC.

Description:
This function will enable RTCINT when ***NewState*** is **ENABLE**, and disable RTCINT when ***NewState*** is **DISABLE**.

***Note:**
To set interrupt enable bits to <ENATMR>, <ENAALM> and <INTENA>, you must follow the order specified here. Make sure not to set them at the same time (make sure that there is time lag between interrupt enable and clock/alarm enable). To change the setting of <ENATMR> and <ENAALM>, <INTENA> must be disabled first.

Return:
None

15.2.3.28 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

15.2.3.29 RTC_ResetAlarm

Reset alarm.

Prototype:
void
RTC_ResetAlarm(void);

Parameters:
None

Description:
This function will reset alarm.

Return:
None

15.2.3.30 RTC_ResetClockSec

Reset RTC clock second counter.

Prototype:
void
RTC_ResetClockSec(void);

Parameters:
None

Description:
This function will reset sec counter.

Return:
None

15.2.3.31 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.

15.2.3.32 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

15.2.3.33 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

15.2.3.34 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

15.2.3.35 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

15.2.3.36 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

15.2.3.37 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

15.2.3.38 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

15.2.3.39 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

15.2.3.40 RTC_SetProtectCtrl

Enable or disable to protect RTC registers: RTCADJCTL and RTCADJDAT

Prototype:
void
RTC_SetProtectCtrl(FunctionalState ***NewState***);

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

Description:
This function will enable or disable to protect RTC registers: RTCADJCTL and RTCADJDAT
Return:
None

15.2.3.41 RTC_EnableCorrection

Enable RTC correction function.

Prototype:
void
RTC_EnableCorrection(void);

Parameters:
None

Description:
This function will enable RTC correction function.

Return:
None

15.2.3.42 RTC_DisableCorrection

Disable RTC correction function.

Prototype:

void
RTC_DisableCorrection(void);

Parameters:
None

Description:
This function will disable RTC correction function.

Return:
None

15.2.3.43 RTC_SetCorrectionTime

Set correction reference time.

Prototype:
void
RTC_SetCorrectionTime(uint8_t ***Time***);

Parameters:
Time: The reference time of correction
This parameter can be one of the following values:

- **RTC_ADJ_TIME_1_SEC**: correction reference time is 1 second.
- **RTC_ADJ_TIME_10_SEC**: correction reference time is 10 seconds.
- **RTC_ADJ_TIME_20_SEC**: correction reference time is 20 seconds.
- **RTC_ADJ_TIME_30_SEC**: correction reference time is 30 seconds.
- **RTC_ADJ_TIME_1_MIN**: correction reference time is 1 minute.

Description:
This function will set correction reference time.

Return:
None

15.2.3.44 RTC_SetCorrectionValue

Set correction value.

Prototype:
void
RTC_SetCorrectionValue(RTC_CorrectionMode ***Mode***,uint16_t ***Cnt***);

Parameters:
Mode: the mode of correction
This parameter can be one of the following values:

- **RTC_CORRECTION_PLUS**: a plus correction is applied.
- **RTC_CORRECTION_MINUS**: a minus correction is applied.

Cnt: a correction value per second.
For **RTC_CORRECTION_PLUS**, this parameter can only be 0~255.
For **RTC_CORRECTION_ MINUS**, this parameter can only be 1~256.

Description:

This function will set correction value.

Return:
None

15.2.4 Data Structure Description

15.2.4.1 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.

15.2.4.2 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.

15.2.4.3 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.

16. SHA

16.1 Overview

TOSHIBA TMPM46B contains an SHA processor (SHA: Secure Hash Algorithm). The SHA processor generates fixed length (256-bit) Hash values from message data.

The SHA processor has the following features:
- Conforms to FIPS PUB 180-3 Secure Hash standard Algorithm (SHA2).
Supports SHA-224/SHA-256
- Message length
Up to (2^{61} - 1) bytes. Calculations are performed in unit of 512 bits.
- Automatic padding
- Halting or restarting of calculation
Thanks to stacking results of calculation in progress, calculation can be restarted.

The SHA drivers API provide a set of functions to configure SHA, including such parameters as run state, interrupt setting, Hash initial mode, Hash initial value, DMA transfer, message length setting, calculation result, calculation status and so on.

This driver is contained in \Libraries\TX04_Periph_Driver\src\tmpm46b_sha.c, with \Libraries\TX04_Periph_Driver\inc\tmpm46b_sha.h containing the API definitions for use by applications.

16.2 API Functions

16.2.1 Function List

- Result SHA_SetRunState(SHA_RunCmd ***Cmd***);
- Result SHA_SetCalculationInt(SHA_CalculationInt ***CalculationInt***);
- Result SHA_SetInitMode(SHA_InitMode ***InitMode***);
- Result SHA_SetInitValue(uint32_t ***INIT[8U]***);
- Result SHA_SetDMAState(FunctionalState ***DMATransfer***);
- FunctionalState SHA_GetDMAState(void);
- Result SHA_SetMsgLen(uint32_t ***MSGLEN[2U]***);
- Result SHA_SetRmnMsgLen(uint32_t ***REMAIN[2U]***);
- void SHA_GetRmnMsgLen(uint32_t ***RmnMsgLen[2U]***);
- Result SHA_SetMessage(uint32_t ***MSG[16U]***);
- void SHA_GetResult(uint32_t HashRes[8U]);
- SHA_CalculationStatus SHA_GetCalculationStatus(void);
- void SHA_IPReset(void);

16.2.2 Detailed Description

Functions listed above can be divided into three parts:
1) The SHA basic configuration is handled by the SHA_SetCalculationInt(), SHA_SetInitMode(), SHA_SetInitValue(), SHA_SetDMAState(), SHA_SetMsgLen(), SHA_SetRmnMsgLen() and SHA_SetMessage() functions.
2) The SHA operation result and status are got by the SHA_GetDMAState(), SHA_GetRmnMsgLen (), SHA_GetResult() and SHA_GetCalculationStatus() functions.
3) The SHA start and peripheral function reset is handled by SHA_SetRunState() and SHA_IPReset() functions.

16.2.3 Function Documentation

16.2.3.1 SHA_SetRunState

Start or stop the SHA processor.

Prototype:
Result
SHA_SetRunState(SHA_RunCmd ***Cmd***)

Parameters:
Cmd: The command for the SHA processor.
This parameter can be one of the following values:

- **SHA_START**: Start SHA operation when CPU transfer is used.
- **SHA_STOP**: Stop SHA operation when CPU transfer is used.

Description:
This function will start or stop the SHA processor.

***Note:**
This function setting is ignored when SHADMAEN<DMAEN>=1.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.2 SHA_SetCalculationInt

Set interrupt output after calculation is complete.

Prototype:
Result
SHA_SetCalculationInt(SHA_CalculationInt ***CalculationInt***)

Parameters:
CalculationInt: Interrupt control.
This parameter can be one of the following values:

- **SHA_INT_LAST_CALCULATION**: An interrupt is output only at the last calculation..
- **SHA_INT_EACH_CALCULATION**: Interrupts are output every time calculation is complete when continuous data is handled.

Description:
This function will set interrupt output after calculation is complete.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.3 SHA_SetInitMode

Set the Hash initial value mode.

Prototype:
Result
SHA_SetInitMode(SHA_InitMode ***InitMode***)

Parameters:

InitMode: The Hash initial value mode.
This parameter can be one of the following values:

- **SHA_INIT_VALUE_PREVIOUS**: The Hash value in the previous block is used.
- **SHA_INIT_VALUE_REG**: The Hash value specified with the SHAINITx register.
- **SHA_INIT_VALUE_256_BIT**: A 256-bit Hash value specified with FIPS PUB 180-3 stored in the core internally.
- **SHA_INIT_VALUE_224_BIT**: A 224-bit Hash value specified with FIPS PUB 180-3 stored in the core internally.

Description:
This function will set the Hash initial value mode.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.4 SHA_SetInitValue

Set the Hash initial value register.

Prototype:
Result
SHA_SetInitValue(uint32_t ***INIT[8U]***)

Parameters:
INIT[8U]: An array that contains the Hash initial value.

Description:
This function will set the Hash initial value register.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.5 SHA_SetDMAState

Enable or disable the DMA transfer.

Prototype:
Result
SHA_SetDMAState(FunctionalState ***DMATransfer***)

Parameters:
DMATransfer: Specify the DMA transfer.
This parameter can be one of the following values:

- **ENABLE**: Enable DMA transfer.
- **DISABLE**: Disable DMA transfer.

Description:
This function will enable or disable the DMA transfer.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.6 SHA_GetDMAState

Get the DMA transfer state.

Prototype:
FunctionalState
SHA_GetDMAState(void)

Parameters:
None

Description:
This function will get the DMA transfer state.

Return:
The DMA transfer state:
ENABLE: DMA transfer is being enabled.
DISABLE: DMA transfer is being disabled.

16.2.3.7 SHA_SetMsgLen

Set the whole message length in unit of byte.

Prototype:
Result
SHA_SetMsgLen(uint32_t ***MSGLEN[2U]***)

Parameters:
MSGLEN[2U]: An array that contains the whole message length in unit of byte.

Description:
This function will set the whole message length in unit of byte.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.8 SHA_SetRmnMsgLen

Set the unhandled message length in unit of byte.

Prototype:
Result
SHA_SetRmnMsgLen(uint32_t ***REMAIN[2U]***)

Parameters:
REMAIN[2U]: An array that contains the unhandled message length in unit of byte.

Description:
This function will set the unhandled message length in unit of byte.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.9 SHA_GetRmnMsgLen

Get the unhandled message length in unit of byte.

Prototype:
void
SHA_GetRmnMsgLen(uint32_t ***RmnMsgLen[2U]***)

Parameters:
RmnMsgLen[2U]: An array that contains the unhandled message length in unit of byte.

Description:
This function will get t the unhandled message length in unit of byte.

Return:
None.

16.2.3.10 SHA_SetMessage

Set a 512-bit message.

Prototype:
Result
SHA_SetMessage(uint32_t ***MSG[16U]***)

Parameters:
MSG[16U]: An array that contains a 512-bit message.

Description:
This function will set a 512-bit message.

***Note:**
Data is stored as shown below:

Bit	31 24	23 16	15 8	7 0
Bit 511 - 480	SHAMSG15			
Bit 479 - 448	SHAMSG14			
Bit 447 - 416	SHAMSG13			
Bit 415 - 384	SHAMSG12			
Bit 383 - 352	SHAMSG11			
Bit 351 - 320	SHAMSG10			
Bit 319 - 288	SHAMSG09			
Bit 287 - 256	SHAMSG08			
Bit 255 - 224	SHAMSG07			
Bit 223 - 192	SHAMSG06			
Bit 191 - 160	SHAMSG05			
Bit 159 - 128	SHAMSG04			
Bit 127 - 96	SHAMSG03			
Bit 95 - 64	SHAMSG02			
Bit 63 - 32	SHAMSG01			
Bit 31 - 0	SHAMSG00			

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

16.2.3.11 SHA_GetResult

Get the calculation result.

Prototype:
void
SHA_GetResult(uint32_t ***HashRes[8U]***)

Parameters:
HashRes[8U]: An array that contains the calculation result.

Description:
This function will get the calculation result.

***Note:**
The last calculation result is stored as shown below:

bit	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0
SHA-224		SHARESULT6	SHARESULT5	SHARESULT4	SHARESULT3	SHARESULT2	SHARESULT1	SHARESULT0
SHA-256	SHARESULT7	SHARESULT6	SHARESULT5	SHARESULT4	SHARESULT3	SHARESULT2	SHARESULT1	SHARESULT0

Return:
None.

16.2.3.12 SHA_GetCalculationStatus

Get the calculation status.

Prototype:
SHA_CalculationStatus
SHA_GetCalculationStatus(void)

Parameters:
None

Description:
This function will get the calculation status.

***Note:**
Do not write any value to SHA registers when calculation is in process.

Return:
The calculation status:
SHA_CALCULATION_COMPLETE: Calculation is complete.
SHA_CALCULATION_PROCESS: Calculation is in process.

16.2.3.13 SHA_IPReset

Reset SHA by peripheral function.

Prototype:
void
SHA_IPReset(void)

Parameters:
None

Description:
This function will reset SHA by peripheral function.

Return:
None

16.2.4 Data Structure Description

None

17. SSP

17.1 Overview

TOSHIBA TMPM46B contains SSP (Synchronous Serial Port) module with 3 channels (SSP0, SSP1 and SSP2).

The SSP is an interface that enables serial communications with the peripheral devices with three types of synchronous serial interface functions.

The SSP performs serial-parallel conversion of the data received from a peripheral device. The transmit path buffers data in the independent 16-bit wide and 8-layered transmit FIFO in the transmit mode, and the receive path buffers data in the 16-bit wide and 8-layered receive FIFO in receive mode. Serial data is transmitted via SPDO and received via SPDI. The SSP contains a programmable prescaler to generate the serial output clock SPCLK from the input clock fsys. The operation mode, frame format, and data size of the SSP are programmed in the control registers SSP0CR0 and SSP0CR1.

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_ssp.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_ssp.h containing the macros, data types, structures and API definitions for use by applications.

17.2 API Functions

17.2.1 Function List

- void SSP_Enable(TSB_SSP_TypeDef * ***SSPx***);
- void SSP_Disable(TSB_SSP_TypeDef * ***SSPx***);
- void SSP_Init(TSB_SSP_TypeDef * ***SSPx***, SSP_InitTypeDef * ***InitStruct***);
- void SSP_SetClkPreScale(TSB_SSP_TypeDef * ***SSPx***, uint8_t ***PreScale***, uint8_t ***ClkRate***);
- void SSP_SetFrameFormat(TSB_SSP_TypeDef * ***SSPx***, SSP_FrameFormat ***FrameFormat***);
- void SSP_SetClkPolarity(TSB_SSP_TypeDef * ***SSPx***, SSP_ClkPolarity ***ClkPolarity***);
- void SSP_SetClkPhase(TSB_SSP_TypeDef * ***SSPx***, SSP_ClkPhase ***ClkPhase***);
- void SSP_SetDataSize(TSB_SSP_TypeDef * ***SSPx***, uint8_t ***DataSize***);
- void SSP_SetSlaveOutputCtrl(TSB_SSP_TypeDef * ***SSPx***, FunctionalState ***NewState***);
- void SSP_SetMSMode(TSB_SSP_TypeDef * ***SSPx***, SSP_MS_Mode ***Mode***);
- void SSP_SetLoopBackMode(TSB_SSP_TypeDef * ***SSPx***, FunctionalState ***NewState***);
- void SSP_SetTxData(TSB_SSP_TypeDef * ***SSPx***, uint16_t ***Data***);
- uint16_t SSP_GetRxData(TSB_SSP_TypeDef * ***SSPx***);
- WorkState SSP_GetWorkState(TSB_SSP_TypeDef * ***SSPx***);
- SSP_FIFOState SSP_GetFIFOState(TSB_SSP_TypeDef * ***SSPx***, SSP_Direction ***Direction***);
- void SSP_SetINTConfig(TSB_SSP_TypeDef * ***SSPx***, uint32_t ***IntSrc***);
- SSP_INTState SSP_GetINTConfig(TSB_SSP_TypeDef * ***SSPx***);
- SSP_INTState SSP_GetPreEnableINTState(TSB_SSP_TypeDef * ***SSPx***);
- SSP_INTState SSP_GetPostEnableINTState(TSB_SSP_TypeDef * ***SSPx***);
- void SSP_ClearINTFlag(TSB_SSP_TypeDef * ***SSPx***, uint32_t ***IntSrc***);
- void SSP_SetDMACtrl(TSB_SSP_TypeDef * ***SSPx***, SSP_Direction ***Direction***, FunctionalState ***NewState***);

17.2.2 Detailed Description

Functions listed above can be divided into six parts:

1) Configure the common functions of SSP are handled by SSP_Init(), which will call SSP_SetClkPreScale(), SSP_SetFrameFormat(), SSP_SetClkPolarity(), SSP_SetClkPhase(), SSP_SetDataSize(), SSP_SetMSMode().
2) Data transmit and receive are handled by SSP_SetTxData(), SSP_GetRxData() .
3) SSP interrupt relative function are: SSP_SetINTConfig(), SSP_GetINTConfig(), SSP_GetPreEnableINTState(), SSP_GetPostEnableINTState(), SSP_ClearINTFlag().
4) Get SSP status are handled by SSP_GetWorkState(), SSP_GetFIFOState()
5) Enable/Disable SSP module are handled by SSP_Enable(), SSP_Disable().
6) SSP_SetSlaveOutputCtrl(), SSP_SetLoopBackMode() and SSP_SetDMACtrl() handle other specified functions.

17.2.3 Function Documentation

Note**: in all of the following APIs, parameter "TSB_SSP_TypeDef* ***SSPx" can be one of the following values: **SSP0** ,**SSP1** or **SSP2**

17.2.3.1 SSP_Enable

Enable the specified SSP channel.

Prototype:
void
SSP_Enable(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function is to enable specified SSP channel by ***SSPx***.

Return:
None

17.2.3.2 SSP_Disable

Disable the specified SSP channel.

Prototype:
void
SSP_ Disable(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function is to disable specified SSP channel by ***SSPx***.

Return:
None

17.2.3.3 SSP_Init

Initialize the specified SSP channel through the data in structure SSP_InitTypeDef.

Prototype:

```
void
SSP_Init(TSB_SSP_TypeDef * SSPx,
            SSP_InitTypeDef* InitStruct)
```

Parameters:

SSPx: Select the SSP channel.

InitStruct: It is a structure with detail as below:

```
typedef struct {
    SSP_FrameFormat FrameFormat;
    uint8_t PreScale;
    uint8_t ClkRate;
    SSP_ClkPolarity ClkPolarity;
    SSP_ClkPhase ClkPhase;
    uint8_t DataSize;
    SSP_MS_Mode Mode;
} SSP_InitTypeDef;
```

For detail of this structure, refer to part “Data Structure Description”.

Description:

This function will configure the SSP channel by ***SSPx*** and SSP_InitTypeDef ***InitStruct***.

It will call the functions below:

SSP_SetFrameFormat(),
SSP_SetClkPreScale(),
SSP_SetClkPolarity(),
SSP_SetClkPhase(),
SSP_SetDataSize(),
SSP_SetMSMode().

Return:

None

17.2.3.4 SSP_SetClkPreScale

Set the bit rate for transmit and receive for the specified SSP channel.

Prototype:

```
void
SSP_SetClkPreScale(TSB_SSP_TypeDef * SSPx,
                   uint8_t PreScale,
                   uint8_t ClkRate)
```

Parameters:

SSPx: Select the SSP channel

PreScale: Clock prescale divider, must be even number from 2 to 254.

ClkRate: Serial clock rate (from 0 to 255).

Description:

This function is to set the SSP channel by ***SSPx***, the bit rate for transmit and receive by ***PreScale*** & ***ClkRate***, generally it is called by SSP_Init().

This bit rate for Tx and Rx is obtained by the following equation:

BitRate = **fsys** / (**PreScale** × (1 + **ClkRate**))

where **fsys** is the frequency of system.

Return:
None

17.2.3.5 SSP_SetFrameFormat

Specify the Frame Format of specified SSP channel.

Prototype:
void
SSP_SetFrameFormat(TSB_SSP_TypeDef * ***SSPx***,
SSP_FrameFormat ***FrameFormat***)

Parameters:
SSPx: Select the SSP channel.
FrameFormat: Frame format of SSP which can be:

- **SSP_FORMAT_SPI:** configure SSP module to SPI mode.
- **SSP_FORMAT_SSI:** configure SSP module to SSI mode.
- **SSP_FORMAT_MICROWIRE:** configure SSP module to Microwire mode.

Description:
This function is to set the SSP channel by ***SSPx***, specify the Frame Format of SSP by ***FrameFormat***, generally it is called by **SSP_Init()**.

Return:
None

17.2.3.6 SSP_SetClkPolarity

When specified SSP channel is configured as SPI mode, specify the clock polarity in its idle state.

Prototype:
void
SSP_SetClkPolarity(TSB_SSP_TypeDef * ***SSPx***,
SSP_ClkPolarity ***ClkPolarity***)

Parameters:
SSPx: Select the SSP channel.
ClkPolarity: SPI clock polarity
This parameter can be one of the following values:

- **SSP_POLARITY_LOW:** SCLK pin is low level in idle state.
- **SSP_POLARITY_HIGH:** SCLK pin is high level in idle state.

Description:
This function is to set the SSP channel by ***SSPx***, specify the clock polarity by ***ClkPolarity*** in idle state of SCLK pin when the Frame Format is set as SPI, generally it is called by **SSP_Init()**.

Return:
None

17.2.3.7 SSP_SetClkPhase

When specified SSP channel is configured as SPI mode, specify its clock phase.

Prototype:
void
SSP_SetClkPhase(TSB_SSP_TypeDef * ***SSPx***,
SSP_ClkPhase ***ClkPhase***)

Parameters:
SSPx: Select the SSP channel.
ClkPhase: SPI clock phase
This parameter can be one of the following values:

- **SSP_PHASE_FIRST_EDGE:** capture data in first edge of SCLK pin.
- **SSP_PHASE_SECOND_EDGE:** capture data in second **edge** of SCLK pin.

Description:
This function is to set the SSP channel by ***SSPx***, specify the clock phase by ***ClkPhase*** when the Frame Format is set as SPI, generally it is called by **SSP_Init()**.

Return:
None

17.2.3.8 SSP_SetDataSize

Set the Rx/Tx data size for the specified SSP channel.

Prototype:
Void
SSP_SetDataSize(TSB_SSP_TypeDef * ***SSPx***,
uint8_t ***DataSize***)

Parameters:
SSPx: Select the SSP channel.
DataSize: Data size select from 4 to 16.

Description:
This function is to set the SSP channel by ***SSPx***, set the Rx/Tx Data Size by ***DataSize***, generally it is called by **SSP_Init()**.

Return:
None

17.2.3.9 SSP_SetSlaveOutputCtrl

Enable/Disable slave mode output for the specified SSP channel.

Prototype:

void
SSP_SetSlaveOutputCtrl(TSB_SSP_TypeDef * ***SSPx***,
FunctionalState ***NewState***)

Parameters:
SSPx: Select the SSP channel.
NewState: Specifies the state of the SPDO output when SSP is set in slave mode, This parameter can be one of the following values:

- **ENABLE:** enable the SPDO output.
- **DISABLE:** disable the SPDO output.

Description:
This function is to set the SSP channel by ***SSPx***, Enable/Disable slave mode SPDO output by ***NewState***.

Return:
None

17.2.3.10 SSP_SetMSMode

Set the SSP Master or Slave mode for the specified SSP channel.

Prototype:
void
SSP_SetMSMode(TSB_SSP_TypeDef * ***SSPx***,
SSP_MS_Mode ***Mode***)

Parameters:
SSPx: Select the SSP channel.
Mode: Select the SSP mode
This parameter can be one of the following values:

- **SSP_MASTER:** SSP run in master mode.
- **SSP_SLAVE:** SSP run in slave mode.

Description:
This function is to set the SSP channel by ***SSPx***, select the SSP run in Master mode or Slave mode by ***Mode***.

Return:
None

17.2.3.11 SSP_SetLoopBackMode

Set loop back mode of SSP for the specified SSP channel.

Prototype:
void
SSP_SetLoopBackMode(TSB_SSP_TypeDef * ***SSPx***,
FunctionalState ***NewState***)

Parameters:
SSPx: Select the SSP channel.
NewState: Specifies the state for self-loop back of SSP.
This parameter can be one of the following values:

- **ENABLE**: enable the self-loop back mode.

- **DISABLE**: disable the self-loop back mode.

Description:
This function is to set the SSP channel by ***SSPx***, the loop back mode of SSP by ***NewState***.
For example, loop back mode can be enabled to do self testing between transmit and receive.

Return:
None

17.2.3.12 SSP_SetTxData

Set the data to be sent into Tx FIFO of the specified SSP channel.

Prototype:
void
SSP_SetTxData(TSB_SSP_TypeDef * ***SSPx***,
uint16_t ***Data***)

Parameters:
SSPx: Select the SSP channel.
Data: 4~16bit data to be send

Description:
This function will set the data by ***Data*** and start to send it into Tx FIFO of the specified SSP channel by ***SSPx***.

Return:
None

17.2.3.13 SSP_GetRxData

Read the data received from Rx FIFO of the specified SSP channel.

Prototype:
uint16_t
SSP_GetRxData(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function will read received data from Rx FIFO of the specified SSP channel by ***SSPx***.

Return:
Data with uint16_t type

17.2.3.14 SSP_GetWorkState

Get the Busy or Idle state of the specified SSP channel.

Prototype:

WorkState
SSP_GetWorkState(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function will get the Busy/Idle state of the specified SSP channel by ***SSPx***.

Return:
WorkState type, the value means:
BUSY: SSP module is busy.
DONE: SSP module is idle.

17.2.3.15 SSP_GetFIFOState

Get the Rx/Tx FIFO state of the specified SSP channel.

Prototype:
SSP_FIFOState
SSP_GetFIFOState(TSB_SSP_TypeDef * ***SSPx***
SSP_Direction ***Direction***)

Parameters:
SSPx: Select the SSP channel.
Direction: The direction which means transmit or receive
This parameter can be one of the following values:

- **SSP_RX**: target is to check state of receive FIFO.
- **SSP_TX**: target is to check state of transmit FIFO.

Description:
This function will the specified SSP channel by ***SSPx,*** get the Rx/Tx FIFO state by ***Direction***.
For example, data can be sent after judging Tx FIFO is available by the code below:

```
                SSP_FIFOState  fifoState;

                fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_TX);
                if ((fifoState == SSP_FIFO_EMPTY) || (fifoState ==
SSP_FIFO_NORMAL))
                   { SSP_SetTxData(SSP0, data_to_be_sent ); }
```

Return:
The state of SSP FIFO, which can be
SSP_FIFO_EMPTY: FIFO is empty.
SSP_FIFO_NORMAL: FIFO is not full and not empty.
SSP_FIFO_INVALID: FIFO is invalid state.
SSP_FIFO_FULL: FIFO is full

17.2.3.16 SSP_SetINTConfig

Enable/Disable interrupt source of the specified SSP channel.

Prototype:
void

SSP_SetINTConfig(TSB_SSP_TypeDef * ***SSPx***,
uint32_t ***IntSrc***)

Parameters:
SSPx: Select the SSP channel.

IntSrc: The interrupt source for SSP to be enabled or disabled.
To disable all interrupt sources, use the parameter:

- **SSP_INTCFG_NONE**

To enable the interrupt one by one, use the logical operator " **|** " with below parameter:

- **SSP_INTCFG_RX_OVERRUN:** Receive overrun interrupt.
- **SSP_INTCFG_RX_TIMEOUT:** Receive timeout interrupt.
- **SSP_INTCFG_RX:** Receive FIFO interrupt (at least half full).
- **SSP_INTCFG_TX:** Transmit FIFO interrupt (at least half empty).

To enable all the 4 interrupt above together, use the parameter:

- **SSP_INTCFG_ALL**

Description:
This function will specified SSP channel by ***SSPx***, enable/disable interrupts by ***IntSrc***.
For example, we can enable Tx and Rx interrupt by code like below:
SSP_SetINTConfig(SSP0, SSP_INTCFG_RX | SSP_INTCFG_TX)

Return:
None

17.2.3.17 SSP_GetINTConfig

Get the Enable/Disable setting for each Interrupt source in the specified SSP channel.

Prototype:
SSP_INTState
SSP_GetINTConfig(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function will get the masked interrupt status of the specified SSP channel by ***SSPx***.
For example, it can be used to check which interrupt source is enabled or disabled by SSP_SetINTConfig().

Return:
SSP_INTState type. It contains the state of SSP interrupt setting, for more detail refer to the description for union SSP_INTState in "Data Structure Description" part.

17.2.3.18 SSP_GetPreEnableINTState

Get the raw status of each interrupt source in the specified SSP channel.

Prototype:
SSP_INTState
SSP_GetPreEnableINTState(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function will get the pre-enable interrupt status of the specified SSP channel by ***SSPx***.

Return:
SSP_INTState type. It contains the pre-enable interrupt status (raw status before masked) , for more detail refer to the description for union SSP_INTState in "Data Structure Description" part.

17.2.3.19 SSP_GetPostEnableINTState

Get the specified SSP channel post-enable interrupt status. (after masked)

Prototype:
SSP_INTState
SSP_GetPostEnableINTState(TSB_SSP_TypeDef * ***SSPx***)

Parameters:
SSPx: Select the SSP channel.

Description:
This function will get post-enable interrupt status of the specified SSP channel by ***SSPx***.

Return:
SSP_INTState type. It contains the post-enable interrupt status (after masked) , for more detail refer to the description for union SSP_INTState in "Data Structure Description" part.

17.2.3.20 SSP_ClearINTFlag

Clear interrupt flag of specified SSP channel by writing '1' to correspond bit.

Prototype:
void
SSP_ClearINTFlag(TSB_SSP_TypeDef * ***SSPx***,
uint32_t ***IntSrc***)

Parameters:
SSPx: Select the SSP channel.
IntSrc: The interrupt source to be cleared.
This parameter can be one of the following values:

- **SSP_INTCFG_RX_OVERRUN:** Receive overrun interrupt.
- **SSP_INTCFG_RX_TIMEOUT:** Receive timeout interrupt.
- **SSP_INTCFG_ALL:** all the 2 interrupt above together

Description:

This function will clear interrupt flag by ***IntSrc*** of the specified SSP channel by ***SSPx***.

Return:
None

17.2.3.21 SSP_SetDMACtrl

Enable/Disable the DMA FIFO for Rx/Tx of specified SSP channel.

Prototype:
void
SSP_SetDMACtrl(TSB_SSP_TypeDef * ***SSPx***,
SSP_Direction ***Direction***,
FunctionalState ***NewState***)

Parameters:
SSPx: Select the SSP channel.
Direction: The direction which means transmit or receive.
This parameter can be one of the following values:

- **SSP_RX**: target is to set receive DMA FIFO.
- **SSP_TX**: target is to set transmit DMA FIFO.

NewState: New state of DMA FIFO mode.
This parameter can be one of the following values:

- **ENABLE**: enables the DMA for FIFO.
- **DISABLE**: disables the DMA for FIFO.

Description:
This function will enable/disable the DMA FIFO Rx/Tx of the specified SSP channel by ***SSPx***.

Return:
None

17.2.4 Data Structure Description

17.2.4.1 SSP_InitTypeDef

Data Fields for this structure:
SSP_FrameFormat
FrameFormat Set frame format of SSP.
Which can be:

- **SSP_FORMAT_SPI**: configure the SSP in SPI mode.
- **SSP_FORMAT_SSI**: configure the SSP in SSI mode.
- **SSP_FORMAT_MICROWIRE**: configure the SSP in Microwire mode

uint8_t
PreScale Clock prescale divider, must be even number from 2 to 254.

uint8_t
ClkRate Serial clock rate, from 0 to 255.

SSP_ClkPolarity

ClkPolarity SPI clock polarity, Specify the clock polarity in idle state of SCLK pin when the Frame Format is set as SPI.
Which can be:

- **SSP_POLARITY_LOW**: SCLK pin is low level in idle state.
- **SSP_POLARITY_HIGH**: SCLK pin is high level in idle state.

SSP_ClkPhase
ClkPhase Specify the clock phase when the Frame Format is set as SPI.
Which can be:

- **SSP_PHASE_FIRST_EDGE**: capture data in first edge of SCLK pin.
- **SSP_PHASE_SECOND_EDGE**: capture data in second edge of SCLK pin.

uint8_t
DataSize Select data size From 4 to 16

SSP_MS_Mode
Mode SSP device mode.
Which can be:

- **SSP_MASTER**: SSP module is run in master mode.
- **SSP_SLAVE**: SSP module is run in slave mode.

17.2.4.2 SSP_INTState

Data Fields for this union:
uint32_t
All: SSP interrupt factor.
Bit
uint32_t
OverRun: 1 Receive Overrun.
uint32_t
TimeOut: 1 Receive Timeout.
uint32_t
Rx: 1 Receive.
uint32_t
Tx: 1 Transmit.
uint32_t
Reserved: 28 Reserved.

18. TMRB

18.1 Overview

TOSHIBA TMPM46B contains 8channels of multi-functional 16-bit timer/event counter (TMRB0 through TMRB7). Each channel can operate in the following modes:

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

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

- Frequency measurement
- Pulse width measurement
- Time difference measurement

TMPM46B also has 16-bit multi-purpose timer (MPT), when being operated in timer mode, they are the same as common timer channels.

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

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_tmrb.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_tmrb.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***)
- void TMRB_SetINTMask(TSB_TB_TypeDef * ***TBx***, uint32_t ***INTMask***)
- void TMRB_ChangeLeadingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***LeadingTiming***)
- void TMRB_ChangeTrailingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***TrailingTiming***)
- uint16_t TMRB_GetUpCntValue(TSB_TB_TypeDef * ***TBx***)
- uint16_t TMRB_GetCaptureValue(TSB_TB_TypeDef * ***TBx***, uint8_t ***CapReg***)
- void TMRB_ExecuteSWCapture(TSB_TB_TypeDef * ***TBx***)
- void TMRB_SetIdleMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***)
- void TMRB_SetSyncMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***)
- void TMRB_SetDoubleBuf(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***,

uint8_t ***WriteRegMode***)

- void TMRB_SetExtStartTrg(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***TrgMode***)
- void TMRB_SetClkInCoreHalt(TSB_TB_TypeDef * ***TBx***, uint8_t ***ClkState***)

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_GetUpCntValue() and TMRB_GetCaptureValue().
4) TMRB_SetFlipFlop(), TMRB_SetINTMask(), TMRB_SetIdleMode(), TMRB_SetSyncMode(), TMRB_SetDoubleBuf(), TMRB_SetExtStartTrg() and TMRB_SetClkInCoreHalt ()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_TB_MPT0, TSB_TB_MPT1, TSB_TB_MPT2, TSB_TB_MPT3.

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***.
If channel is MPT, this function will also select MPT channel as timer mode.

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***.
If channel is MPT, this function will also select MPT channel as timer mode.

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 and up-counter clearing timing.

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

Parameters:
TBx is the specified TMRB channel.

CaptureTiming specifies TMRB capture timing, which can be
When TBx = TSB_TB_MPT0 to TSB_TB_MPT3:

- **MPT_DISABLE_CAPTURE**: Capture is disabled.
- **MPT_CAPTURE_IN_RISING**: At the rising edge of MTxTBIN input, counter values are captured to the capture register 0 (MTxCP0)
- **MPT_CAPTURE_IN_RISING_FALLING**: At the rising edge of MTxTBIN input, counter values are captured to the capture register 0 (MTxCP0).At the falling edge of MTxTBIN input, counter values are captured to the capture register 1 (MTxCP1).

When TBx = TSB_TB0 to TSB_TB7:

- **TMRB_DISABLE_CAPTURE**: Capture is disabled.
- **TMRB_CAPTURE_TBIN0_TBIN1_RISING**: Captures a counter value on rising edge of TBxIN0 input into Capture register 0 (TBxCP0).Captures a counter value on rising edge of TBxIN1 input into Capture register 1 (TBxCP1). (**Only TSB_TB4 to TSB_TB7 can choose TMRB_CAPTURE_TBIN0_TBIN1_RISING**)
- **TMRB_CAPTURE_TBIN0_RISING_FALLING**: Captures a counter value on rising edge of TBxIN0 input into Capture register 0 (TBxCP0).Captures a counter value on falling edge of TBxIN0 input into Capture register 1 (TBxCP1).
- **TMRB_CAPTURE_TBFF0_EDGE**: Captures a counter value on rising edge of TBxFF0 input into Capture register 0 (TBxCP0).Captures a counter value on falling edge of TBxFF0 input into Capture register 1 (TBxCP1).
- **TMRB_CLEAR_TBIN1_RISING**: Clears up-counter on rising edge of TBxIN1 input.(**Only TSB_TB4 to TSB_TB7 can choose TMRB_CLEAR_TBIN1_RISING**)
- **TMRB_CAPTURE_TBIN0_RISING_CLEAR_TBIN1_RISING:** Captures a counter value on rising edge of TBxIN0 input into Capture register 0(TBxCP0); clears up-counter on rising edge of TBxIN1 input. If capture timing and up-counter clearing timing are same, capturing is performed first, and then up-counter is cleared. (**Only TSB_TB4 to TSB_TB7 can choose TMRB_CAPTURE_TBIN0_RISING_CLEAR_TBIN1_RISING**)

Description:
This function will configure the capture timing and up-counter clearing timing.

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_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_TRAILING_INT**: Mask the interrupt the factor of which is that the value in up-counter and trailingtiming are match.
- **TMRB_MASK_MATCH_LEADING_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.
- **TMRB_MASK_MATCH_LEADING_INT | TMRB_MASK_MATCH_TRAILING_INT**: Mask the interrupt the factor of which is that the value in up-counter and trailingtiming are match or mask the interrupt the factor of which is that the value in up-counter and leadingtiming are match.
- **TMRB_MASK_MATCH_LEADING_INT | TMRB_MASK_OVERFLOW_INT**: Mask the interrupt the factor of which is that the value in up-counter and leadingtiming are match or mask the interrupt the factor of which is the occurrence of overflow.
- **TMRB_MASK_MATCH_TRAILING_INT | TMRB_MASK_OVERFLOW_INT**: Mask the interrupt the factor of which is that the value in up-counter and trailingtiming are match or mask the interrupt the factor of which is the occurrence of overflow.
- **TMRB_MASK_MATCH_LEADING_INT | TMRB_MASK_MATCH_TRAILING_INT | TMRB_MASK_OVERFLOW_INT**: Mask the interrupt the factor of which is that the value in up-counter and trailingtiming are match or mask the interrupt the factor of which is that the value in up-counter and leadingtiming are match or mask the interrupt the factor of which is the occurrence of overflow

Description:

If **TMRB_MASK_MATCH_TRAILING_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_LEADING_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.
If the combination of **TMRB_MASK_MATCH_TRAILING_INT** and **TMRB_MASK_MATCH_LEADING_INT** and **TMRB_MASK_OVERFLOW_INT** is selected, the interrupt of the specified TMRB channel will not happen even if the relevant situation happened.

Return:

None

18.2.3.9 TMRB_ChangeLeadingTiming

Change the value of leadingtiming for the specified channel.

Prototype:

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

Parameters:

TBx is the specified TMRB channel.
LeadingTiming specifies the value of leadingtiming, max. is 0xFFFF.

Description:

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

Return:

None

***Note:**

LeadingTiming can not exceed ***TrailingTiming***.

18.2.3.10 TMRB_ChangeTrailingTiming

Change the value of trailingtiming for the specified channel.

Prototype:

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

Parameters:

TBx is the specified TMRB channel.
TrailingTiming specifies the value of trailingtiming, max. is 0xFFFF.

Description:

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

Return:

None

***Note:**

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

18.2.3.11 TMRB_GetUpCntValue

Get up-counter value of the specified TMRB channel.

Prototype:

uint16_t

TMRB_GetUpCntValue(TSB_TB_TypeDef* ***TBx***)

Parameters:
TBx is the specified TMRB channel.

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

Return:
The value of up-counter

18.2.3.12 TMRB_GetCaptureValue

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

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

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

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

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

Return:
The captured value

18.2.3.13 TMRB_ExecuteSWCapture

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

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

Parameters:
TBx is the specified TMRB channel.

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

Return:
None

18.2.3.14 TMRB_SetIdleMode

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

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

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

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

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

Return:
None

18.2.3.15 TMRB_SetSyncMode

Enable or disable the synchronous mode of specified TMRB channel.

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

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

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

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

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

Return:
None

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

18.2.3.16 TMRB_SetDoubleBuf

Enable or disable double buffering for the specified TMRB channel and set the timing to write to timer register 0 and 1 when double buffer enabled.

Prototype:

void
TMRB_SetDoubleBuf(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***,
uint8_t ***WriteRegMode***)

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.

WriteRegMode specifies timing to write to timer register 0 and 1 when double buffer enabled, which can be

- **TMRB_WRITE_REG_SEPARATE:** Timer register 0 and 1 can be written separately, even in case writing preparation is ready for only one register.
- **TMRB_WRITE_REG_SIMULTANEOUS:** In case both registers are not ready to be written, timer registers 0 and 1 can't be written.

Description:

The register TBxRG0 (***LeadingTiming***) and TBxRG1 (***TrailingTiming***) and their buffers are assigned to the same address. If double buffering is disabled, the same value is written to the registers and their buffers.
If double buffering is enabled, the value is only written to each register buffer. Therefore, to write an initial value to the registers, TBxRG0 (***LeadingTiming***) and TBxRG1 (***TrailingTiming***), the double buffering must be set to **DISABLE**. Then **ENABLE** double buffering and write the following data to the register, which can be loaded when the corresponding interrupt occurs automatically.

Return:

None

***Note:**

WriteRegMode is invalid for TMRB0~TMRB7. So when this API is used for these channels, 0 is recommended for ***WriteRegMode.***

18.2.3.17 TMRB_SetExtStartTrg

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

Prototype:

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

Parameters:

TBx is the specified TMRB channel.

NewState specifies the state external trigger, which can be

- **ENABLE:** use external trigger signal,
- **DISABLE:** use software start.

TrgMode specifies active edge of the external trigger signal. which can be

- **TMRB_TRG_EDGE_RISING:** Select rising edge of external trigger.

- **TMRB_TRG_EDGE_FALLING:** Select falling edge of external trigger.

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

Return:
None

18.2.3.18 TMRB_SetClkInCoreHalt

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

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

Parameters:
TBx is the specified TMRB channel.
ClkState specifies timer state in HALT mode, which can be

- **TMRB_RUNNING_IN_CORE_HALT:** clock not stops in Core HALT
- **TMRB_STOP_IN_CORE_HALT:** clock stops in Core HALT.

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

Return:
None

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(TMRB0~TMRB7 only).
- **TMRB_CLK_DIV_128**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 128(TMRB0~TMRB7 only).

- **TMRB_CLK_DIV_256**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 256(TMRB0~TMRB7 only).
- **TMRB_CLK_DIV_512**, which means that the frequency of source clock for internal interval timer is quotient of fperiph divided by 512(TMRB0~TMRB7 only).

uint32_t
TrailingTiming specifies the trailingtiming 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.(TMRB0~TMRB7 only)
- **TMRB_AUTO_CLEAR**, which means that the up-counter will restart counting from 0 immediately when the value in up-counter matches **TrailingTiming**. (TMRB0~TMRB7 only)
- **MPT_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. (MPT0~MPT3 only)
- **MPT_AUTO_CLEAR**, which means that the up-counter will restart counting from 0 immediately when the value in up-counter matches **TrailingTiming**. (MPT0~MPT3 only)

uint32_t
LeadingTiming specifies the leadingtiming value to be written into TBnRG0, max. 0xFFFF, and it cannot 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_TRAILING**, which means that the reversing flip-flop output will be triggered when the up-counter matches the trailingtiming,
- **TMRB_FLIPFLOP_MATCH_LEADING**, 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
  Reserverd : 29 -

19. SIO/UART

19.1 Overview

TMPM46B has four serial I/O channels. Each channel can operate in both UART mode (asynchronous communication) and I/O Interface mode (synchronous communication), which can be 7-bit length, 8-bit length and 9-bit length.
In 9-bit UART mode, a wakeup function can be used when the master controller can start up slave controllers via the serial link (multi-controller system).

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

All driver APIs are contained in /Libraries/TX04_Periph_Driver/src/tmpm46b_uart.c, with /Libraries/TX04_Periph_Driver/inc/tmpm46b_uart.h containing the macros, data types, structures and API definitions for use by applications.

19.2 API Functions

19.2.1 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_SetInputClock(TSB_SC_TypeDef * ***UARTx***, uint32_t ***clock***)
- void SIO_SetInputClock(TSB_SC_TypeDef * ***SIOx***, uint32_t ***Clock***)
- void SIO_Enable(TSB_SC_TypeDef* ***SIOx***)

- void SIO_Disable(TSB_SC_TypeDef* ***SIOx***)
- void SIO_Init(TSB_SC_TypeDef* ***SIOx***,
  uint32_t ***IOClkSel***,
  UART_InitTypeDef* ***InitStruct***)
- uint8_t SIO_GetRxData(TSB_SC_TypeDef* ***SIOx***)
- void SIO_SetTxData(TSB_SC_TypeDef* ***SIOx***, uint8_t ***Data***)

19.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Initialize and configure the common functions of each UART channel are handled by UART_Enable(), UART_Disable(),UART_SetInputClock(),UART_Init() and UART_DefaultConfig(),SIO_Enable(), SIO_Disable(),SIO_SetInputClock(),SIO_Init().
2) Transfer control and error check of each UART channel are handled by UART_GetBufState(), UART_GetRxData(), UART_SetTxData() and UART_GetErrState(),SIO_GetRxData(), SIO_SetTxData().
3) UART_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(),

19.2.3 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.
parameter “TSB_SC_TypeDef* ***SIOx***” can be one of the following values:
SIO0, SIO1, SIO2, SIO3.

19.2.3.1 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

19.2.3.2 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

19.2.3.3 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.

19.2.3.4 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

19.2.3.5 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

19.2.3.6 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

19.2.3.7 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

19.2.3.8 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

19.2.3.9 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.

19.2.3.10 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

19.2.3.11 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

19.2.3.12 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

19.2.3.13 UART_FIFOConfig

Enable or disable the FIFO of specified UART channel.

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

Parameters:
UARTx is the specified UART channel.
NewState is the new state of the UART FIFO.
This parameter can be one of the following values:
ENABLE or DISABLE

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

Return:
None

19.2.3.14 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

19.2.3.15 UART_TRxAutoDisable

Controls automatic disabling of transmission and reception.

Prototype:
void
UART_TRxAutoDisable (TSB_SC_TypeDef* ***UARTx***,
UART_TRxAutoDisable ***TRxAutoDisable***);

Parameters:
UARTx is the specified UART channel.
TRxAutoDisable Disabling transmission and reception or not
This parameter can be one of the following values:
UART_RXTXCNT_NONE or UART_RXTXCNT_AUTODISABLE .

Description:
Controls automatic disabling of transmission and reception.

Return:
None

19.2.3.16 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

19.2.3.17 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

19.2.3.18 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

19.2.3.19 UART_RxFIFOFillLevel

Receive FIFO fill level to generate receive interrupts.

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

Parameters:
UARTx is the specified UART channel.
RxFIFOLevel is receive FIFO fill level.
This parameter can be one of the following values:

UART_RXFIFO4B_FLEVLE_4_2B, UART_RXFIFO4B_FLEVLE_1_1B, UART_RXFIFO4B_FLEVLE_2_2B or UART_RXFIFO4B_FLEVLE_3_1B.

Description:
Receive FIFO fill level to generate receive interrupts.

Return:
None

19.2.3.20 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

19.2.3.21 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

19.2.3.22 UART_TxFIFOFillLevel

Transmit FIFO fill level to generate transmit interrupts.

Prototype:
void
UART_TxFIFOFillLevel (TSB_SC_TypeDef* ***UARTx***,

uint32_t ***TxFIFOLevel***);

Parameters:
UARTx is the specified UART channel.
TxFIFOLevel is transmit FIFO fill level.
This parameter can be one of the following values:
UART_TXFIFO4B_FLEVLE_0_0B, UART_TXFIFO4B_FLEVLE_1_1B, UART_TXFIFO4B_FLEVLE_2_0B or UART_TXFIFO4B_FLEVLE_3_1B.

Description:
Transmit FIFO fill level to generate transmit interrupts.

Return:
None

19.2.3.23 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

19.2.3.24 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

19.2.3.25 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

19.2.3.26 UART_GetRxFIFOFillLevelStatus

Status of receive FIFO fill level.

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

Parameters:
UARTx is the specified UART channel.
Description:
Status of receive FIFO fill level.

Return:
UART_TRXFIFO_EMPTY: TX FIFO fill level is empty.
UART_TRXFIFO_1B: TX FIFO fill level is 1 byte.
UART_TRXFIFO_2B: TX FIFO fill level is 2 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 3 bytes.
UART_TRXFIFO_4B: TX FIFO fill level is 4 bytes.

19.2.3.27 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.

19.2.3.28 UART_GetTxFIFOFillLevelStatus

Status of transmit FIFO fill level.

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

Parameters:
UARTx is the specified UART channel.

Description:
Status of transmit FIFO fill level.

Return:
UART_TRXFIFO_EMPTY: TX FIFO fill level is empty.
UART_TRXFIFO_1B: TX FIFO fill level is 1 byte.
UART_TRXFIFO_2B: TX FIFO fill level is 2 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 3 bytes.
UART_TRXFIFO_4B: TX FIFO fill level is 4 bytes.

19.2.3.29 UART_GetTxFIFOUnderRunStatus

Transmit FIFO under run

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

Parameters:
UARTx is the specified UART channel.

Description:
Transmit FIFO under run

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

19.2.3.30 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

19.2.3.31 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

19.2.3.32 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

19.2.3.33 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

19.2.3.34 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

19.2.3.35 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.

IOClkSel is the selected clock.
This parameter can be one of the following values:
SIO_CLK_SCLKOUTPUT or SIO_CLK_SCLKINPUT.

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

19.2.4 Data Structure Description

19.2.4.1 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

19.2.4.2 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.

20. uDMAC

20.1 Overview

TMPM46B incorporates 3 units of built-in DMA controller.
The main functions for one unit are shown below:

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

The uDMAC API provides a set of functions for using the TMPM46B uDMAC modules. It includes uDMAC transfer type set, channel set, mask set, primary/alternative data area set, channel priority, initialize data filling and so on.

This driver is contained in TX04_Periph_Driver\src\tmpm46b_udmac.c, with TX04_Periph_Driver\inc\tmpm46b_udmac.h containing the API definitions for use by applications.

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

20.2 API Functions

20.2.1 Function List

- FunctionalState DMAC_GetDMACState(TSB_DMA_TypeDef * ***DMACx***)
- void DMAC_Enable(TSB_DMA_TypeDef * ***DMACx***)
- void DMAC_Disable(TSB_DMA_TypeDef * ***DMACx***)
- void DMAC_SetPrimaryBaseAddr(TSB_DMA_TypeDef * ***DMACx***, uint32_t ***Addr***)
- uint32_t DMAC_GetBaseAddr(TSB_DMA_TypeDef * ***DMACx***,
  DMAC_PrimaryAlt ***PriAlt***)
- void DMAC_SetSWReq(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel***)

- void DMACA_SetTransferType(DMACA_Channel ***Channel***,
  DMAC_TransferType ***Type***)
- DMAC_TransferType DMACA_GetTransferType(DMACA_Channel ***Channel***)
- void DMACB_SetTransferType(DMACB_Channel ***Channel***,
  DMAC_TransferType ***Type***)
- DMAC_TransferType DMACB_GetTransferType(DMACB_Channel ***Channel***)
- void DMAC_SetMask(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel*** ,
  FunctionalState ***NewState***)
- FunctionalState DMAC_GetMask(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel***)
- void DMAC_SetChannel(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel*** ,
  FunctionalState ***NewState***)
- FunctionalState DMAC_GetChannelState(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel***)
- void DMAC_SetPrimaryAlt(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel***
  DMAC_PrimaryAlt ***PriAlt***)
- DMAC_PrimaryAlt DMAC_GetPrimaryAlt(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel***)
- void DMAC_SetChannelPriority(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel*** ,
  DMAC_Priority ***Priority***)
- DMAC_Priority DMAC_GetChannelPriority(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel***)
- void DMAC_ClearBusErr(TSB_DMA_TypeDef * ***DMACx***)
- Result DMAC_GetBusErrState(TSB_DMA_TypeDef * ***DMACx***)
- void DMAC_FillInitData(TSB_DMA_TypeDef * ***DMACx*** ,
  uint8_t ***Channel*** ,
  DMAC_InitTypeDef * ***InitStruct***)
- DMACA_Flag DMACA_GetINTFlag(void)
- DMACB_Flag DMACB_GetINTFlag(void)
- DMACC_Flag DMACC_GetINTFlag(void)

20.2.2 Detailed Description

Functions listed above can be divided into six parts:

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

20.2.3 Function Documentation

NOTE: For the parameter **‘DMACx’** and **‘Channel’** of all functions, if there isn’t special explanation, the sentence **‘DMACx:** Select DMAC unit.**’** and **‘Channel**: Select channel**’’**will follow the content below:

DMACx: Select DMAC unit.
This parameter can be one of the following values:

- **DMAC_UNIT_A:** DMAC unit A
- **DMAC_UNIT_B:** DMAC unit B
- **DMAC_UNIT_C:** DMAC unit C

Channel: Select channel.
The parameter can be one of the following values:
For DMAC_UNIT_A:

- **DMACA_SNFC_PRD11 :** DMA UNITA request pin SNFC_PRD11
- **DMACA_SNFC_PRD12 :** DMA UNITA request pin SNFC_PRD12
- **DMACA_SNFC_PRD21 :** DMA UNITA request pin SNFC_PRD21
- **DMACA_SMFC_PRD22 :** DMA UNITA request pin SNFC_PRD22
- **DMACA_ADC_COMPLETION :** ADC conversion completion
- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission
- **DMACA_UART1_RX :** UART1 reception
- **DMACA_UART1_TX :** UART1 transmission
- **DMACA_SIO0_UART0_RX :** SIO/UART0 reception
- **DMACA_SIO0_UART0_TX :** SIO/UART0 transmission
- **DMACA_SIO1_UART1_RX :** SIO/UART1 reception
- **DMACA_SIO1_UART1_TX :** SIO/UART1 transmission
- **DMACA_SIO2_UART2_RX :** SIO/UART2 reception
- **DMACA_SIO2_UART2_TX :** SIO/UART2 transmission
- **DMACA_SIO3_UART3_RX :** SIO/UART3 reception
- **DMACA_SIO3_UART3_TX :** SIO/UART3 transmission
- **DMACA_TMRB0_CMP_MATCH :** TMRB0 compare match
- **DMACA_TMRB1_CMP_MATCH :** TMRB1 compare match
- **DMACA_TMRB2_CMP_MATCH :** TMRB2 compare match
- **DMACA_TMRB3_CMP_MATCH :** TMRB3 compare match
- **DMACA_TMRB4_CMP_MATCH :** TMRB4 compare match
- **DMACA_TMRB5_CMP_MATCH :** TMRB5 compare match
- **DMACA_TMRB6_CMP_MATCH :** TMRB6 compare match
- **DMACA_TMRB7_CMP_MATCH :** TMRB7 compare match
- **DMACA_TMRB0_INPUT_CAP0 :** TMRB0 input capture 0
- **DMACA_TMRB0_INPUT_CAP1 :** TMRB0 input capture 1
- **DMACA_TMRB1_INPUT_CAP0 :** TMRB1 input capture 0
- **DMACA_TMRB1_INPUT_CAP1 :** TMRB1 input capture 1
- **DMACA_TMRB2_INPUT_CAP0 :** TMRB2 input capture 0
- **DMACA_TMRB2_INPUT_CAP1 :** TMRB2 input capture 1
- **DMACA_DMAREQA :** DMA UNITA request pin DMAREQA

For DMAC_UNIT_B:

- **DMACB_SNFC_GIE1 :** DMA UNITB request pin SNFC_GIE1
- **DMACB_SNFC_GIE2 :** DMA UNITB request pin SNFC_GIE2
- **DMACB_SNFC_GIE3 :** DMA UNITB request pin SNFC_GIE3
- **DMACB_SNFC_GIE4 :** DMA UNITB request pin SNFC_GIE4
- **DMACB_SNFC_GIE5 :** DMA UNITB request pin SNFC_GIE5
- **DMACB_SNFC_GIE6 :** DMA UNITB request pin SNFC_GIE6
- **DMACB_SNFC_GIE7 :** DMA UNITB request pin SNFC_GIE7
- **DMACB_SNFC_GIE8 :** DMA UNITB request pin SNFC_GIE8
- **DMACB_SNFC_GID11 :** DMA UNITB request pin SNFC_GIE11
- **DMACB_SNFC_GID12 :** DMA UNITB request pin SNFC_GIE12
- **DMACB_SNFC_GID13 :** DMA UNITB request pin SNFC_GIE13
- **DMACB_SNFC_GID14 :** DMA UNITB request pin SNFC_GIE14

- **DMACB_SNFC_GID15 :** DMA UNITB request pin SNFC_GIE15
- **DMACB_SNFC_GID16 :** DMA UNITB request pin SNFC_GIE16
- **DMACB_SNFC_GID17 :** DMA UNITB request pin SNFC_GIE17
- **DMACB_SNFC_GID18 :** DMA UNITB request pin SNFC_GIE18
- **DMACB_SNFC_GID21 :** DMA UNITB request pin SNFC_GIE21
- **DMACB_SNFC_GID22 :** DMA UNITB request pin SNFC_GIE22
- **DMACB_SNFC_GID23 :** DMA UNITB request pin SNFC_GIE23
- **DMACB_SNFC_GID24 :** DMA UNITB request pin SNFC_GIE24
- **DMACB_SNFC_GID25 :** DMA UNITB request pin SNFC_GIE25
- **DMACB_SNFC_GID26 :** DMA UNITB request pin SNFC_GIE26
- **DMACB_SNFC_GID27 :** DMA UNITB request pin SNFC_GIE27
- **DMACB_SNFC_GID28 :** DMA UNITB request pin SNFC_GIE28
- **DMACB_SSP0_RX :** SSP0 reception
- **DMACB_SSP0_TX :** SSP0 transmission
- **DMACB_SSP1_RX :** SSP1 reception
- **DMACB_SSP1_TX :** SSP1 transmission
- **DMACB_SSP2_RX :** SSP2 reception
- **DMACA_SSP2_TX :** SSP2 transmission
- **DMACB_DMAREQB :** DMA UNITB request pin DMAREQB

For DMAC_UNIT_C:

- **DMACC_SNFC_RD1 :** DMA UNITC request pin SNFC_RD1
- **DMACC_SNFC_RD2 :** DMA UNITC request pin SNFC_RD2
- **DMACC_SNFC_RD3 :** DMA UNITC request pin SNFC_RD3
- **DMACC_SNFC_RD4 :** DMA UNITC request pin SNFC_RD4
- **DMACC_SNFC_RD5 :** DMA UNITC request pin SNFC_RD5
- **DMACC_SNFC_RD6 :** DMA UNITC request pin SNFC_RD6
- **DMACC_SNFC_RD7 :** DMA UNITC request pin SNFC_RD7
- **DMACC_SNFC_RD8 :** DMA UNITC request pin SNFC_RD8
- **DMACC_AES_READ :** AES read
- **DMACC_AES_WRITE :** AES write
- **DMACC_SHA_WRITE :** SHA write
- **DMACC_DMA_COMPLETION :** DMA transfer completion
- **DMACC_I2C0_TX_RX :** I2C0 transmission/reception
- **DMACC_I2C1_TX_RX:** I2C1 transmission/reception
- **DMACC_I2C2_TX_RX :** I2C2 transmission/reception
- **DMACC_MPT0_CMP0_MATCH :** MPT0 compare match 0
- **DMACC_MPT0_CMP1_MATCH :** MPT0 compare match 1
- **DMACC_MPT1_CMP0_MATCH :** MPT1 compare match 0
- **DMACC_MPT1_CMP1_MATCH :** MPT1 compare match 1
- **DMACC_MPT2_CMP0_MATCH :** MPT2 compare match 0
- **DMACC_MPT2_CMP1_MATCH :** MPT2 compare match 1
- **DMACC_MPT3_CMP0_MATCH :** MPT3 compare match 0
- **DMACC_MPT3_CMP1_MATCH :** MPT3 compare match 1
- **DMACC_TMRB3_INPUT_CAP0 :** TMRB3 input capture 0
- **DMACC_TMRB3_INPUT_CAP1 :** TMRB3 input capture 1
- **DMACC_TMRB4_INPUT_CAP0 :** TMRB4 input capture 0
- **DMACC_TMRB4_INPUT_CAP1 :** TMRB4 input capture 1
- **DMACC_TMRB5_INPUT_CAP0 :** TMRB5 input capture 0
- **DMACC_TMRB5_INPUT_CAP1 :** TMRB5 input capture 1
- **DMACC_TMRB6_INPUT_CAP0 :** TMRB6 input capture 0
- **DMACC_TMRB6_INPUT_CAP1 :** TMRB6 input capture 1
- **DMACC_DMAREQC :** DMA UNITC request pin DMAREQC

20.2.3.1 DMAC_GetDMACState

Get the state of specified DMAC unit.

Prototype:
FunctionalState
DMAC_GetDMACState(TSB_DMA_TypeDef * ***DMACx***)

Parameters:
DMACx: Select DMAC unit.

Description:
This function will get the state of specified DMAC unit.

Return:

- **DISABLE:** The DMAC unit is disable
- **ENABLE:** The DMAC unit is enable

20.2.3.2 DMAC_Enable

Enable the specified DMAC unit.

Prototype:
void
DMAC_Enable(TSB_DMA_TypeDef * ***DMACx***)

Parameters:
DMACx: Select DMAC unit.

Description:
This function will enable the specified DMAC unit.

Return:
None

20.2.3.3 DMAC_Disable

Disable the specified DMAC unit.

Prototype:
void
DMAC_Disable(TSB_DMA_TypeDef * ***DMACx***)

Parameters:
DMACx: Select DMAC unit.

Description:
This function will disable the specified DMAC unit.

Return:
None

20.2.3.4 DMAC_SetPrimaryBaseAddr

Set the base address of the primary data of the specified DMAC unit.

Prototype:
void
DMAC_SetPrimaryBaseAddr(TSB_DMA_TypeDef * ***DMACx***,
uint32_t ***Addr***)

Parameters:
DMACx: Select DMAC unit.

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

Description:
This function will set the base address of the primary data of the specified DMAC unit.

Return:
None

20.2.3.5 DMAC_GetBaseAddr

Get the primary/alternative base address of the specified DMAC unit.

Prototype:
uint32_t
DMAC_GetBaseAddr(TSB_DMA_TypeDef * ***DMACx***,
DMAC_PrimaryAlt ***PriAlt***)

Parameters:
DMACx: Select DMAC unit.

PriAlt: Select base address type
This parameter can be one of the following values:

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

Description:
This function will get the primary/alternative base address of the specified DMAC unit.

Return:
The base address of primary/alternative data

20.2.3.6 DMAC_SetSWReq

Set software transfer request to the specified channel of the specified DMAC unit.

Prototype:
void
DMAC_SetSWReq(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel***)

Parameters:
DMACx: Select DMAC unit.

Channel: Select channel.

Description:
This function will set software transfer request to the specified channel by ***Channel*** of the specified DMAC unit.

Return:
None

20.2.3.7 DMACA_SetTransferType

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

Prototype:

```
void
DMACA_SetTransferType(uint8_t Channel,
                                   DMAC_TransferType Type)
```

Parameters:
Channel: Select UNITA channel.
This parameter can be one of the following values:
When *Type* is DMAC_BURST:

- **DMACA_SNFC_PRD11 :** DMA UNITA request pin SNFC_PRD11
- **DMACA_SNFC_PRD12 :** DMA UNITA request pin SNFC_PRD12
- **DMACA_SNFC_PRD21 :** DMA UNITA request pin SNFC_PRD21
- **DMACA_SMFC_PRD22 :** DMA UNITA request pin SNFC_PRD22
- **DMACA_ADC_COMPLETION :** ADC conversion completion
- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission
- **DMACA_UART1_RX :** UART1 reception
- **DMACA_UART1_TX :** UART1 transmission
- **DMACA_SIO0_UART0_RX :** SIO/UART0 reception
- **DMACA_SIO0_UART0_TX :** SIO/UART0 transmission
- **DMACA_SIO1_UART1_RX :** SIO/UART1 reception
- **DMACA_SIO1_UART1_TX :** SIO/UART1 transmission
- **DMACA_SIO2_UART2_RX :** SIO/UART2 reception
- **DMACA_SIO2_UART2_TX :** SIO/UART2 transmission
- **DMACA_SIO3_UART3_RX :** SIO/UART3 reception
- **DMACA_SIO3_UART3_TX :** SIO/UART3 transmission
- **DMACA_TMRB0_CMP_MATCH :** TMRB0 compare match
- **DMACA_TMRB1_CMP_MATCH :** TMRB1 compare match
- **DMACA_TMRB2_CMP_MATCH :** TMRB2 compare match
- **DMACA_TMRB3_CMP_MATCH :** TMRB3 compare match
- **DMACA_TMRB4_CMP_MATCH :** TMRB4 compare match
- **DMACA_TMRB5_CMP_MATCH :** TMRB5 compare match
- **DMACA_TMRB6_CMP_MATCH :** TMRB6 compare match
- **DMACA_TMRB7_CMP_MATCH :** TMRB7 compare match
- **DMACA_TMRB0_INPUT_CAP0 :** TMRB0 input capture 0
- **DMACA_TMRB0_INPUT_CAP1 :** TMRB0 input capture 1
- **DMACA_TMRB1_INPUT_CAP0 :** TMRB1 input capture 0
- **DMACA_TMRB1_INPUT_CAP1 :** TMRB1 input capture 1
- **DMACA_TMRB2_INPUT_CAP0 :** TMRB2 input capture 0
- **DMACA_TMRB2_INPUT_CAP1 :** TMRB2 input capture 1
- **DMACA_DMAREQA :** DMA UNITA request pin DMAREQA

When *Type* is DMAC_SINGLE:

- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission
- **DMACA_UART1_RX :** UART1 reception
- **DMACA_UART1_TX :** UART1 transmission

Type: Select transfer type.
This parameter can be one of the following values:

- **DMAC_BURST :** Single transfer is disable, only burst transfer request can be used
- **DMAC_SINGLE :** Single transfer is enable

Description:
This function will set transfer type to the specified channel of the DMAC UNITA.

Return:
None

20.2.3.8 DMACA_GetTransferType

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

Prototype:
DMAC_TransferType
DMACA_GetTransferType(uint8_t ***Channel***)

Parameters:
Channel: Select UNITA channel.
The parameter can be one of the following values:

- **DMACA_SNFC_PRD11 :** DMA UNITA request pin SNFC_PRD11
- **DMACA_SNFC_PRD12 :** DMA UNITA request pin SNFC_PRD12
- **DMACA_SNFC_PRD21 :** DMA UNITA request pin SNFC_PRD21
- **DMACA_SMFC_PRD22 :** DMA UNITA request pin SNFC_PRD22
- **DMACA_ADC_COMPLETION :** ADC conversion completion
- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission
- **DMACA_UART1_RX :** UART1 reception
- **DMACA_UART1_TX :** UART1 transmission
- **DMACA_SIO0_UART0_RX :** SIO/UART0 reception
- **DMACA_SIO0_UART0_TX :** SIO/UART0 transmission
- **DMACA_SIO1_UART1_RX :** SIO/UART1 reception
- **DMACA_SIO1_UART1_TX :** SIO/UART1 transmission
- **DMACA_SIO2_UART2_RX :** SIO/UART2 reception
- **DMACA_SIO2_UART2_TX :** SIO/UART2 transmission
- **DMACA_SIO3_UART3_RX :** SIO/UART3 reception
- **DMACA_SIO3_UART3_TX :** SIO/UART3 transmission
- **DMACA_TMRB0_CMP_MATCH :** TMRB0 compare match
- **DMACA_TMRB1_CMP_MATCH :** TMRB1 compare match
- **DMACA_TMRB2_CMP_MATCH :** TMRB2 compare match
- **DMACA_TMRB3_CMP_MATCH :** TMRB3 compare match
- **DMACA_TMRB4_CMP_MATCH :** TMRB4 compare match
- **DMACA_TMRB5_CMP_MATCH :** TMRB5 compare match
- **DMACA_TMRB6_CMP_MATCH :** TMRB6 compare match
- **DMACA_TMRB7_CMP_MATCH :** TMRB7 compare match
- **DMACA_TMRB0_INPUT_CAP0 :** TMRB0 input capture 0
- **DMACA_TMRB0_INPUT_CAP1 :** TMRB0 input capture 1
- **DMACA_TMRB1_INPUT_CAP0 :** TMRB1 input capture 0

- **DMACA_TMRB1_INPUT_CAP1 :** TMRB1 input capture 1
- **DMACA_TMRB2_INPUT_CAP0 :** TMRB2 input capture 0
- **DMACA_TMRB2_INPUT_CAP1 :** TMRB2 input capture 1
- **DMACA_DMAREQA :** DMA UNITA request pin DMAREQA

Description:
This function will get transfer type setting for the specified channel of the DMAC UNITA.

Return:
The transfer type with DMAC_TransferType type:

- **DMAC_BURST :** Single transfer is disable, only burst transfer request can be used
- **DMAC_SINGLE :** Single transfer is enable

20.2.3.9 DMACB_SetTransferType

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

Prototype:
void
DMACB_SetTransferType(uint8_t ***Channel***,
DMAC_TransferType ***Type***)

Parameters:
Channel: Select UNITB channel.
This parameter can be one of the following values:
When *Type* is DMAC_BURST:

- **DMACB_SNFC_GIE1 :** DMA UNITB request pin SNFC_GIE1
- **DMACB_SNFC_GIE2 :** DMA UNITB request pin SNFC_GIE2
- **DMACB_SNFC_GIE3 :** DMA UNITB request pin SNFC_GIE3
- **DMACB_SNFC_GIE4 :** DMA UNITB request pin SNFC_GIE4
- **DMACB_SNFC_GIE5 :** DMA UNITB request pin SNFC_GIE5
- **DMACB_SNFC_GIE6 :** DMA UNITB request pin SNFC_GIE6
- **DMACB_SNFC_GIE7 :** DMA UNITB request pin SNFC_GIE7
- **DMACB_SNFC_GIE8 :** DMA UNITB request pin SNFC_GIE8
- **DMACB_SNFC_GID11 :** DMA UNITB request pin SNFC_GIE11
- **DMACB_SNFC_GID12 :** DMA UNITB request pin SNFC_GIE12
- **DMACB_SNFC_GID13 :** DMA UNITB request pin SNFC_GIE13
- **DMACB_SNFC_GID14 :** DMA UNITB request pin SNFC_GIE14
- **DMACB_SNFC_GID15 :** DMA UNITB request pin SNFC_GIE15
- **DMACB_SNFC_GID16 :** DMA UNITB request pin SNFC_GIE16
- **DMACB_SNFC_GID17 :** DMA UNITB request pin SNFC_GIE17
- **DMACB_SNFC_GID18 :** DMA UNITB request pin SNFC_GIE18
- **DMACB_SNFC_GID21 :** DMA UNITB request pin SNFC_GIE21
- **DMACB_SNFC_GID22 :** DMA UNITB request pin SNFC_GIE22
- **DMACB_SNFC_GID23 :** DMA UNITB request pin SNFC_GIE23
- **DMACB_SNFC_GID24 :** DMA UNITB request pin SNFC_GIE24
- **DMACB_SNFC_GID25 :** DMA UNITB request pin SNFC_GIE25
- **DMACB_SNFC_GID26 :** DMA UNITB request pin SNFC_GIE26
- **DMACB_SNFC_GID27 :** DMA UNITB request pin SNFC_GIE27
- **DMACB_SNFC_GID28 :** DMA UNITB request pin SNFC_GIE28
- **DMACB_SSP0_RX :** SSP0 reception
- **DMACB_SSP0_TX :** SSP0 transmission
- **DMACB_SSP1_RX :** SSP1 reception

- **DMACB_SSP1_TX :** SSP1 transmission
- **DMACB_SSP2_RX :** SSP2 reception
- **DMACA_SSP2_TX :** SSP2 transmission
- **DMACB_DMAREQB :** DMA UNITB request pin DMAREQB

When *Type* is DMAC_SINGLE:
- **DMACB_SSP0_RX :** SSP0 reception
- **DMACB_SSP0_TX :** SSP0 transmission
- **DMACB_SSP1_RX :** SSP1 reception
- **DMACB_SSP1_TX :** SSP1 transmission
- **DMACB_SSP2_RX :** SSP2 reception
- **DMACA_SSP2_TX :** SSP2 transmission

Type: Select transfer type.
This parameter can be one of the following values:
- **DMAC_BURST :** Single transfer is disabled, only burst transfer request can be used
- **DMAC_SINGLE :** Single transfer is enabled

Description:
This function will set transfer type to the specified channel of the DMAC UNITB.

Return:
None

20.2.3.10 DMACB_GetTransferType

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

Prototype:
DMAC_TransferType
DMACB_GetTransferType(uint8_t ***Channel***)

Parameters:
Channel: Select UNITB channel.
The parameter can be one of the following values:
- **DMACB_SNFC_GIE1 :** DMA UNITB request pin SNFC_GIE1
- **DMACB_SNFC_GIE2 :** DMA UNITB request pin SNFC_GIE2
- **DMACB_SNFC_GIE3 :** DMA UNITB request pin SNFC_GIE3
- **DMACB_SNFC_GIE4 :** DMA UNITB request pin SNFC_GIE4
- **DMACB_SNFC_GIE5 :** DMA UNITB request pin SNFC_GIE5
- **DMACB_SNFC_GIE6 :** DMA UNITB request pin SNFC_GIE6
- **DMACB_SNFC_GIE7 :** DMA UNITB request pin SNFC_GIE7
- **DMACB_SNFC_GIE8 :** DMA UNITB request pin SNFC_GIE8
- **DMACB_SNFC_GID11 :** DMA UNITB request pin SNFC_GID11
- **DMACB_SNFC_GID12 :** DMA UNITB request pin SNFC_GID12
- **DMACB_SNFC_GID13 :** DMA UNITB request pin SNFC_GID13
- **DMACB_SNFC_GID14 :** DMA UNITB request pin SNFC_GID14
- **DMACB_SNFC_GID15 :** DMA UNITB request pin SNFC_GID15
- **DMACB_SNFC_GID16 :** DMA UNITB request pin SNFC_GID16
- **DMACB_SNFC_GID17 :** DMA UNITB request pin SNFC_GID17
- **DMACB_SNFC_GID18 :** DMA UNITB request pin SNFC_GID18
- **DMACB_SNFC_GID21 :** DMA UNITB request pin SNFC_GID21
- **DMACB_SNFC_GID22 :** DMA UNITB request pin SNFC_GID22
- **DMACB_SNFC_GID23 :** DMA UNITB request pin SNFC_GID23
- **DMACB_SNFC_GID24 :** DMA UNITB request pin SNFC_GID24

- **DMACB_SNFC_GID25 :** DMA UNITB request pin SNFC_GID25
- **DMACB_SNFC_GID26 :** DMA UNITB request pin SNFC_GID26
- **DMACB_SNFC_GID27 :** DMA UNITB request pin SNFC_GID27
- **DMACB_SNFC_GID28 :** DMA UNITB request pin SNFC_GID28
- **DMACB_SSP0_RX :** SSP0 reception
- **DMACB_SSP0_TX :** SSP0 transmission
- **DMACB_SSP1_RX :** SSP1 reception
- **DMACB_SSP1_TX :** SSP1 transmission
- **DMACB_SSP2_RX :** SSP2 reception
- **DMACA_SSP2_TX :** SSP2 transmission
- **DMACB_DMAREQB :** DMA UNITB request pin DMAREQB

Description:
This function will get transfer type setting for the specified channel of the DMAC UNITB.

Return:
The transfer type with DMAC_TransferType type:

- **DMAC_BURST :** Single transfer is disable, only burst transfer request can be used
- **DMAC_SINGLE :** Single transfer is enable

20.2.3.11 DMAC_SetMask

Set mask for the specified channel of the specified DMAC unit.

Prototype:
void
DMAC_SetMask(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel*** ,
FunctionalState ***NewState***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

NewState: Clear or set the mask to enable or disable the DMA channel.
This parameter can be one of the following values:

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

Description:
This function will set mask for the specified channel of the specified DMAC unit.

Return:
None

20.2.3.12 DMAC_GetMask

Get mask setting for the specified channel of the specified DMAC unit.

Prototype:

FunctionalState
DMAC_GetMask(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

Description:
This function will get mask setting for the specified channel of the specified DMAC unit.

Return:
The inverted mask setting:

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

20.2.3.13 DMAC_SetChannel

Enable or disable the specified channel of the specified DMAC unit.

Prototype:
void
DMAC_SetChannel(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel*** ,
FunctionalState ***NewState***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

NewState: Enable or disable the DMA channel.
This parameter can be one of the following values:

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

Description:
This function will enable or disable the specified channel of the specified DMAC unit. by ***NewState***.

Return:
None

20.2.3.14 DMAC_GetChannelState

Get the enable/disable setting for specified channel of the specified DMAC unit.

Prototype:
FunctionalState
DMAC_GetChannelState(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel***)

Parameters:

DMACx: Select DMAC unit.
Channel: Select channel.

Description:
This function will get the enable/disable setting for specified channel of the specified DMAC unit.

Return:
The enable/disable setting for channel:

- **ENABLE:** The DMA channel is enable
- **DISABLE:** The DMA channel is disable

20.2.3.15 DMAC_SetPrimaryAlt

Set to use primary data or alternative data for specified channel of the specified DMAC unit.

Prototype:
void
DMAC_SetPrimaryAlt(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel*** ,
DMAC_PrimaryAlt ***PriAlt***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

PriAlt: Select primary data or alternative data for channel specified by 'ChannelA' above.
This parameter can be one of the following values:

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

Description:
This function will set to use primary data or alternative data for specified channel of the specified DMAC unit.

Return:
None

20.2.3.16 DMAC_GetPrimaryAlt

Get the setting of the using of primary data or alternative data for specified channel of the specified DMAC unit.

Prototype:
DMAC_PrimaryAlt
DMAC_GetPrimaryAlt(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

Description:

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

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

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

20.2.3.17 DMAC_SetChannelPriority

Set the priority for specified channel of the specified DMAC unit.

Prototype:
void
DMAC_SetChannelPriority(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel*** ,
DMAC_Priority ***Priority***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

Priority: Select Priority.
This parameter can be one of the following values:

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

Description:
This function will set the priority for specified channel of the specified DMAC unit.

Return:
None

20.2.3.18 DMAC_GetChannelPriority

Get the priority setting for specified channel of the specified DMAC unit.

Prototype:
DMAC_Priority
DMAC_GetChannelPriority(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

Description:
This function will get the priority setting for specified channel of the specified DMAC unit

Return:
The priority setting of channel:

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

20.2.3.19 DMAC_ClearBusErr

Clear the bus error of the specified DMAC unit.

Prototype:
void
DMAC_ClearBusErr(TSB_DMA_TypeDef * ***DMACx***)

Parameters:
DMACx: Select DMAC unit.

Description:
This function will clear the bus error of the specified DMAC unit.

Return:
None

20.2.3.20 DMAC_GetBusErrState

Get the bus error state of the specified DMAC unit.

Prototype:
Result
DMAC_GetBusErrState(TSB_DMA_TypeDef * ***DMACx***)

Parameters:
DMACx: Select DMAC unit.

Description:
This function will get the bus error state of the specified DMAC unit.

Return:
The bus error state:

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

20.2.3.21 DMAC_FillInitData

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

Prototype:
void
DMAC_FillInitData(TSB_DMA_TypeDef * ***DMACx*** ,
uint8_t ***Channel*** ,
DMAC_InitTypeDef * ***InitStruct***)

Parameters:
DMACx: Select DMAC unit.
Channel: Select channel.

InitStruct: The structure contains the DMA setting values.

Description:
This function will fill the DMA setting data of specified channel of the DMAC UNITA to RAM.

Return:
None

20.2.3.22 DMACA_GetINTFlag

Get the DMA factor flag of the DMAC UNITA

Prototype:
DMACA_Flag
DMACA_GetINTFlag(void)

Parameters:
None

Description:
This function will get the DMA factor flag of the DMAC UNITA

Return:
A union with DMA factor flag of DMAC UNITA(refer to Data Structure Description of DMACA_Flag for details)

20.2.3.23 DMACB_GetINTFlag

Get the DMA factor flag of the DMAC UNITB

Prototype:
DMACB_Flag
DMACB_GetINTFlag(void)

Parameters:
None

Description:
This function will get the DMA factor flag of the DMAC UNITB

Return:
A union with DMA factor flag of DMAC UNITB(refer to Data Structure Description of DMACB_Flag for details)

20.2.3.24 DMACC_GetINTFlag

Get the DMA factor flag of the DMAC UNITC

Prototype:
DMACC_Flag
DMACC_GetINTFlag(void)

Parameters:
None

Description:
This function will get the DMA factor flag of the DMAC UNITC

Return:
A union with DMA factor flag of DMAC UNITC(refer to Data Structure Description of DMACC_Flag for details)

20.2.4 Data Structure Description

20.2.4.1 DMAC_InitTypeDef

Data fields:
uint32_t
SrcEndPointer: The final address of data source.

uint32_t
DstEndPointer: The final address of data destination.

DMAC_CycleCtrl
Mode: Set operation mode,
which can be:

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

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

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

uint32_t
TxNum: Set the actual number of transfers. Maximum is 1024.

DMAC_Arbitration
ArbitrationMoment: Specifies the arbitration moment(R_Power).
It can be one of the following values:

- **DMAC_AFTER_1_TX:** After 1 transfer
- **DMAC_AFTER_2_TX:** After 2 transfers
- **DMAC_AFTER_4_TX:** After 4 transfers
- **DMAC_AFTER_8_TX:** After 8 transfers
- **DMAC_AFTER_16_TX:** After 16 transfers
- **DMAC_AFTER_32_TX:** After 32 transfers
- **DMAC_AFTER_64_TX:** After 64 transfers
- **DMAC_AFTER_128_TX:** After 128 transfers

- **DMAC_AFTER_256_TX:** After 256 transfers
- **DMAC_AFTER_512_TX:** After 512 transfers
- **DMAC_NEVER:** No arbitration

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

DMAC_BitWidth
SrcWidth: Set source bit width,
which can be:

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

DMAC_IncWidth
SrcInc: Set increment of the source address,
which can be:

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

DMAC_BitWidth
DstWidth: Set destination bit width,
which can be:

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

DMAC_IncWidth
DstInc: Set increment of the destination address,
which can be:

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

20.2.4.2 DMACA_Flag

Data Fields for this union:

uint32_t
All The flag of DMA UINTA interrupt.

Bit Fields:

uint32_t
SNFC_PRD11 (Bit 0) The flag of SNFC_PRD11 occurs an interrupt
'1' means occurs an interrupt

uint32_t
SNFC_PRD12 (Bit 1) The flag of SNFC_PRD12 occurs an interrupt
'1' means occurs an interrupt

uint32_t

SNFC_PRD21 (Bit 2)	The flag of SNFC_PRD21 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	
SNFC_PRD22 (Bit 3)	The flag of SNFC_PRD22 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	
ADCCompletion (Bit 4)	The flag of ADC completion occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
UART0Reception (Bit 5)	The flag of UART0 reception occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	
UART0Transmission (Bit 6)	The flag of UART0 transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	
UART1Reception (Bit 7)	The flag of UART1 reception occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	
UART1Transmission (Bit 8)	The flag of UART1 transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	
SIO_UART0Reception (Bit 9)	The flag of SIO/UART0 reception occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
SIO_UART0Transmission (Bit 10)	The flag of SIO/UART0 transmission occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
SIO_UART1Reception (Bit 11)	The flag of SIO/UART1 reception occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
SIO_UART1Transmission (Bit 12)	The flag of SIO/UART1 transmission occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
SIO_UART2Reception (Bit 13)	The flag of SIO/UART2 reception occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
SIO_UART2Transmission (Bit 14)	The flag of SIO/UART2 transmission occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	
SIO_UART3Reception (Bit 15)	The flag of SIO/UART3 reception occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t	

SIO_UART3Transmission (Bit 16)The flag of SIO/UART3 transmission
occurs an interrupt.
'1' means occurs an interrupt
uint32_t
TMRB0CompareMatch (Bit 17) The flag of TMRB0 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB1CompareMatch (Bit 18) The flag of TMRB1 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB2CompareMatch (Bit 19) The flag of TMRB2 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB3CompareMatch (Bit 20) The flag of TMRB3 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB4CompareMatch (Bit 21) The flag of TMRB4 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB5CompareMatch (Bit 22) The flag of TMRB5 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB6CompareMatch (Bit 23) The flag of TMRB6 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB7CompareMatch (Bit 24) The flag of TMRB7 compare match
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB0InputCapture0 (Bit 25) The flag of TMRB0 input capture 0
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB0InputCapture1 (Bit 26) The flag of TMRB0 input capture 1
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB1InputCapture0 (Bit 27) The flag of TMRB1 input capture 0
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB1InputCapture1 (Bit 28) The flag of TMRB1 input capture 1
occurs an interrupt
'1' means occurs an interrupt
uint32_t
TMRB2InputCapture0 (Bit 29) The flag of TMRB2 input capture 0
occurs an interrupt
'1' means occurs an interrupt
uint32_t

TMRB2InputCapture1 (Bit 30)	The flag of TMRB2 input capture 1 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***DMAREQA*** (Bit 31)	The flag of pin DMAREQA occurs an interrupt ‘1’ means occurs an interrupt

20.2.4.3 DMACB_Flag

Data Fields for this union:

uint32_t
All The flag of DMA UINTB interrupt.

Bit Fields:

uint32_t ***SNFC_GIE1*** (Bit 0)	The flag of SNFC_GIE1 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE2*** (Bit 1)	The flag of SNFC_GIE2 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE3*** (Bit 2)	The flag of SNFC_GIE3 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE4*** (Bit 3)	The flag of SNFC_GIE4 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE5*** (Bit 4)	The flag of SNFC_GIE5 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE6*** (Bit 5)	The flag of SNFC_GIE6 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE7*** (Bit 6)	The flag of SNFC_GIE7 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GIE8*** (Bit 7)	The flag of SNFC_GIE8 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID11*** (Bit 8)	The flag of SNFC_GIE11 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID12*** (Bit 9)	The flag of SNFC_GIE12 occurs an interrupt ‘1’ means occurs an interrupt

uint32_t ***SNFC_GID13*** (Bit 10)	The flag of SNFC_GIE13 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID14*** (Bit 11)	The flag of SNFC_GIE14 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID15*** (Bit 12)	The flag of SNFC_GIE15 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID16*** (Bit 13)	The flag of SNFC_GIE16 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID17*** (Bit 14)	The flag of SNFC_GIE17 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID18*** (Bit 15)	The flag of SNFC_GIE18 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID21*** (Bit 16)	The flag of SNFC_GIE21 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID22*** (Bit 17)	The flag of SNFC_GIE22 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID23*** (Bit 18)	The flag of SNFC_GIE23 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID24*** (Bit 19)	The flag of SNFC_GIE24 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID25*** (Bit 20)	The flag of SNFC_GIE25 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID26*** (Bit 21)	The flag of SNFC_GIE26 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID27*** (Bit 22)	The flag of SNFC_GIE27 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_GID28*** (Bit 23)	The flag of SNFC_GIE28 occurs an interrupt ‘1’ means occurs an interrupt

uint32_t ***ADCCompletion*** (Bit 24)	The flag of ADC completion occurs an interrupt. ‘1’ means occurs an interrupt
uint32_t ***SSP0Reception*** (Bit 25)	The flag of SSP0 reception occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SSP0Transmission*** (Bit 26)	The flag of SSP0 transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SSP1Reception*** (Bit 27)	The flag of SSP1 reception occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SSP1Transmission*** (Bit 28)	The flag of SSP1 transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SSP2Reception*** (Bit 29)	The flag of SSP2 reception occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SSP2Transmission*** (Bit 30)	The flag of SSP2 transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***DMAREQB*** (Bit 31)	The flag of pin DMAREQB occurs an interrupt ‘1’ means occurs an interrupt

20.2.4.4 DMACC_Flag

Data Fields for this union:

uint32_t
All The flag of DMA UINTC interrupt.

Bit Fields:

uint32_t ***SNFC_RD1*** (Bit 0)	The flag of SNFC_RD1 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_RD2*** (Bit 1)	The flag of SNFC_RD2 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_RD3*** (Bit 2)	The flag of SNFC_RD3 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_RD4*** (Bit 3)	The flag of SNFC_RD4 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t	

SNFC_RD5 (Bit 4)	The flag of SNFC_RD5 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_RD6*** (Bit 5)	The flag of SNFC_RD6 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_RD7*** (Bit 6)	The flag of SNFC_RD7 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***SNFC_RD8*** (Bit 7)	The flag of SNFC_RD8 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***AES_Read*** (Bit 8)	The flag of SNFC_RD1 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***AES_Write*** (Bit 9)	AES write completion ‘1’ means occurs an interrupt
uint32_t ***SHA_Write*** (Bit 10)	SHA write completion ‘1’ means occurs an interrupt
uint32_t ***DMA_SHA_Completion*** (Bit 11)	DMA ch10(SHA write) completion ‘1’ means occurs an interrupt
uint32_t ***I2C0RxorTx*** (Bit 12)	The flag of I2C0 reception/transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***I2C1RxorTx*** (Bit 13)	The flag of I2C1 reception/transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***I2C2RxorTx*** (Bit 14)	The flag of I2C2 reception/transmission occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***MPT0CompareMatch0*** (Bit 15)	The flag of MPT0 compare match0 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***MPT0CompareMatch1*** (Bit 16)	The flag of MPT0 compare match1 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***MPT1CompareMatch0*** (Bit 17)	The flag of MPT1 compare match0 occurs an interrupt ‘1’ means occurs an interrupt
uint32_t ***MPT1CompareMatch1*** (Bit 18)	The flag of MPT1 compare match1 occurs an interrupt ‘1’ means occurs an interrupt

uint32_t
MPT2CompareMatch0 (Bit 19) The flag of MPT2 compare match0 occurs an interrupt
'1' means occurs an interrupt

uint32_t
MPT2CompareMatch1 (Bit 20) The flag of MPT2 compare match1 occurs an interrupt
'1' means occurs an interrupt

uint32_t
MPT3CompareMatch0 (Bit 21) The flag of MPT3 compare match0 occurs an interrupt
'1' means occurs an interrupt

uint32_t
MPT3CompareMatch1 (Bit 22) The flag of MPT3 compare match1 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB3InputCapture0 (Bit 23) The flag of TMRB3 input capture 0 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB3InputCapture1 (Bit 24) The flag of TMRB3 input capture 1 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB4InputCapture0 (Bit 25) The flag of TMRB4 input capture 0 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB4InputCapture1 (Bit 26) The flag of TMRB4 input capture 1 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB5InputCapture0 (Bit 27) The flag of TMRB5 input capture 0 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB5InputCapture1 (Bit 28) The flag of TMRB5 input capture 1 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB6InputCapture0 (Bit 29) The flag of TMRB6 input capture 0 occurs an interrupt
'1' means occurs an interrupt

uint32_t
TMRB6InputCapture1 (Bit 30) The flag of TMRB6 input capture 1 occurs an interrupt
'1' means occurs an interrupt

uint32_t
DMAREQC (Bit 31) The flag of pin DMAREQC occurs an interrupt
'1' means occurs an interrupt

21. WDT

21.1 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\tmpm46b_wdt.c, with \Libraries\TX04_Periph_Driver\inc\tmpm46b_wdt.h containing the API definitions for use by applications.

21.2 API Functions

21.2.1 Function List

- void WDT_SetDetectTime(uint32_t ***DetectTime***)
- Result WDT_SetIdleMode(FunctionalState ***NewState***)
- Result WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)
- Result WDT_Init(WDT_InitTypeDef * ***InitStruct***)
- Result WDT_Enable(void)
- Result WDT_Disable(void)
- Result WDT_WriteClearCode(void)
- FunctionalState WDT_GetWritingFlg(void)

21.2.2 Detailed Description

Functions listed above can be divided into three 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().
3) The flag that enable or disable writing to WDMOD or WDCR is handled by the WDT_GetWritingFlg().

21.2.3 Function Documentation

21.2.3.1 WDT_SetDetectTime

Set detection time for WDT.

Prototype:
Result
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/fIHOSC
- **WDT_DETECT_TIME_EXP_17:** ***DetectTime*** is 2^17/fIHOSC
- **WDT_DETECT_TIME_EXP_19:** ***DetectTime*** is 2^19/fIHOSC
- **WDT_DETECT_TIME_EXP_21:** ***DetectTime*** is 2^21/fIHOSC

- **WDT_DETECT_TIME_EXP_23:** ***DetectTime*** is 2^23/fIHOSC
- **WDT_DETECT_TIME_EXP_25:** ***DetectTime*** is 2^25/fIHOSC

Description:
This function will set detection time for WDT.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.2 WDT_SetIdleMode

Run or stop the WDT counter when the system enters IDLE mode.

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

Parameters:
NewState: Run or stop WDT counter.
This parameter can be one of the following values:

- **ENABLE**: Run the WDT counter.
- **DISABLE**. Stop the WDT counter.

Description:
This function will run the WDT counter when the system enters IDLE mode when ***NewState*** is **ENABLE**, and stop the WDT counter when the system enters IDLE mode when ***NewState*** is **DISABLE**.

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

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.3 WDT_SetOverflowOutput

Set WDT to generate NMI interrupt or reset when the counter overflows.

Prototype:
Result
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:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.4 WDT_Init

Initialize and configure WDT.

Prototype:
Result
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:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.5 WDT_Enable

Enable the WDT function.

Prototype:
Result
WDT_Enable(void)

Parameters:
None

Description:
This function will enable WDT.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.6 WDT_Disable

Disable the WDT function.

Prototype:
Result
WDT_Disable(void)

Parameters:

None

Description:
This function will disable WDT.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.7 WDT_WriteClearCode

Write the clear code.

Prototype:
Result
WDT_WriteClearCode (void)

Parameters:
None

Description:
This function will clear the WDT counter.

Return:
SUCCESS means set successful.
ERROR means set failed and do nothing.

21.2.3.1 WDT_GetWritingFlg

Get the flag for writing to registers.

Prototype:
FunctionalState
WDT_GetWritingFlg (void)

Parameters:
None

Description:
This function will get the flag for writing to registers

***Note:**
When writing to WDMOD or WDCR, confirm writing flag enable.

Return:
The flag for writing to registers.
The value returned can be one of the following values:
ENABLE: Writing to WDT registers is accessible.
DISABLE: Writing to WDT registers is not accessible.

21.2.4 Data Structure Description

21.2.4.1 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/fIHOSC
- **WDT_DETECT_TIME_EXP_17:** ***DetectTime*** is 2^17/fIHOSC
- **WDT_DETECT_TIME_EXP_19:** ***DetectTime*** is 2^19/fIHOSC
- **WDT_DETECT_TIME_EXP_21:** ***DetectTime*** is 2^21/fIHOSC
- **WDT_DETECT_TIME_EXP_23:** ***DetectTime*** is 2^23/fIHOSC
- **WDT_DETECT_TIME_EXP_25:** ***DetectTime*** is 2^25/fIHOSC

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.