TOSHIBA

TOSHIBA TX03 Peripheral Driver User Guide
(TMPM311)

Ver 1.000
Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M311UG-00xE

RESTRICTIONS ON PRODUCT USE

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

Index

1 Introduction

TOSHIBA TX03 Peripheral Driver is a set of drivers for all peripherals found on the TOSHIBA TX03 series microcontrollers. TMPM311CHDUG Peripheral Driver is an important part of TOSHIBA TX03 Peripheral Driver, which is designed for TMPM311CHDUG series MCUs.

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

The design goals of TOSHIBA TMPM311CHDUG Peripheral Driver:
- Completely written in C except the start-up routine and where not possible
- Cover all the peripherals on MCU

2 Organization of TOSHIBA TX03 Peripheral Driver

/TMPM311 /Libraries
This folder contains all CMSIS files and TMPM311 Peripheral Drivers.

/TMPM311 /Libraries/ TX03_CMSIS
This folder contains the device peripheral access layer of TMPM311 CMSIS files.

/TMPM311 /Libraries/TX03_Periph_Driver
This folder contains all the source code of the drivers, the core of TOSHIBA TMPM311 Peripheral Driver.

/TMPM311 /Libraries/TX03_Periph_Driver/inc
This folder contains all the header files of TMPM311 Peripheral Drivers for each peripheral.

/TMPM311 /Libraries/TX03_Periph_Driver/src
This folder contains all the source files of TMPM311 Peripheral Drivers for each peripheral.

/TMPM311 /Project
This folder contains template project and examples for using TMPM311 Peripheral Driver.

/TMPM311 /Project/Template
This folder contains template project of TOSHIBA TMPM311 Peripheral Driver.

/TMPM311 /Project/Examples
This folder contains a set of examples for using TMPM311 Peripheral Driver

/TMPM311 /Project/Examples/Utilities/TMPM311-EVAL
This folder contains the configuration and driver files for hardware resources (e.g. led, key) on TMPM311 boards.

3 CG

3.1 Overview

The CG API provides a set of functions for using the TMPM311CHDUG CG module as the following:

- Set up high-speed oscillators and input clock.
- Select clock gear, prescaler clock and oscillator.
- Set warm up timer and read the warm up result.
- Clear interrupt request.

This driver is contained in /TX03_Periph_Driver/src/tmpm311_cg.c, with /TX03_Periph_Driver/inc/tmpm311_cg.h containing the API definitions for use by applications.

The following symbols fosc, fc, fgear, fsys, fperiph, ΦT0 are used for kinds of clock in CG. Please refer to the clock system diagram in section "Schematic diagram of the clocks of the datasheet for their meaning.

EHCLKIN: Clock input from the X1 pin
EHOSC: Output clock from the external high-speed oscillator
IHOSC: Output clock from the internal high-speed oscillator.
fosc: Clock specified by CG0OSCEN<EHOSCEN[1:0]>
fc: Clock specified by CG0OSCSEL<OSCSEL[1:0]> (high-speed clock).
fgear: Clock specified by CG0CLKCR<GEAR[2:0]>.
fsys: Clock specified by CG0CLKCR<GEAR[2:0]>.(system clock)
fperiph: Clock specified by CG0CLKCR<FPSEL>.
ΦT0: Clock specified by CG0CLKCR<PRCK[2:0]> (prescaler clock).

3.2 API Functions

3.2.1 Function List

- void CG_SetFgearLevel(CG_DivideLevel ***DivideFgearFromFc***)
- CG_DivideLevel CG_GetFgearLevel(void)
- void CG_SetPhiT0Src(CG_PhiT0Src ***PhiT0Src***)
- CG_PhiT0Src CG_GetPhiT0Src(void)
- void CG_SetSysTickSrc(CG_SysTickSrc ***SysTickSrc)***
- CG_SysTickSrc CG_GetSysTickSrc(void)
- Result CG_SetPhiT0Level(CG_DivideLevel ***DividePhiT0FromFc***)
- CG_DivideLevel CG_GetPhiT0Level(void)
- void CG_SetWarmUpTime(CG_WarmUpSrc ***Source***, uint16_t ***Time***)
- void CG_StartWarmUp(void)
- WorkState CG_GetWarmUpState(void)
- Result CG_SetFosc(CG_FoscSrc Source, FunctionalState ***NewState***)
- void CG_SetFoscSrc(CG_FoscSrc ***Source***)
- CG_FoscSrc CG_GetFoscSrc(void)
- FunctionalState CG_GetFoscState(CG_FoscSrc ***Source***)
- 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***)

- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***)
- void CG_ClearINTReq(CG_INTSrc ***INTSource***)
- CG_ResetFlag CG_GetResetFlag(void)

3.2.2 Detailed Description

The CG APIs can be broken into two 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_SetSysTickSrc(), CG_GetSysTickSrc(), CG_SetPhiT0Level(), CG_GetPhiT0Level(),CG_SetWarmUpTime(), CG_StartWarmUp(), CG_GetWarmUpState(),CG_SetFosc(),CG_SetFoscSrc(), CG_GetFoscSrc(), CG_GetFoscState(),CG_SetFcSrc(),CG_GetFcSrc(),CG_SetProtectCtrl().

2) The 2nd group of APIs handle settings of interrupts:
CG_SetSTBYReleaseINTSrc(), CG_GetSTBYReleaseINTState(), CG_ClearINTReq(),CG_GetResetFlag().

3.2.3 Function Documentation

3.2.3.1 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

3.2.3.2 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

3.2.3.3 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

3.2.3.4 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.

3.2.3.5 CG_SetSysTickSrc

Set source clock for the SysTick reference clock.

Prototype:
void

CG_SetSysTickSrc (CG_SysTickSrc ***SysTickSrc)***

Parameters:
SysTickSrc: Select SysTick source.
This parameter can be one of the following values:

- **CG_STICK_SRC_IHOSC** means SysTick reference clock source is IHOSC.
- **CG_STICK_SRC_FOSC** means SysTick reference clock source is fosc.

Description:
This function selects the source for SysTick reference clock.

Return:
None

3.2.3.6 CG_GetSysTickSrc

Get the SysTick reference clock source.

Prototype:
CG_SysTickSrc
CG_GetSysTickSrc(void)

Parameters:
None

Description:
This function will get the SysTick reference clock source.

Return:
CG_STICK_SRC_IHOSC means SysTick reference clock source is IHOSC.
CG_STICK_SRC_FOSC means SysTick reference clock source is fosc.

3.2.3.7 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**.**

3.2.3.8 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.

3.2.3.9 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.

Time:
If ***Source*** is **CG_WARM_UP_SRC_OSC_INT_HIGH** or **CG_WARM_UP_SRC_OSC_EXT_HIGH**, Time value range is 0U to 0x1000U.

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) = 50000cycle = 0xC350.
Round lower 4 bit off, set 0xC35 to CG0WUHCR<WUPT[11:0]>

Return:
None.

3.2.3.10 CG_StartWarmUp

Start operation of the specified timer for oscillator.

Prototype:
void
CG_StartWarmUp(void)

Parameters:
None

Description:
This function will start the specified warm up timer.

Return:
None

3.2.3.11 CG_GetWarmUpState

Check whether the specified warm up is completed or not.

Prototype:
WorkState
CG_GetWarmUpState(void)

Parameters:
None

Description:
This function will check that whether the specified 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 the specified warm-up operation is finished.
BUSY: means the specified warm-up operation is in progress.

3.2.3.12 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.

3.2.3.13 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.

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

Return:
None

3.2.3.14 CG_GetFoscSrc

Get the source of the high-speed oscillator (fosc).

Prototype:
CG_FoscSrc
CG_GetFoscSrc(void)

Parameters:
None

Description:
This function will get the source of the high-speed oscillator (fosc).
If the value “Reserved” is read from the register, the API will return **CG_FOSC_UNKNOWN**.

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_UNKNOWN: invalid data is read

3.2.3.15 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.

3.2.3.16 CG_SetFcSrc

Set the clock source of fc

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

Parameters:
Source: the source for fc
This parameter can be one of the following values:
- **CG_FC_SRC_FOSC:** fc source will be set to fosc
- **CG_FC_SRC_IHOSC:** fc source will be set to IHOSC

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.

3.2.3.17 CG_GetFcSrc

Get the clock source of fc.

Prototype:
CG_FcSrc
CG_GetFosc(void)

Parameters:
None

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

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

3.2.3.18 CG_SetProtectCtrl

Enable or disable to protect CG registers.

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

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

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

Return:
None

3.2.3.19 CG_SetSTBYReleaseINTSrc

Set the INT source for releasing low power mode.

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

Parameters:
INTSource: select the INT source for releasing standby mode
This parameter can be one of the following values:
- **CG_INT_SRC_0** : INT0
- **CG_INT_SRC_1** : INT1

ActiveState: select the active state for release trigger.
- **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

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

Return:
None

3.2.3.20 CG_GetSTBYReleaseINTState

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

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

Parameters:
INTSource: select the release INT source
This parameter can be one of the following values:

- **CG_INT_SRC_0** : INT0
- **CG_INT_SRC_1** : INT1

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

3.2.3.21 CG_ClearINTReq

Clear the input INT request.

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

Parameters:
INTSource: select the release INT source.
This parameter can be one of the following values:

- **CG_INT_SRC_0** : INT0
- **CG_INT_SRC_1** : INT1

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

Return:
None

3.2.3.22 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 3) means reset from WDT.
DebugReset (Bit 4) means reset from SYSRESETREQ.

3.2.4 Data Structure Description

3.2.4.1 CG_ResetFlag

Data Fields:
uint32_t
All specifies CG reset source.

Bit Fields:
uint32_t
PinReset(Bit0) Reset from RESET pin
uint32_t
Reserved1 (Bit1~bit2) Reserved
uint32_t
WDTReset(Bit3) Reset from WDT
uint32_t
DebugReset(Bit4) Reset from SYSRESETREQ
uint32_t
Reserved2 (Bit5~bit31) Reserved

4 DSADC

4.1 Overview

TMPM311CHDUG contains 4 units of Delta-Sigma Analog/Digital Converter (DSADC).
In the synchronous start function of DSADC, the following table is an assignment of a master unit and slave unit.

Master/slave assignment

Master	Slave
Unit A	Unit B Unit C Unit D

A reference voltage circuit (BGR) used in the DSADC is shared with a temperature sensor and needs to set the control register (TEMPEN) of temperature sensor.

Features

DSADC has the following features:

- Conversion start
  - Conversion can be started by software.
  - Conversion can be started by hardware triggers.
- Conversion modes
  - Single conversion
  - Repeat conversion
- Status flags
  - Conversion result store flag
  - Overrun flag
  - Conversion end flag
  - Conversion flag
- A conversion clock can be divided by ratios below:
  - fc/1, fc/2, fc/4, and fc/8
- Conversion end interrupt output
- Conversion start correct function
- Synchronous start function for multiple units
- Conversion completion signal output

When DSADC is used, provide pin treatments as follows:

- Do not connect VREFINx to a reference voltage.
- Connect AGNDREFx to DVSS level.
- Connect a 1μF capacitor to between VREFINx and AGNDREFx.

When DSADC is not used, below settings are required.

- Adjust AGNDREFx to the DVSS level.

When a temperature sensor is also not used, a reference voltage circuit requires below settings.

- Connect DSRVDD3 and SRVDD to DVDD3.
- Connect DSRVSS to DVSS.

The DSADC drivers API provide a set of functions to configure DSADC module. It includes DSADC conversion clock set, mode set, start set, correct function set, DSADC status read, DSADC result value read and so on.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm311_dsad.c, with \Libraries/TX03_Periph_Driver\inc\tmpm311_dsad.h containing the API definitions for use by applications.

4.2 API Functions

4.2.1 Function List

- void DSADC_SetClk(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***Clk***)
- void DSADC_SWReset(TSB_DSAD_TypeDef * ***DSADCx***)
- void DSADC_Start(TSB_DSAD_TypeDef * ***DSADCx***)
- void DSADC_ChangeMode(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***SyncMode***, uint32_t ***ConvMode***)
- void DSADC_SetHWStartup(TSB_DSAD_TypeDef * ***DSADCx***, FunctionalState ***NewState***)
- void DSADC_SetHWStartupFactor(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***StartupFactor***)
- void DSADC_SetAmplifier(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***Amplifier***)
- void DSADC_SetAnalogInput(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***AnalogInput***)
- uint32_t DSADC_GetConvertResult(TSB_DSAD_TypeDef * ***DSADCx***)
- void DSADC_Init(TSB_DSAD_TypeDef * ***DSADCx***, DSADC_InitTypeDef * ***InitStruct***)
- DSAD_status DSADC_GetStatus(TSB_DSAD_TypeDef * ***DSADCx***)

4.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) ADC setting by DSADC_SetClk(),DSADC_ChangeMode (),DSADC_Init (), DSADC_SetAmplifier () and DSADC_SetAnalogInput().
2) ADC function start by DSADC_Start(),DSADC_SetHWStartup() and DSADC_SetHWStartupFactor().
3) ADC state or data read functions by DSADC_GetConvertResult (), DSADC_GetStatus ().
4) DSADC_SWReset() handle other specified functions.

4.2.3 Function Documentation

4.2.3.1 DSADC_SetClk

Set AD conversion clock.

Prototype:

Void
DSADC_SetClk(TSB_DSAD_TypeDef * ***DSADCx***,uint32_t ***Clk***)

Parameters:

DSADCx: Select the DSADC unit.

This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

Clk: AD conversion clock selection.

This parameter can be one of the following values:

- **DSADC_FC_DIVIDE_LEVEL_1:** fc / 1
- **DSADC_FC_DIVIDE_LEVEL_2:** fc / 4
- **DSADC_FC_DIVIDE_LEVEL_4:** fc / 4
- **DSADC_FC_DIVIDE_LEVEL_8:** fc / 8

Description:
This function will set DSADC prescaler output by ***Clk***.

***Note:**
During the analog to digital conversion, do not call this function to change the conversion clock setting.
Before calling this function, use **DSADC_GetStatus ()** to check DSADC conversion state is not **BUSY**.

Return:
None

4.2.3.2 DSADC_SWReset

Software reset DSADC

Prototype:
void
DSADC_SWReset(TSB_DSAD_TypeDef * ***DSADCx***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

Description:
This function will software reset DSADC.

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

Return:
None

4.2.3.3 DSADC_Start

Start DSADC function.

Prototype:
void
DSADC_Start(TSB_DSAD_TypeDef * ***DSADCx***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

Description:
This function will start DSADC conversion.

***Note1:**
This function should be called after specifying the mode, which is one of the followings:
Single conversion mode
Repeat conversion mode
Please refer to the description of **DSADC_ChangeMode ()** for the details.

***Note2:**
There is timing restrictions in setting DSADC, before starting AD conversion, please refer to part “Start Sequence” in chapter DSADC in datasheet.

Return:
None

4.2.3.4 DSADC_ChangeMode

Change DSADC synchronous mode and conversion mode.

Prototype:
void
DSADC_ChangeMode(TSB_DSAD_TypeDef * ***DSADCx***,uint32_t ***SyncMode***,uint32_t ***ConvMode***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

SyncMode: Select the DSADC Synchronous mode.
This parameter can be one of the following values:

- **DSADC_A_SYNC_MODE:** Asynchronous operation
- **DSADC_SYNC_MODE:** Synchronous operation

ConvMode: Select the ConvMode mode.
This parameter can be one of the following values:

- **DSADC_SINGLE_MODE:** Single conversion
- **DSADC_REPEAT_MODE:** Repeat conversion

Description:
This function will change DSADC synchronous mode and conversion mode.

Return:
None

4.2.3.5 DSADC_SetHWStartup

Enable or disable hardware startup.

Prototype:
void
DSADC_SetHWStartup(TSB_DSAD_TypeDef * ***DSADCx***, FunctionalState ***NewState***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

➢ **TSB_DSADA:**	DSADC module unit A
➢ **TSB_DSADB:**	DSADC module unit B
➢ **TSB_DSADC:**	DSADC module unit C
➢ **TSB_DSADD:**	DSADC module unit D

NewState: Hardware startup is enabled or disabled.
This parameter can be one of the following values:

- ➢ **ENABLE:** Enable hardware startup
- ➢ **DISABLE:** Disable hardware startup

Description:
This function will enable or disable hardware startup.

Return:
None

4.2.3.6 DSADC_SetHWStartupFactor

Specify the hardware startup factor.

Prototype:
void
void DSADC_SetHWStartupFactor(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***StartupFactor***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

➢ **TSB_DSADA:**	DSADC module unit A
➢ **TSB_DSADB:**	DSADC module unit B
➢ **TSB_DSADC:**	DSADC module unit C
➢ **TSB_DSADD:**	DSADC module unit D

StartupFactor: Hardware startup factor.
This parameter can be one of the following values:

- ➢ **DSADC_HARDWARE_TRIGGER_EXT:** External trigger.
- ➢ **DSADC_HARDWARE_TRIGGER_INT:** Internal trigger.

Description:
This function will specify the hardware startup factor.

Return:
None

4.2.3.7 DSADC_SetAmplifier

Set DSADC Amplifier.

Prototype:
void
DSADC_SetAmplifier(TSB_DSAD_TypeDef * ***DSADCx***,uint32_t ***Amplifier***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

Gain: Amplifier gain setting for the specified Channel.
This parameter can be one of the following values:

- **DSADC_GAIN_1x:** Amplifier gain is 1
- **DSADC_GAIN_2x:** Amplifier gain is 2
- **DSADC_GAIN_4x:** Amplifier gain is 4
- **DSADC_GAIN_8x:** Amplifier gain is 8
- **DSADC_GAIN_16x:** Amplifier gain is 16

Description:
Set gains for the specified channel of DSADC, the input range will become 1/Gain.

Return:
None

4.2.3.8 DSADC_SetAnalogInput

Set the analog inputs.

Prototype:
void
DSADC_SetAnalogInput(TSB_DSAD_TypeDef * ***DSADCx***, uint32_t ***AnalogInput***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

AnalogInput: The analog inputs.
This parameter can be one of the following values:
For unit A, unit B and unit C:

- **DSADC_ANALOG_INPUT_DAIN:** The analog inputs are DAINx (+/-).

For unit D:

- **DSADC_ANALOG_INPUT_DAIN:** The analog inputs are DAINx (+/-).
- **DSADC_ANALOG_INPUT_INT:** The analog inputs are internal analog input (+/-)

Description:
This function will set the analog inputs.

Return:
None

4.2.3.9 DSADC_GetConvertResult

Get DSADC convert result.

Prototype:
uint32_t
DSADC_GetConvertResult(TSB_DSAD_TypeDef * ***DSADCx***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

Description:
This function will read DSADC register's result storage flag state, overrun state, and result value by ***DSADRES*** setting.

Return:
Result

4.2.3.10 DSADC_Init

Initialize the specified DSADC unit.

Prototype:
void
DSADC_Init(TSB_DSAD_TypeDef * ***DSADCx***,DSADC_InitTypeDef * ***InitStruct***)

Parameters:
DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

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

Description:
This function will initialize the specified DSADC unit.

Return:
None

4.2.3.11 DSADC_GetStatus

Indicate DSADC Convertor status and result.

Prototype:

DSAD_status
DSADC_GetStatus(TSB_DSAD_TypeDef * ***DSADCx***)

Parameters:

DSADCx: Select the DSADC unit.
This parameter can be one of the following values:

- **TSB_DSADA:** DSADC module unit A
- **TSB_DSADB:** DSADC module unit B
- **TSB_DSADC:** DSADC module unit C
- **TSB_DSADD:** DSADC module unit D

Description:

This function will read AD conversion busy/completion flag and start or not flag. This function is used to check whether AD conversion has completed or not and started or not.

Return:

A union with the state of AD conversion:

retval. F_ResultStore (Bit 0): ‘1’ means AD conversion result is stored.
retval.F_Overrun (Bit 1): ‘1’ means AD is Overrunning.
retval. F_Convert (Bit 2): ‘1’ means top-priority AD is converting.
retval. F_ConvertEnd (Bit 3): 1’ means normal AD conversion is complete.
retval. ConversionResult (Bit 8 to 31): Conversion result is stored in two’s complement format.

4.2.4 Data Structure Description

4.2.4.1DSADC_InitTypeDef

Bit Fields:

uint32_t
Clk Select the AD conversion clock, which can be set as:

- **DSADC_FC_DIVIDE_LEVEL_1:** fc / 1
- **DSADC_FC_DIVIDE_LEVEL_2:** fc / 4
- **DSADC_FC_DIVIDE_LEVEL_4:** fc / 4
- **DSADC_FC_DIVIDE_LEVEL_8:** fc / 8

uint32_t
BiasE Set the bias control, which can be set as:

- **0:** Stop bias control
- **1:** Bias control operation

uint32_t
ModulatorEn Set the modulator control, which can be set as:

- **0:** Stop modulator control
- **1:** Start modulator control

uint32_t
HardwareFactor Set hardware startup factor, which can be set as:

- **DSADC_HARDWARE_TRIGGER_EXT:** External trigger.

- **DSADC_HARDWARE_TRIGGER_INT:** Internal trigger.

FunctionalState
HardwareEn Enable or disable hardware startup, which can be set as:

- **ENABLE:** Enable hardware startup
- **DISABLE:** Disable hardware startup

uint32_t
SyncMode Select the DSADC synchronous mode, which can be set as:

- **DSADC_A_SYNC_MODE:** Asynchronous operation
- **DSADC_SYNC_MODE:** Synchronous operation

uint32_t
Repeatmode Select the ConvMode, which can be set as:

- **DSADC_SINGLE_MODE:** Single conversion
- **DSADC_REPEAT_MODE:** Repeat conversion

uint32_t
Amplifier Set DSADC Amplifier, which can be set as:

- **DSADC_GAIN_1x:** Amplifier gain is 1
- **DSADC_GAIN_2x:** Amplifier gain is 2
- **DSADC_GAIN_4x:** Amplifier gain is 4
- **DSADC_GAIN_8x:** Amplifier gain is 8
- **DSADC_GAIN_16x:** Amplifier gain is 16

uint32_t
AnalogInput The analog inputs, which can be set as:
For unit A, unit B and unit C:

- **DSADC_ANALOG_INPUT_DAIN:** The analog inputs are DAINx (+/-).

For unit D:

- **DSADC_ANALOG_INPUT_DAIN:** The analog inputs are DAINx (+/-).
- **DSADC_ANALOG_INPUT_INT:** The analog inputs are internal analog input (+/-)

uint16_t
Offset (Bit 7) Set conversion start correction time (OFFSET)

uint32_t
CorrectEn Correct the start of conversion, which can be set as:

- **0:** No correction
- **1:** Correction

4.2.4.2 DSAD_status

Data Fields for this union:

uint32_t
All specifies AD conversion status and result.

uint32_t
F_ResultStore (Bit 0) Conversion result store flag.

Bit Fields:
uint32_t
F_Overrun (Bit 1) Overrun flag.

uint32_t

F_Convert (Bit 2) Conversion flag.

uint32_t
F_ConvertEnd (Bit 3) Conversion end flag.

uint32_t
Reserved (Bit4 to Bit7) reserved.

uint32_t
ConversionResult (Bit8 to Bit31) Conversion result.

5 GPIO

5.1 Overview

TOSHIBA TMPM311CHDUG has 4 general-purpose ports (A, B, C, D), for these 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, CMOS and so on.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/ tmpm311_gpio.c, with /Libraries/TX03_Periph_Driver/inc/tmpm311_gpio.h containing the macros, data types, structures and API definitions for use by applications.

5.2 API Functions

5.2.1 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_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***)

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

5.2.3 Function Documentation

5.2.3.1 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.

Description:
This function will read GPIO Data register.

Return:
The value read from DATA register.

5.2.3.2 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.

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.

5.2.3.3 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.

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

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

Return:
None

5.2.3.4 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.

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

5.2.3.5 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.

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 CMOS port. **GPIO_SetOutput()**, **GPIO_SetInput()**,**GPIO_SetPullUp ()** and **GPIO_SetPullDown()**will be called by it.

Return:
None

5.2.3.6 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.

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

5.2.3.7 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.

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

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

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

Return:
None

5.2.3.8 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.

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

5.2.3.9 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.

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

5.2.3.10 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.

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

5.2.3.11 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_3:** GPIO pin 3,
- **GPIO_BIT_ALL:** GPIO pin[0:7],
- Combination of the effective bits.

NewState:

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

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

Return:
None

5.2.3.12 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.

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

- **GPIO_FUNC_REG_1** for GPIO function register 1,

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

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

Description:
This function will enable GPIO pin specified function.

Return:
None

5.2.3.13 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.

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

- **GPIO_FUNC_REG_1** for GPIO function register 1,

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

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

- **GPIO_BIT_5:** GPIO pin 5,
- **GPIO_BIT_6:** GPIO pin 6,
- **GPIO_BIT_ALL:** GPIO pin[0:7],
- Combination of the effective bits.

Description:
This function will disable GPIO pin specified function.

Return:
None

5.2.4 Data Structure Description

5.2.4.1 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
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.

5.2.4.2 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
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**.**

5.2.4.3 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
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

6 SSP

6.1 Overview

TOSHIBA TMPM311CHDUG contains SSP (Synchronous Serial Port) module with 1 channel (SSP0).

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/TX03_Periph_Driver/src/tmpm311_ssp.c, with /Libraries/TX03_Periph_Driver/inc/tmpm311_ssp.h containing the macros, data types, structures and API definitions for use by applications.

6.2 API Functions

6.2.1 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***);

6.2.2 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() and SSP_SetLoopBackMode() handle other specified functions.

6.2.3 Function Documentation

Note**: in all of the following APIs, parameter “TSB_SSP_TypeDef* ***SSPx” can be one of the following values: **SSP0**

6.2.3.1 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

6.2.3.2 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

6.2.3.3 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

6.2.3.4 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

6.2.3.5 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

6.2.3.6 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

6.2.3.7 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

6.2.3.8 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

6.2.3.9 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

6.2.3.10 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

6.2.3.11 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

6.2.3.12 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

6.2.3.13 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

6.2.3.14 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.

6.2.3.15 SSP_GetFIFOState

Get the Busy or Idle 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

6.2.3.16 SSP_SetINTConfig

Set the data to be sent into Tx FIFO 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

6.2.3.17 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.

6.2.3.18 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.

6.2.3.19 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.

6.2.3.20 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

6.2.4 Data Structure Description

6.2.4.1 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.

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.

6.2.4.2 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: 28Reserved.

7 TEMP

7.1 Overview

The MCU measures a relative temperature using a temperature sensor.
A temperature sensor outputs a voltage based on the reference voltage circuit (BGR) according to temperatures. The output voltage is input to Channel 2 in the analog/digital converter (ADC). With AD conversion, a corresponding digital value to temperatures is obtained.

A difference among the temperature sensor output voltages is linearity related to temperature changes. To obtain a relative temperature, collect data under several conditions.
Channel 3 of ADC is input a 1V from BGR. In variable power voltage system, power voltage can be relatively identified by the result where BGR voltage was performed AD conversion.

If the temperature sensor or DSADC is not used, the reference voltage circuit requires below settings.

- Connect DSRVDD3 and SRVDD to DVDD3
- Connect DSRVSS to DGND

The TEMP drivers API provide a set of functions to configure TEMP, including such parameters as AMP operation, reference voltage circuit, temperature sensor operation and so on.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm311_ temp.c, with \Libraries/TX03_Periph_Driver\inc\tmpm311_temp.h containing the API definitions for use by applications.

***Note**: The reference voltage circuit (BGR) is shared with a ΔΣ type analog/digital converter (DSADC).

7.2 API Functions

7.2.1 Function List

- void TEMP_SetAMPState(FunctionalState ***NewState***)
- FunctionalState TEMP_GetAMPState(void)
- void TEMP_SetBGRState(FunctionalState ***NewState***)
- FunctionalState TEMP_GetBGRState(void)
- void TEMP_SetSensorState(FunctionalState ***NewState***)
- FunctionalState TEMP_GetSensorState(void)

7.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) The Temperature Sensor basic operation is handled by the TEMP_SetAMPState(), TEMP_SetBGRState() and TEMP_SetSensorState() functions.
2) TEMP operation state is get by the TEMP_GetAMPState(), TEMP_GetBGRState() and TEMP_GetSensorState() functions..

7.2.3 Function Documentation

7.2.3.1 TEMP_SetAMPState

Enable or disable AMP operation for ΔΣADC.

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

Parameters:
NewState: Specify the AMP operation.
This parameter can be one of the following values:

- **ENABLE**: Enable AMP operation.
- **DISABLE**: Disable AMP operation.

Description:
This function will enable or disable AMP operation for ΔΣADC.

***Note:**
The AMP operation must be enabled when BGR operation is enabled.

Return:
None

7.2.3.2 TEMP_GetAMPState

Get AMP operation state for ΔΣADC.

Prototype:
FunctionalState
TEMP_GetAMPState(void)

Parameters:
None

Description:
This function will get AMP operation state for ΔΣADC.

Return:
The AMP operation state:
ENABLE: AMP operation is being enabled.
DISABLE: AMP operation is being disabled.

7.2.3.3 TEMP_SetBGRState

Enable or disable the reference voltage circuit.

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

Parameters:
NewState: Specify the reference voltage circuit.
This parameter can be one of the following values:

- **ENABLE**: Enable the reference voltage circuit.
- **DISABLE**: Disable the reference voltage circuit.

Description:
This function will enable or disable the reference voltage circuit.

Return:
None

7.2.3.4 TEMP_GetBGRState

Get the reference voltage circuit state.

Prototype:
FunctionalState
TEMP_GetBGRState(void)

Parameters:
None

Description:
This function will get the reference voltage circuit state.

Return:
The reference voltage circuit state:
ENABLE: Reference voltage circuit is being enabled.
DISABLE: Reference voltage circuit is being disabled.

7.2.3.5 TEMP_SetSensorState

Enable or disable temperature sensor operation.

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

Parameters:
NewState: Specify the temperature sensor operation.
This parameter can be one of the following values:

- **ENABLE**: Enable the temperature sensor operation.
- **DISABLE**: Disable the temperature sensor operation.

Description:
This function will enable or disable temperature sensor operation.

Return:
None

7.2.3.6 TEMP_GetSensorState

Get the temperature sensor operation state.

Prototype:
FunctionalState

TEMP_GetSensorState(void)

Parameters:
None

Description:
This function will get the temperature sensor operation state.

Return:
The temperature sensor operation state:
ENABLE: The temperature sensor operation is being enabled.
DISABLE: The temperature sensor operation is being disabled.

7.2.4 Data Structure Description

None

8 TMR16A

8.1 Overview

TOSHIBA TMPM311CHDUG has 1 channel of TMR16A and does not provide output signal to the rectangular wave pin. TMR16A contains the following functions:

- Match interrupt
- Square waveform output
- Read capture.

The TMR16A drivers API provide a set of functions to configure TMR16A module. It includes setting of start, setting of clock, configure of flip-flop, setting of cycle, capture and so on.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm311_tmr16a.c, with /Libraries/TX03_Periph_Driver/inc/tmpm311_tmr16a.h containing the macros, data types, structures and API definitions for use by applications.

8.2 API Functions

8.2.1 Function List

- void TMR16A_SetIdleMode(TSB_T16A_TypeDef * ***T16Ax***);
- void TMR16A_SetClkInCoreHalt(TSB_T16A_TypeDef * ***T16Ax***, uint8_t ***ClkState***);
- void TMR16A_SetRunState(TSB_T16A_TypeDef * ***T16Ax***, uint32_t ***Cmd***);
- void TMR16A_SetSrcClk(TSB_T16A_TypeDef * ***T16Ax***, uint32_t ***SrcClk***);
- void TMR16A_SetFlipFlop(TSB_T16A_TypeDef * ***T16Ax***, TMR16A_FFOutputTypeDef * ***FFStruct***);
- void TMR16A_ChangeCycle(TSB_T16A_TypeDef * ***T16Ax***, uint32_t ***Cycle***);
- uint16_t TMR16A_GetCaptureValue(TSB_T16A_TypeDef* ***T16Ax***);

8.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) Configure and control the common functions of each TMR16A channel are handled by TMR16A_SetSrcClk(),TMR16A_SetRunState()and TMR16A_ChangeCycle().
2) The status indication of each TMR16A channel is handled by TMR16A_GetCaptureValue().
3) TMR16A_SetFlipFlop(),TMR16A_SetClkInCoreHalt (),TMR16A_SetIdleMode() handle other specified functions.

8.2.3 Function Documentation

Note**: In all of the following APIs, unless it is specified, the parameter: “TSB_T16A_TypeDef * ***T16Ax” can be one of the following values:
TSB_T16A0.

8.2.3.1 TMR16A_SetIdleMode

Enable or disable the specified TMR16A channel when system is in idle mode.

Prototype:
void

TMR16A _SetIdleMode(TSB_ T16A _TypeDef* ***T16Ax***)
Parameters:
T16Ax is the specified TMR16A channel.

Description:
None
Return:
None

8.2.3.2 TMR16A_SetClkInCoreHalt

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

Prototype:
void
TMR16A_SetClkInCoreHalt (TSB_T16A_TypeDef* ***T16Ax***,
uint8_t ***ClkState***)

Parameters:
T16Ax is the specified TMR16A channel.
ClkState specifies timer state in HALT mode, which can be

- **TMR16A_RUNNING_IN_CORE_HALT:** clock not stops in Core HALT
- **TMR16A_STOP_IN_CORE_HALT:** clock stops in Core HALT.

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

Return:
None

8.2.3.3 TMR16A_SetRunState

Start or stop counter of the specified T16A channel.

Prototype:
void
TMR16A_SetRunState(TSB_T16A_TypeDef* ***T16Ax***,
uint32_t ***Cmd***)

Parameters:
T16Ax is the specified TMR16A channel.
Cmd sets the state of up-counter, which can be:

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

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

Return:
None

8.2.3.4 TMR16A_SetSrcClk

Specifies a source clock.
Prototype:
void
TMR16A_SetSrcClk(TSB_T16A_TypeDef* ***T16Ax***,
uint32_t ***SrcClk***)

Parameters:
T16Ax is the specified TMR16A channel.
SrcClk specifies the state of the TMR16A source clock, which can be

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

Description:
This function can select TMR16A channel's source clock.

Return:
None

8.2.3.5 TMR16A_SetFlipFlop

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

Prototype:
void
TMR16A_SetFlipFlop(TSB_T16A_TypeDef* ***T16Ax***,
TMR16A_FFOutputTypeDef* ***FFStruct***)

Parameters:
T16Ax is the specified TMR16A channel.
FFStruct is the structure containing TMR16A flip-flop function configuration including flip-flop output level and flip-flop-reverse trigger (refer to "Data Structure Description" for details).

Description:
This function will set the timing of changing the flip-flop output of the specified TMR16A channel. Also the level of the output can be controlled by this API.

Return:
None

8.2.3.6 TMR16A_ChangeCycle

Change the value of cycle for the specified channel.

Prototype:
void
TMR16A_ChangeCycle(TSB_T16A_TypeDef* ***T16Ax***,
uint32_t ***Cycle***)

Parameters:
T16Ax is the specified TMR16A channel.
Cycle specifies the value of cycle, max is 0xFFFF.

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

Return:
None

8.2.3.7 TMR16A_GetCaptureValue

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

Prototype:
uint16_t
TMR16A_GetCaptureValue(TSB_T16A_TypeDef* ***T16Ax***)

Parameters:
T16Ax is the specified TMR16A channel.

Description:
This function will return the value of capture register of the specified TMR16A channel.
Return:
The captured value.

8.2.4 Data Structure Description

8.2.4.1 TMR16A_FFOutputTypeDef

Data Fields:

uint32_t
TMR16AFlipflopCtrl selects the level of flip-flop output which can be
1) **TMR16A_FLIPFLOP_INVERT:** setting output reversed by using software.
2) **TMR16A_FLIPFLOP_SET:** setting output to be high level.
3) **TMR16A_FLIPFLOP_CLEAR:** setting output to be low level.

uint32_t
TMR16AFlipflopReverseTrg specifies the reverse trigger of the flip-flop output, which can be set as:
1) **TMR16A_DISALBE_FLIPFLOP**, which disables the flip-flop output reverse trigger.
2) **TMR16A_FLIPFLOP_MATCH_CYCLE**, which means that the reversing flip-flop output will be triggered when the up-counter matches the cycle.

9 TMRB

9.1 Overview

TOSHIBA TMPM311CHDUG has 4 channels (TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3) of multi-functional 16-bit timer/event counter (TMRB0 through TMRB3). 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 measurements:

- Frequency measurement
- Pulse width measurement

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

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm311_tmrb.c, with /Libraries/TX03_Periph_Driver/inc/tmpm311_tmrb.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 Function List

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

9.2.2 Detailed Description

Functions listed above can be divided into four parts:

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

9.2.3 Function Documentation

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.

9.2.3.1 TMRB_Enable

Enable the specified TMRB channel.

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

Parameters:
TBx is the specified TMRB channel.

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

Return:
None

9.2.3.2 TMRB_Disable

Disable the specified TMRB channel.

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

Parameters:
TBx is the specified TMRB channel.

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

Return:

None

9.2.3.3 TMRB_SetRunState

Start or stop counter of the specified TB channel.

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

Parameters:
TBx is the specified TMRB channel.
Cmd sets the state of up-counter, which can be:

- **TMRB_RUN:** starting counting
- **TMRB_STOP:** stopping counting

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

Return:
None

9.2.3.4 TMRB_Init

Initialize the specified TMRB channel.

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

Parameters:
TBx is the specified TMRB channel.
InitStruct is the structure containing basic TMRB configuration including count mode, source clock division, 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

9.2.3.5 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_TB0 to TSB_TB3:

- **TMRB_DISABLE_CAPTURE**: Capture is disabled.
- **TMRB_CAPTURE_TBIN_RISING_FALLING**: Captures a counter value on rising edge of TBxIN input into Capture register 0 (TBxCP0).Captures a counter value on falling edge of TBxIN 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).

Description:
This function will configure the capture timing and up-counter clearing timing.

Return:
None

9.2.3.6 TMRB_SetFlipFlop

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

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

Parameters:
TBx is the specified TMRB channel.

FFStruct is the structure containing TMRB flip-flop function configuration including flip-flop output level and flip-flop-reverse trigger (refer to “Data Structure Description” for details).

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

Return:
None

9.2.3.7 TMRB_GetINTFactor

Indicate what causes the interrupt.

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

Parameters:

TBx is the specified TMRB channel.

Description:
This function should be used in ISR to indicate the factor of interrupt. Bit of **MatchLeadingTiming** indicates if the up-counter matches with 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 factors. 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
 }
```

9.2.3.8 TMRB_SetINTMask

Mask the specified TMRB interrupt.

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

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

- **TMRB_MASK_MATCH_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

9.2.3.9 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***.

9.2.3.10 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.

9.2.3.11 TMRB_GetUpCntValue

Get up-counter value of the specified TMRB channel.

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

Parameters:
TBx is the specified TMRB channel.

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

Return:
The value of up-counter

9.2.3.12 TMRB_GetCaptureValue

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

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

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

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

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

Return:
The captured value

9.2.3.13 TMRB_ExecuteSWCapture

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

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

Parameters:
TBx is the specified TMRB channel.

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

Return:
None

9.2.3.14 TMRB_SetIdleMode

Disable the specified TMRB channel when system is in idle mode.

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

Parameters:
TBx is the specified TMRB channel.

Description:
The specified TMRB channel can stop the running TMRB if system enters idle mode.

Return:
None

***Note:**
Low-power consumption mode operation function of the TMRB has no meaning. Write "0" to the <I2TB>.

9.2.3.15 TMRB_SetSyncMode

Enable or disable the synchronous mode of specified TMRB channel.

Prototype:

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

Parameters:

TBx is the specified TMRB channel, which can be
TSB_TB1, TSB_TB2,TSB_TB3.

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.

Return:

None

9.2.3.16 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***)

Parameters:

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

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

Description:

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

9.2.3.17 TMRB_SetExtStartTrg

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

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

Parameters:
TBx is the specified TMRB channel.
NewState specifies the state external trigger, which can be

- **ENABLE:** use external trigger signal,
- **DISABLE:** use software start.

TrgMode specifies active edge of the external trigger signal. which can be

- **TMRB_TRG_EDGE_RISING:** Select rising edge of external trigger.
- **TMRB_TRG_EDGE_FALLING:** Select falling edge of external trigger.

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

Return:
None

9.2.3.18 TMRB_SetClkInCoreHalt

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

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

Parameters:
TBx is the specified TMRB channel.
ClkState specifies timer state in HALT mode, which can be

- **TMRB_RUNNING_IN_CORE_HALT:** clock not stops in Core HALT
- **TMRB_STOP_IN_CORE_HALT:** clock stops in Core HALT.

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

Return:
None

9.2.4 Data Structure Description

9.2.4.1 TMRB_InitTypeDef

Data Fields:

uint32_t

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

uint32_t
ClkDiv specifies the division of the source clock for the internal interval timer, which can be set as:

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

uint32_t
TrailingTiming specifies the 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.
- **TMRB_AUTO_CLEAR**, which means that the up-counter will restart counting from 0 immediately when the value in up-counter matches **TrailingTiming**.

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

9.2.4.2 TMRB_FFOutputTypeDef

Data Fields:

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

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

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

- **TMRB_DISALBE_FLIPFLOP**, which disables the flip-flop output reverse trigger,
- **TMRB_FLIPFLOP_TAKE_CATPURE_0**, which means that the reversing flip-flop output will be triggered when the up-counter value is taken into capture register 0,

- **TMRB_FLIPFLOP_TAKE_CATPURE_1**, which means that the reversing flip-flop output will be triggered when the up-counter value is taken into capture register 1,
- **TMRB_FLIPFLOP_MATCH_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.

9.2.4.3 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 -

10 UART

10.1 Overview

TOSHIBA TMPM311CHDUG contains 1 serial I/O channel (UART0). 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/TX03_Periph_Driver/src/tmpm311_uart.c, with /Libraries/TX03_Periph_Driver/inc/tmpm311_uart.h containing the macros, data types, structures and API definitions for use by applications.

10.2 API Functions

10.2.1 Function List

- void UART_Enable(TSB_SC_TypeDef* ***UARTx***)
- void UART_Disable(TSB_SC_TypeDef* ***UARTx***)
- WorkState UART_GetBufState(TSB_SC_TypeDef* ***UARTx***, uint8_t ***Direction***)
- void UART_SWReset(TSB_SC_TypeDef* ***UARTx***)
- void UART_Init(TSB_SC_TypeDef* ***UARTx***, UART_InitTypeDef* ***InitStruct***)
- uint32_t UART_GetRxData(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetTxData(TSB_SC_TypeDef* ***UARTx***, uint32_t ***Data***)
- void UART_DefaultConfig(TSB_SC_TypeDef* ***UARTx***)
- UART_Err UART_GetErrState(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetWakeUpFunc(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetIdleMode(TSB_SC_TypeDef* ***UARTx***)
- 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***)

10.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Initialize and configure the common functions of each UART channel are handled by UART_Enable(), UART_Disable(),UART_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(),

10.2.3 Function Documentation

***Note**: In all of the following APIs:

Parameter “TSB_SC_TypeDef* ***UARTx***” can be one of
the following values:
UART0.

Parameter “TSB_SC_TypeDef* ***SIOx***” can be one of the following values:
SIO0.

10.2.3.1 UART_Enable

Enable the specified UART channel.

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

Parameters:
UARTx is the specified UART channel.

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

Return:
None

10.2.3.2 UART_Disable

Disable the specified UART channel.

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

Parameters:
UARTx is the specified UART channel.

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

Return:
None

10.2.3.3 UART_GetBufState

Indicate the state of transmission or reception buffer.

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

Parameters:
UARTx is the specified UART channel.
Direction select the direction of transfer, which can be one of:

- **UART_RX** for reception
- **UART_TX** for transmission

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

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

10.2.3.4 UART_SWReset

Reset the specified UART channel.

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

Parameters:
UARTx is the specified UART channel.

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

Return:
None

10.2.3.5 UART_Init

Initialize and configure the specified UART channel.

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

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

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

Return:
None

10.2.3.6 UART_GetRxData

Get data received from the specified UART channel.

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

Parameters:
UARTx is the specified UART channel.

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

Return:
Data which has been received

10.2.3.7 UART_SetTxData

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

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

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

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

Return:
None

10.2.3.8 UART_DefaultConfig

Initialize the specified UART channel in the default configuration.

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

Parameters:
UARTx is the specified UART channel.

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

Return:
None

10.2.3.9 UART_GetErrState

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

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

Parameters:
UARTx is the specified UART channel.

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

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

10.2.3.10 UART_SetWakeUpFunc

Disable wake-up function in 9-bit mode of the specified UART channel.

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

Parameters:
UARTx is the specified UART channel.

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

Return:
None

10.2.3.11 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

10.2.3.12 UART_SetIdleMode

Disable the specified UART channel when system is in idle mode.

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

Parameters:
UARTx is the specified UART channel.

Description:
This function will disable the specified UART channel selected by ***UARTx*** in system idle mode

Return:
None

10.2.3.13 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

10.2.3.14 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

10.2.3.15 UART_TRxAutoDisable

Control 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:
Control automatic disabling of transmission and reception.

Return:
None

10.2.3.16 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

10.2.3.17 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

10.2.3.18 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

10.2.3.19 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

10.2.3.20 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

10.2.3.21 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

10.2.3.22 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

10.2.3.23 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

10.2.3.24 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

10.2.3.25 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

10.2.3.26 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.

10.2.3.27 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.

10.2.3.28 UART_GetTxFIFOFillLevelStatus

Status of transmit FIFO fill level.

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

Parameters:
UARTx is the specified UART channel.

Description:
Status of transmit FIFO fill level.

Return:
UART_TRXFIFO_EMPTY: TX FIFO fill level is empty.
UART_TRXFIFO_1B: TX FIFO fill level is 1 byte.
UART_TRXFIFO_2B: TX FIFO fill level is 2 bytes.
UART_TRXFIFO_3B: TX FIFO fill level is 3 bytes.
UART_TRXFIFO_4B: TX FIFO fill level is 4 bytes.

10.2.3.29 UART_GetTxFIFOUnderRunStatus

Transmit FIFO under run

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

Parameters:
UARTx is the specified UART channel.

Description:
Transmit FIFO under run

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

10.2.3.30 SIO_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

10.2.3.31 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

10.2.3.32 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

10.2.3.33 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

10.2.3.34 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

10.2.3.35 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

10.2.4 Data Structure Description

10.2.4.1 UART_InitTypeDef

Data Fields:

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

uint32_t
DataBits specifies data bits per transfer, which can be set as:

- **UART_DATA_BITS_7** for 7-bit mode
- **UART_DATA_BITS_8** for 8-bit mode

- **UART_DATA_BITS_9** for 9-bit mode

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

- **UART_STOP_BITS_1** for 1 stop bit
- **UART_STOP_BITS_2** for 2 stop bits

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

- **UART_NO_PARITY** for no parity
- **UART_EVEN_PARITY** for even parity
- **UART_ODD_PARITY** for odd parity

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

- **UART_ENABLE_TX** for enabling transmission
- **UART_ENABLE_RX** for enabling reception

uint32_t
FlowCtrl specifies whether the hardware flow control mode is enabled or disabled (**). It can be set as:

- **UART_NONE_FLOW_CTRL** for no flow control

10.2.4.2 SIO_InitTypeDef

Data Fields:

uint32_t
InputClkEdge Select the input clock edge, which can be set as:

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

uint32_t
TIDLE The status of TXDx pin after output of the last bit, which can be set as:

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

uint32_t
TXDEMP The status of TXDx pin when an under run error is occured in SCLK input mode, which can be set as:

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

uint32_t
EHOLDTime The last bit hold time of TXDx pin in SCLK input mode, which can be set as:

- **SIO_EHOLD_FC_2** Set a last bit hold time is 2/fc.
- **SIO_EHOLD_FC_4** Set a last bit hold time is 4/fc.
- **SIO_EHOLD_FC_8** Set a last bit hold time is 8/fc.
- **SIO_EHOLD_FC_16** Set a last bit hold time is 16/fc.
- **SIO_EHOLD_FC_32** Set a last bit hold time is 32/fc.
- **SIO_EHOLD_FC_64** Set a last bit hold time is 64/fc.
- **SIO_EHOLD_FC_128** Set a last bit hold time is 128/fc.

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

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

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

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

uint32_t
TransferDir Setting transfer mode, which can be set as:

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

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

- **UART_ENABLE_TX** for enabling transmission.
- **UART_ENABLE_RX** for enabling reception.

uint32_t
DoubleBuffer Double Buffer mode, which can be set as:

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

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

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

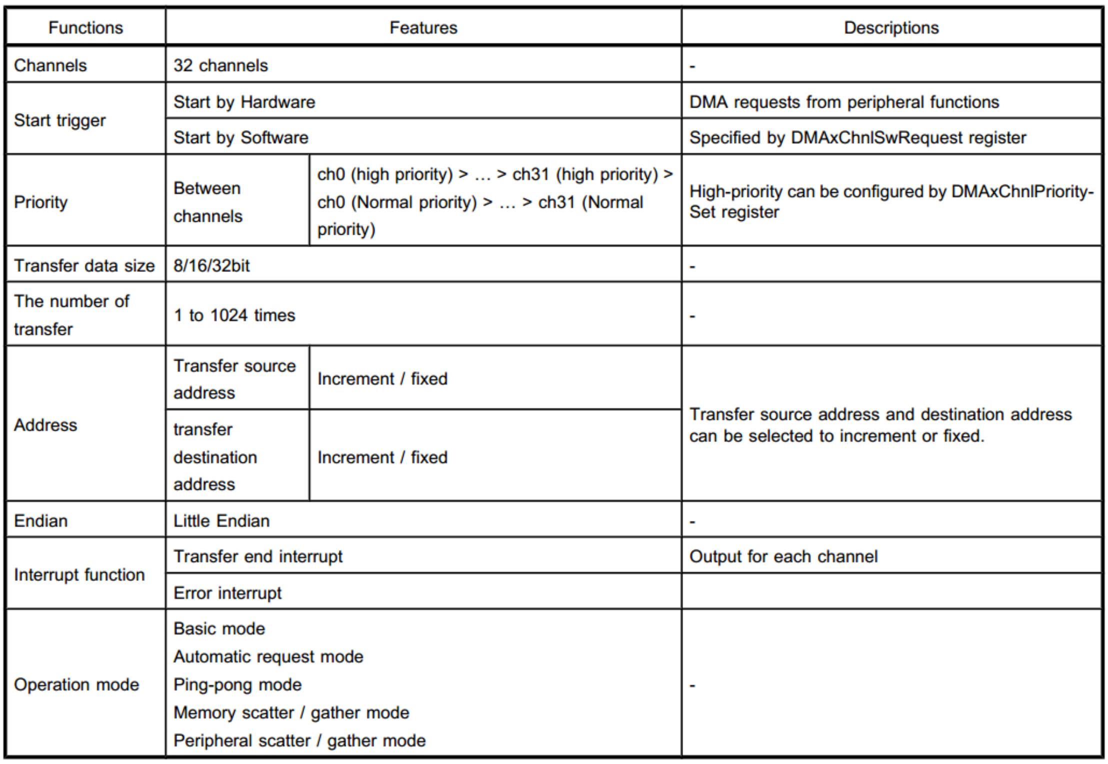
uint32_t
Divider Division ratio "N", which can be set as :

- **SIO_BR_DIVIDER_16** Division ratio is 16.
- **SIO_BR_DIVIDER_1** Division ratio is 1.
- **SIO_BR_DIVIDER_2** Division ratio is 2.
- **SIO_BR_DIVIDER_3** Division ratio is 3.
- **SIO_BR_DIVIDER_4** Division ratio is 4.
- **SIO_BR_DIVIDER_5** Division ratio is 5.
- **SIO_BR_DIVIDER_6** Division ratio is 6.
- **SIO_BR_DIVIDER_7** Division ratio is 7.
- **SIO_BR_DIVIDER_8** Division ratio is 8.
- **SIO_BR_DIVIDER_9** Division ratio is 9.
- **SIO_BR_DIVIDER_10** Division ratio is 10.
- **SIO_BR_DIVIDER_11** Division ratio is 11.
- **SIO_BR_DIVIDER_12** Division ratio is 12.
- **SIO_BR_DIVIDER_13** Division ratio is 13.
- **SIO_BR_DIVIDER_14** Division ratio is 14.
- **SIO_BR_DIVIDER_15** Division ratio is 15.

11 uDMAC

11.1 Overview

TMPM311CHDUG contains 1unit of built-in µDMA controller. (Unit A)
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		-
The number of transfer	1 to 1024 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		-
Interrupt function	Transfer end interrupt		Output for each channel
	Error interrupt		
Operation 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 TMPM311 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 TX03_Periph_Driver\src\tmpm311_udmac.c, with TX03_Periph_Driver\inc\tmpm311_udmac.h containing the API definitions for use by applications.

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

11.2 API Functions

11.2.1 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 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***)

11.2.2 Detailed Description

Functions listed above can be divided into five parts:

1) uDMAC configuration by DMACA_SetTransferType(), DMACA_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().

11.2.3 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:***

11.2.3.1

DMACx: Select DMAC unit.
This parameter can be one of the following values:

- **DMAC_UNIT_A :** DMAC unit A

11.2.3.2

Channel: Select channel.
The parameter can be one of the following values:
For DMAC_UNIT_A:

- **DMACA_SSP0_RX :** SSP0 reception
- **DMACA_SSP0_TX :** SSP0 transmission
- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission

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

11.2.3.4 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

11.2.3.5 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

11.2.3.6 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

11.2.3.7 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

11.2.3.8 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

11.2.3.9 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_SSP0_RX :** SSP0 reception
- **DMACA_SSP0_TX :** SSP0 transmission
- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission

When *Type* is DMAC_SINGLE:

- **DMACA_SSP0_RX :** SSP0 reception
- **DMACA_SSP0_TX :** SSP0 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 UNITA.

Return:
None

11.2.3.10 DMACA_GetTransferType

Get the setting of transferring type 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_SSP0_RX :** SSP0 reception
- **DMACA_SSP0_TX :** SSP0 transmission
- **DMACA_UART0_RX :** UART0 reception
- **DMACA_UART0_TX :** UART0 transmission

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

11.2.3.11 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

11.2.3.12 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)

11.2.3.13 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

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

11.2.3.15 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

11.2.3.16 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

11.2.3.17 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

11.2.3.18 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.

11.2.3.19 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

11.2.3.20 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.

11.2.3.21 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

11.2.4 Data Structure Description

11.2.4.1 DMAC_InitTypeDef

Data fields:

uint32_t
SrcEndPointer: The final address of data source.

uint32_t
DstEndPointer: The final address of data destination.

DMAC_CycleCtrl
Mode: Set operation mode,
which can be:

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

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

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

uint32_t
TxNum: Set the actual number of transfers. Maximum is 1024.

DMAC_Arbitration
ArbitrationMoment: Specifies the arbitration moment(R_Power).
After the specified numbers of transfers, an existence of a transfer request is checked. If there is a high-priority request, the control is switched to high-priority channel.

DMAC_BitWidth
SrcWidth: Set source bit width,
which can be:

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

DMAC_IncWidth
SrcInc: Set increment of the source address,
which can be:

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

DMAC_BitWidth
DstWidth: Set destination bit width,
which can be:

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

DMAC_IncWidth
DstInc: Set increment of the destination address,
which can be:

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

12 WDT

12.1 Overview

The watchdog timer (WDT) is for detecting malfunctions (runaways) of the CPU caused by noises or other disturbances and remedying them to return the CPU to normal operation.

The WDT drivers API provide a set of functions to configure WDT, including such parameters as detection time, output if counter overflows, the state of WDT when enter IDLE mode and so on.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm311_ wdt.c, with \Libraries/TX03_Periph_Driver\inc\tmpm311 _wdt.h containing the API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- Result WDT_SetDetectTime(uint32_t ***DetectTime***)
- Result WDT_SetIdleMode(void)
- 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)

12.2.2 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().

12.2.3 Function Documentation

12.2.3.1 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.

12.2.3.2 WDT_SetIdleMode

Stop the WDT counter when the system enters IDLE mode.

Prototype:
Result
WDT_SetIdleMode(void)

Parameters:
None

Description:
This function will stop the WDT counter.

***Note:**
Low-power consumption mode operation function of the WDT has no meaning. Write "0" to the <I2WDT>.

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

12.2.3.3 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.

12.2.3.4 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.

12.2.3.5 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.

12.2.3.6 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.

12.2.3.7 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.

12.2.3.8 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 WD0MOD or WD0CR, 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.

12.2.4 Data Structure Description

12.2.4.1 WDT_InitTypeDef

Data Fields:

uint32_t
DetectTime Set WDT detection time, which can be set as:

- **WDT_DETECT_TIME_EXP_15:** ***DetectTime*** is 2^15/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.