TOSHIBA

TOSHIBA TX03 Peripheral Driver User Guide
(TMPM375FSDMG)

Ver 1
Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

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. TMPM375FSDMG Peripheral Driver is an important part of TOSHIBA TX03 Peripheral Driver, which is designed for TMPM375FSDMG series MCUs.

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

The design goals of TOSHIBA TMPM375FSDMG Peripheral Driver:

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

2. Organization of TOSHIBA TX03 Peripheral Driver

/Libraries
This folder contains all CMSIS files and TMPM375FSDMG Peripheral Drivers.

/Libraries/ TX03_CMSIS
This folder contains the device peripheral access layer of TMPM375FSDMG CMSIS files.

/Libraries/TX03_Periph_Driver
This folder contains all the source code of the drivers, the core of TOSHIBA TMPM375FSDMG Peripheral Driver.

/Libraries/TX03_Periph_Driver/inc
This folder contains all the header files of TMPM375FSDMG Peripheral Drivers for each peripheral.

/Libraries/TX03_Periph_Driver/src
This folder contains all the source files of TMPM375FSDMG Peripheral Drivers for each peripheral.

/Project
This folder contains template project and examples for using TMPM375FSDMG Peripheral Driver.

/Project/Template
This folder contains template project of TOSHIBA TMPM375FSDMG Peripheral Driver.

/Project/Examples
This folder contains a set of examples for using TMPM375FSDMG Peripheral Driver

/Utilities/TMPM375-EVAL
This folder contains the configuration and driver files for hardware resources (e.g. led, key) on TMPM375FSDMG boards.

3. ADC

3.1 Overview

TOSHIBA TMPM375FSDMG contains a 12-bit successive-approximation Analog-to-Digital Converters (ADC).

The ADC unit B (ADC B) has 5 analog inputs. Four inputs are able to use for shunt resistor currents of motor 0. And one input is connected with operational amp output. Thus four inputs can use for external input.

Four external analog input pins (AINB9 to AINB12) can also be used as input/output ports.

Functions and features
(1) It can select analog input and start AD conversion when receiving trigger signal from PMD or TMRB (interrupt).
(2) It can select analog input, in the Software Trigger Program and the Constant Trigger Program.
(3) The ADC has twelve registers for AD conversion result.
(4) The ADC generates an interrupt signal at the end of the program which was started by PMD trigger and TMRB trigger.
(5) The ADC generates an interrupt signal at the end of the program which are the Software Trigger Program and the Constant Trigger Program.
(6) The ADC has the AD conversion monitoring function. When this function is enabled, an interrupt is generated when a conversion result matches the specified comparison value.

The ADC API provides a set of functions for using the TMPM375FSDMG ADC modules. It includes ADC channel set, mode set, monitor function set, interrupt set, ADC status read, ADC result value read and so on.

This driver is contained in TX03_Periph_Driver\src\tmpm375_adc.c, with TX03_Periph_Driver\inc\tmpm375_adc.h containing the API definitions for use by applications.

3.2 API Functions

3.2.1 Function List

- void ADC_SetClk(TSB_AD_TypeDef * ***ADx***, uint32_t ***Sample_HoldTime***,
  ADC_PRESCALER ***Prescaler_Output***);
- void ADC_Enable(TSB_AD_TypeDef * ***ADx***);
- void ADC_Disable(TSB_AD_TypeDef * ***ADx***);
- void ADC_Start(TSB_AD_TypeDef * ***ADx***, ADC_TrgType ***Trg***);
- void ADC_StopConstantTrg(TSB_AD_TypeDef * ***ADx***);
- WorkState ADC_GetConvertState(TSB_AD_TypeDef * ***ADx***, ADC_TrgType ***Trg***);
- void ADC_SetMonitor(TSB_AD_TypeDef * ***ADx***, ADC_MonitorTypeDef * ***Monitor***);
- void ADC_DisableMonitor(TSB_AD_TypeDef * ***ADx***, ADC_CMPCRx ***CMPCRx***);
- ADC_Result ADC_GetConvertResult(TSB_AD_TypeDef * ***ADx***,

ADC_REGx ***Re sultREGx***);

- ◆ void ADC_SelectPMDTrgProgNum(TSB_AD_TypeDef * ***ADx***, PMD_TRG_PROG_SELx ***SELx***, uint8_t ***MacroProgNum***);
- ◆ void ADC_SetPMDTrgProgINT(TSB_AD_TypeDef * ***ADx***, PMD_TrgProgINTTypeDef * ***TrgProgINT***);
- ◆ void ADC_SetPMDTrg(TSB_AD_TypeDef * ***ADx***, PMD_TrgTypeDef * ***PMDTrg***);
- ◆ void ADC_SetTimerTrg(TSB_AD_TypeDef * ***ADx***, ADC_REGx ***ResultREGx***, uint8_t ***MacroAINx***);
- ◆ void ADC_SetSWTrg(TSB_AD_TypeDef * ***ADx***, ADC_REGx ***ResultREGx***, uint8_t ***MacroAINx***);
- ◆ void ADC_SetConstantTrg(TSB_AD_TypeDef * ***ADx***, ADC_REGx ***ResultREGx***, uint8_t ***MacroAINx***);

3.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) ADC setting by ADC_SetClk(), ADC_SetMonitor(), ADC_DisableMonitor(), ADC_SelectPMDTrgProgNum(), ADC_SetPMDTrgProgINT(), ADC_SetPMDTrg(), ADC_SetTimerTrg(), ADC_SetSWTrg(), ADC_SetConstantTrg().
2) ADC function enable/disable & start/stop by ADC_Enable(), ADC_Disable(), ADC_Start(), ADC_StopConstantTrg().
3) ADC state or data read functions by ADC_GetConvertState(), ADC_GetConvertResult().

3.2.3 Function Documentation

3.2.3.1 ADC_SetClk

Set ADC prescaler output(SCLK) of the specified ADC unit.

Prototype:

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

Parameters:

ADx: Select ADC Unit, which can be set as:

- ➢ **TSB_ADB:** ADC Unit B

Sample_HoldTime: Select ADC sample hold time.
which can be set as:
ADC_HOLD_FIX: write “1001b” to TSH<0:3>.

Prescaler_Output: Select ADC prescaler output, which can be set as:

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

Description:

This function will set the specified ADC unit’s sample hold time by ***Sample_HoldTime*** as **ADC_HOLD_FIX** & select ADC prescaler output by ***Prescaler_Output***.

Return:
None

3.2.3.2 ADC_Enable

Enable the specified ADC unit.

Prototype:
void
ADC_Enable(TSB_AD_TypeDef * ***ADx***)

Parameters:
ADx: Select ADC Unit, which can be set as:
- **TSB_ADB:** ADC Unit B

Description:
This function will enable the specified ADC unit.

Return:
None

3.2.3.3 ADC_Disable

Disable the specified ADC unit.

Prototype:
void
ADC_Disable(TSB_AD_TypeDef * ***ADx***)

Parameters:
ADx: Select ADC Unit, which can be set as:
- **TSB_ADB:** ADC Unit B

Description:
This function will disable the specified ADC unit.

Return:
None

3.2.3.4 ADC_Start

Start the specified ADC unit with software trigger or constant trigger.

Prototype:
void
ADC_Start(TSB_AD_TypeDef * ***ADx***,
ADC_TrgType ***Trg***)

Parameters:
ADx: Select ADC Unit, which can be set as:
- **TSB_ADB:** ADC Unit B

Trg**:** Set trigger type, which can be set as:
- **ADC_TRG_SW:** Software triggered conversion
- **ADC_TRG_CONSTANT:** Constant AD conversion

Description:
This function will start the specified ADC unit by ***Trg*** as **ADC_TRG_SW**, **ADC_TRG_CONSTANT**.

Return:
None

3.2.3.5 ADC_StopConstantTrg

Stop the specified ADC unit when use constant trigger.

Prototype:
void
ADC_StopConstantTrg(TSB_AD_TypeDef * ***ADx***)

Parameters:
ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

Description:
This function will stop the specified ADC unit when use constant trigger.

Return:
None

3.2.3.6 ADC_GetConvertState

Get the conversion state of the specified ADC unit.

Prototype:
WorkState
ADC_GetConvertState(TSB_AD_TypeDef * ***ADx***,
ADC_TrgType ***Trg***)

Parameters:
ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

Trg**:** Set trigger type, which can be set as:

- **ADC_TRG_SW:** Software triggered conversion
- **ADC_TRG_CONSTANT:** Constant AD conversion
- **ADC_TRG_TIMER:** Timer triggered conversion
- **ADC_TRG_PMD:** PMD triggered conversion

Description:
This function will get the state of the specified ADC unit’s conversion as **BUSY/DONE**, when AD conversion is triggered set by ***Trg*** as **ADC_TRG_SW**, **ADC_TRG_CONSTANT**, **ADC_TRG_TIMER**, **ADC_TRG_PMD**.

Return:
WorkState type, the value can be:
BUSY: Conversion is in progress
DONE: Conversion is not in process

3.2.3.7 ADC_SetMonitor

Set the monitor function of the specified ADC unit and enable it.

Prototype:

```
void
ADC_SetMonitor(TSB_AD_TypeDef * ADx,
                    ADC_MonitorTypeDef * Monitor)
```

Parameters:

ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

Monitor**:** It is a structure with detail as below:

```
typedef struct {
   ADC_CMPCRx CMPCRx;
   ADC_REGx ResultREGx;
   uint32_t CmpTimes;
   ADC_CmpCondition Condition;
   uint32_t CmpValue;
} ADC_MonitorTypeDef
```

For details of this structure, refer to part “Data Structure Description”.

Description:

This function will set AD conversion result monitoring function of the specified unit by ADC_MonitorTypeDef * ***Monitor*** and enable it.

Return:

None

3.2.3.8 ADC_DisableMonitor

Disable the monitor function of the specified ADC unit.

Prototype:

```
void
ADC_DisableMonitor(TSB_AD_TypeDef * ADx,
                        ADC_CMPCRx CMPCRx)
```

Parameters:

ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

CMPCRx: Select compare control register.
which can be set as:

- **ADC_CMPCR_0:** ADxCMPCR0
- **ADC_CMPCR_1:** ADxCMPCR1

Description:

This function will disable the monitor function of the specified ADC unit by ***CMPCRx*** as **ADC_CMPCR_0** or **ADC_CMPCR_1**.

Return:

None

3.2.3.9 ADC_GetConvertResult

Get result from the specified AD Conversion Result Register.

Prototype:

ADC_Result
ADC_GetConvertResult(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***)

Parameters:

ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

ResultREGx**:** Set ADC result register, which can be set as:

- **ADC_REG0**: ADxREG0
- **ADC_REG1**: ADxREG1
- **ADC_REG2**: ADxREG2
- **ADC_REG3**: ADxREG3
- **ADC_REG4**: ADxREG4
- **ADC_REG5**: ADxREG5
- **ADC_REG6**: ADxREG6
- **ADC_REG7**: ADxREG7
- **ADC_REG8**: ADxREG8
- **ADC_REG9**: ADxREG9
- **ADC_REG10**: ADxREG10
- **ADC_REG11**: ADxREG11

Description:

This function will read AD conversion result, overrun flag & AD conversion result storage flag by specified ADC result register by ***ResultREGx*** as **ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11**.

Return:

AD conversion result. Each bit has the following meaning:
Stored(Bit0): AD conversion result store flag
OverRun(Bit1): Overrun flag
ADResult(Bit4 to Bit15): AD conversion result

3.2.3.10 ADC_SelectPMDTrgProgNum

Select the program (program 0 to 5) to be started by each of trigger inputs PMDx of the specified ADC unit. (x = 6 to 11)

Prototype:

void
ADC_SelectPMDTrgProgNum(TSB_AD_TypeDef * ***ADx***,
PMD_TRG_PROG_SELx ***SELx***,
uint8_t ***MacroProgNum***)

Parameters:

ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

SELx**:** Specify the "trigger program number select register", which can be set as:

- **PMD_TRG_PROG_SEL6**: ADxPSEL6
- **PMD_TRG_PROG_SEL7**: ADxPSEL7

- **PMD_TRG_PROG_SEL8**: ADxPSEL8
- **PMD_TRG_PROG_SEL9**: ADxPSEL9
- **PMD_TRG_PROG_SEL10**: ADxPSEL10
- **PMD_TRG_PROG_SEL11**: ADxPSEL11

MacroProgNum**:** Set the program to be started by each of trigger inputs PMDx. (x = 6 to 11).
This parameter must be inputted with macro as the format below:

- **TRG_ENABLE(PMD_PROGy)**: Enable PMD Trigger Program Select Register with Program y. **(y = 0 to 5)**
- **TRG_DISABLE(PMD_PROGy)**: Disable PMD Trigger Program Select Register with Program y. **(y = 0 to 5)**

Description:
This function will set the PMD Trigger Program Number Select Register of the specified ADC unit by ***SELx***, and enable/disable the specified register to be started the specified program by ***MacroProgNum***.

Return:
None

3.2.3.11 ADC_SetPMDTrgProgINT

Select the interrupt to be generated for each of program 0 to 5 of the specified ADC unit.

Prototype:
void
ADC_SetPMDTrgProgINT(TSB_AD_TypeDef * ***ADx***,
PMD_TrgProgINTTypeDef * ***TrgProgINT***)

Parameters:
ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

TrgProgINT**:** The structure containing interrupt configuration for all of PMD Trigger Program Numbers.

```
typedef struct {
   PMD_INT_NAME INTProg0;
   PMD_INT_NAME INTProg1;
   PMD_INT_NAME INTProg2;
   PMD_INT_NAME INTProg3;
   PMD_INT_NAME INTProg4;
   PMD_INT_NAME INTProg5;
} PMD_TrgProgINTTypeDef
```

For details of this structure, refer to part “Data Structure Description”.

Description:
This function will set the interrupt to be generated for each of program 0 to 5 of the specified ADC unit by ***TrgProgINT***.

Return:
None

3.2.3.12 ADC_SetPMDTrg

Select PMD Trigger Program Register of the specified ADC unit.

Prototype:

void
ADC_SetPMDTrg(TSB_AD_TypeDef * ***ADx***,
PMD_TrgTypeDef * ***PMDTrg***)

Parameters:

ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

PMDTrg**:** The structure containing interrupt configuration for all of PMD Trigger Program Register.

```
typedef struct {
   PMD_PROGx ProgNum;
   VE_PHASE Reg0_Phase;
   VE_PHASE Reg1_Phase;
   VE_PHASE Reg2_Phase;
   VE_PHASE Reg3_Phase;
   uint8_t Reg0_AINx;
   uint8_t Reg1_AINx;
   uint8_t Reg2_AINx;
   uint8_t Reg3_AINx;
} PMD_TrgTypeDef
```

For details of this structure, refer to part “Data Structure Description”.

Description:

This function will set PMD Trigger Program Register of the specified ADC unit by ***PMDTrg***.

Return:

None

3.2.3.13 ADC_SetTimerTrg

Set Timer Trigger Program Register of the specified ADC unit.

Prototype:

void
ADC_SetTimerTrg(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***,
uint8_t ***MacroAINx***)

Parameters:

ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

ResultREGx**:** Set AD Conversion Result Register for programming timer triggers, which can be set as:

- **ADC_REG0:** ADxREG0
- **ADC_REG1:** ADxREG1
- **ADC_REG2:** ADxREG2
- **ADC_REG3:** ADxREG3
- **ADC_REG4:** ADxREG4
- **ADC_REG5:** ADxREG5

- **ADC_REG6:** ADxREG6
- **ADC_REG7:** ADxREG7
- **ADC_REG8:** ADxREG8
- **ADC_REG9:** ADxREG9
- **ADC_REG10:** ADxREG10
- **ADC_REG11:** ADxREG11

MacroAINx: Select AD Channel together with its enabled or disabled setting. This parameter must be inputted with macro as the format below:

- **TRG_ENABLE(y)**: Enable AD channel 'y' for ***ResultREGx***
- **TRG_DISABLE(y)**: Disable AD channel 'y' for ***ResultREGx***

'y' above can be one of the following values:

ADC_AIN9 to ADC_AIN12, ADC_AIN16

Description:
This function will set AD Conversion Result Register of the specified ADC unit by ***ResultREGx*** as **ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11**, and enable/disable REG with AIN pin by ***MacroAINx*** in Timer Trigger Program Register.

Return:
None

3.2.3.14 ADC_SetSWTrg

Set Software Trigger Program Register of the specified ADC unit.

Prototype:

```
void
ADC_SetSWTrg(TSB_AD_TypeDef * ADx,
                   ADC_REGx ResultREGx,
                   uint8_t MacroAINx)
```

Parameters:
ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

***ResultREGx*:** Set AD Conversion Result Register for programming software triggers.
which can be set as:

- **ADC_REG0:** ADxREG0
- **ADC_REG1:** ADxREG1
- **ADC_REG2:** ADxREG2
- **ADC_REG3:** ADxREG3
- **ADC_REG4:** ADxREG4
- **ADC_REG5:** ADxREG5
- **ADC_REG6:** ADxREG6
- **ADC_REG7:** ADxREG7
- **ADC_REG8:** ADxREG8
- **ADC_REG9:** ADxREG9
- **ADC_REG10:** ADxREG10
- **ADC_REG11:** ADxREG11

MacroAINx: Select AD Channel together with its enabled or disabled setting. This parameter must be inputted with macro as the format below:

- **TRG_ENABLE(y)**: Enable AD channel 'y' for ***ResultREGx***

- **TRG_DISABLE(y)**: Disable AD channel 'y' for ***ResultREGx***

'y' above can be one of the following values:
ADC_AIN9 to ADC_AIN12, ADC_AIN16

Description:
This function will set AD Conversion Result Register of the specified ADC unit by ***ResultREGx*** as **ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11**, and enable/disable REG with AIN pin by ***MacroAINx*** in Software Trigger Program Register.

Return:
None

3.2.3.15 ADC_SetConstantTrg

Set Constant Trigger Program Register of the specified ADC unit.

Prototype:
void
ADC_SetConstantTrg(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***,
uint8_t ***MacroAINx***)

Parameters:
ADx: Select ADC Unit, which can be set as:

- **TSB_ADB:** ADC Unit B

ResultREGx**:** Set AD Conversion Result Register for programming constant triggers.
which can be set as:

- **ADC_REG0:** ADxREG0
- **ADC_REG1:** ADxREG1
- **ADC_REG2:** ADxREG2
- **ADC_REG3:** ADxREG3
- **ADC_REG4:** ADxREG4
- **ADC_REG5:** ADxREG5
- **ADC_REG6:** ADxREG6
- **ADC_REG7:** ADxREG7
- **ADC_REG8:** ADxREG8
- **ADC_REG9:** ADxREG9
- **ADC_REG10:** ADxREG10
- **ADC_REG11:** ADxREG11

MacroAINx: Select AD Channel together with its enabled or disabled setting.
This parameter must be inputted with macro as the format below:

- **TRG_ENABLE(y)**: Enable AD channel 'y' for ***ResultREGx***
- **TRG_DISABLE(y)**: Disable AD channel 'y' for ***ResultREGx***

'y' above can be one of the following values:
ADC_AIN9 to ADC_AIN12, ADC_AIN16

Description:
This function will set AD Conversion Result Register of the specified ADC unit by ***ResultREGx*** as **ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11**, and enable/disable REG with AIN pin by ***MacroAINx*** in Constant Trigger Program Register.

Return:
None

3.2.4 Data Structure Description

3.2.4.1 ADC_MonitorTypeDef

Data Fields for this structure:

ADC_CMPCRx
CMPCRx Select Compare Control Register.
which can be:

- **ADC_CMPCR_0:** ADxCMPCR0
- **ADC_CMPCR_1:** ADxCMPCR1

ADC_REGx
ResultREGx Select which ADC Result Register to be used.
which can be set as:

- **ADC_REG0:** ADxREG0
- **ADC_REG1:** ADxREG1
- **ADC_REG2:** ADxREG2
- **ADC_REG3:** ADxREG3
- **ADC_REG4:** ADxREG4
- **ADC_REG5:** ADxREG5
- **ADC_REG6:** ADxREG6
- **ADC_REG7:** ADxREG7
- **ADC_REG8:** ADxREG8
- **ADC_REG9:** ADxREG9
- **ADC_REG10:** ADxREG10
- **ADC_REG11:** ADxREG11

uint32_t
CmpTimes Define how many times will comparison be counted.
which can be:

- **1** to **16**

ADC_CmpCondition
Condition Conditon to compare ADxREGm with ADxCMPn. (m = 0 to 11; x is B; n = 0 to 1)
which can be:

- **ADC_LARGER_THAN_CMP_REG**
- **ADC_SMALLER_THAN_CMP_REG**

uint32_t
CmpValue Comparison value to be set in ADxCMP0 or ADxCMP1. (x is B)
which can be:

- **0 to 4095**

3.2.4.2 PMD_TrgProgINTTypeDef

Data Fields for this structure:

PMD_INT_NAME
INTProg0 Select the interrupt to be generated for program 0, which can be:

- **PMD_INTNONE:** No interrupt output
- **PMD_INTADPDB:** INTADPDB output

PMD_INT_NAME
INTProg1 Select the interrupt to be generated for program 1, Other information is same as that of ***INTProg0*** above.

PMD_INT_NAME
INTProg2 Select the interrupt to be generated for program 2, Other information is same as that of ***INTProg0*** above.

PMD_INT_NAME
INTProg3 Select the interrupt to be generated for program 3,Other information is same as that of ***INTProg0*** above.

PMD_INT_NAME
INTProg4 Select the interrupt to be generated for program 4, Other information is same as that of ***INTProg0*** above.

PMD_INT_NAME
INTProg5 Select the interrupt to be generated for program 5, Other information is same as that of ***INTProg0*** above.

3.2.4.3 PMD_TrgTypeDef

Data Fields for this structure:

PMD_PROGx
ProgNum Select Program Number for ADxPSETn (x is B; n can be 0 to 5).
which can be:

- **PMD_PROG0:** Program Number 0
- **PMD_PROG1:** Program Number 1
- **PMD_PROG2:** Program Number 2
- **PMD_PROG3:** Program Number 3
- **PMD_PROG4:** Program Number 4
- **PMD_PROG5:** Program Number 5

VE_PHASE
Reg0_Phase Select phase for REG0 in ADxPSETn(x is B; n can be 0 to 5).
which can be:

- **VE_PHASE_NONE:** Not specified
- **VE_PHASE_U:** Phase U
- **VE_PHASE_V:** Phase V
- **VE_PHASE_W:** Phase W

VE_PHASE
Reg1_Phase Select phase for REG1 in ADxPSETn. Other information is same as ***Reg0_Phase***

VE_PHASE
Reg2_Phase Select phase for REG2 in ADxPSETn. Other information is same as ***Reg0_Phase***

VE_PHASE

Reg3_Phase Select phase for REG3 in ADxPSETn. Other information is same as ***Reg0_Phase***

uint8_t
Reg0_AINx Select AD Channel together with its enabled or disabled setting.
This parameter must be inputted with macro as the format below:

- **TRG_ENABLE(y)**: Enable AD channel 'y' for REG0
- **TRG_DISABLE(y)**: Disable AD channel 'y' for REG0

'y' above can be one of the following values:
ADC_AIN9 to ADC_AIN12, ADC_AIN16

uint8_t
Reg1_AINx Select Analog Input channel together with its enabled or disabled setting for REG1 in ADxPSETn. Other information is same as that of ***Reg0_AINx*** above.

uint8_t
Reg2_AINx Select Analog Input channel together with its enabled or disabled setting for REG2 in ADxPSETn. Other information is same as that of ***Reg0_AINx*** above.

uint8_t
Reg3_AINx Select Analog Input channel together with its enabled or disabled setting for REG3 in ADxPSETn. Other information is same as that of ***Reg0_AINx*** above.

3.2.4.4 ADC_Result

Data Fields for this structure:

uint32_t
All: AD Conversion Result.
Bit

uint32_t		
Stored:	1	AD result has been stored.
uint32_t		
OverRun:	1	Overrun flag.
uint32_t		
Reserved1:	2	Reserved.
uint32_t		
ADResult:	12	Store AD result.
uint32_t		
Reserved2:	16	Reserved.

4. CG

4.1 Overview

The CG API provides a set of functions for using the TMPM375FSDMG CG modules as the following:

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

This driver is contained in TX03_Periph_Driver\src\tmpm375_cg.c, with TX03_Periph_Driver\inc\tmpm375_cg.h containing the API definitions for use by applications.

The following symbols fosc, fosc1, fosc2, fpll, fc, fgear, fsys, fperiph, ΦT0 are used for kinds of clock in CG. Please refer to the clock system diagram in section “Clock System Block Diagram” of the datasheet for their meaning.

fosc : Clock input from high-speed oscillator.
fosc1 : Clock input from external high-speed oscillator (X1 and X2).
fosc2 : Clock input from internal high-speed oscillator
fpll : Clock quadrupled by PLL.
fc : Clock specified by CGPLLSEL<PLLSEL> (high-speed clock).
fgear : Clock specified by CGSYSCR<GEAR2:0>.
fsys : Clock specified by CGCKSEL<SYSCK> (system clock).
fperiph : Clock specified by CGSYSCR<FPSEL>.
ΦT0 : Clock specified by CGSYSCR<PRCK2:0> (prescaler clock).

4.2 API Functions

4.2.1 Function List

- void CG_SetFgearLevel(CG_DivideLevel ***DivideFgearFromFc***);
- CG_DivideLevel CG_GetFgearLevel(void);
- void CG_SetPhiT0Src(CG_PhiT0Src ***PhiT0Src***);
- CG_PhiT0Src CG_GetPhiT0Src(void);
- Result CG_SetPhiT0Level(CG_DivideLevel ***DividePhiT0FromFc***);
- CG_DivideLevel CG_GetPhiT0Level(void);
- void CG_SetWarmUpTime(CG_WarmUpSrc ***Source***, uint16_t ***Time***);
- void CG_StartWarmUp(void);
- WorkState CG_GetWarmUpState(void);
- Result CG_SetFPLLValue(uint32_t NewValue);
- uint32_t CG_GetFPLLValue(void);
- Result CG_SetPLL(FunctionalState ***NewState***);
- FunctionalState CG_GetPLLState(void);
- Result CG_SetFosc(CG_FoscSrc ***Source***, FunctionalState ***NewState***);
- void CG_SetFoscSrc(CG_FoscSrc ***Source***);

- CG_FoscSrc CG_GetFoscSrc(void);
- FunctionalState CG_GetFoscState(CG_FoscSrc ***Source***);
- void CG_SetPortM(CG_PortMMode ***Mode***);
- void CG_SetSTBYMode(CG_STBYMode ***Mode***);
- CG_STBYMode CG_GetSTBYMode(void);
- void CG_SetPinStateInStopMode(FunctionalState ***NewState***);
- FunctionalState CG_GetPinStateInStopMode(void);
- Result CG_SetFcSrc(CG_FcSrc ***Source***);
- CG_FcSrc CG_GetFcSrc(void);
- void CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,
  CG_INTActiveState ***ActiveState***,
  FunctionalState ***NewState***);
- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***);
- void CG_ClearINTReq(CG_INTSrc ***INTSource***);
- CG_NMIFactor CG_GetNMIFlag(void);
- CG_ResetFlag CG_GetResetFlag(void);

4.2.2 Detailed Description

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

1) One group of APIs are in charge of clock selection, such as:
CG_SetFgearLevel(), CG_GetFgearLevel(), CG_SetPhiT0Src(), CG_GetPhiT0Src(), CG_SetPhiT0Level(), CG_GetPhiT0Level(), CG_SetWarmUpTime(), CG_StartWarmUp(), CG_GetWarmUpState(), CG_SetFPLLValue(), CG_GetFPLLValue(), CG_SetPLL(), CG_GetPLLState(), CG_SetFosc(), CG_GetFoscState(), CG_SetFcSrc(), CG_GetFcSrc(), CG_SetFoscSrc(), CG_GetFoscSrc(), CG_SetPortM().
2) The 2nd group of APIs handle settings of standby modes:
CG_SetSTBYMode(), CG_GetSTBYMode(), CG_SetPinStateInStopMode(), CG_GetPinStateInStopMode().
3) The other APIs handle settings of interrupts:
CG_SetSTBYReleaseINTSrc(), CG_GetSTBYReleaseINTState(),CG_ClearINTReq(), CG_GetNMIFlag(), CG_GetResetFlag().

4.2.3 Function Documentation

4.2.3.1 CG_SetFgearLevel

Set the dividing level between clock fgear and fc.

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

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

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

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

Return:
None

4.2.3.2 CG_GetFgearLevel

Get the dividing level between fgear and fc.

Prototype:
CG_DivideLevel
CG_GetFgearLevel (void)

Parameters:
None

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

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

4.2.3.3 CG_SetPhiT0Src

Select the PhiT0(ΦT0) source between fperiph and fc.

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 will select the PhiT0(ΦT0) source.

Return:
None

4.2.3.4 CG_GetPhiT0Src

Get the PhiT0 (ΦT0) source.

Prototype:
CG_PhiT0Src
CG_GetPhiT0Src (void)

Parameters:
None

Description:
This function will get the PhiT0(ΦT0) source.

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

4.2.3.5 CG_SetPhiT0Level

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

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

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

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

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

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

4.2.3.6 CG_GetPhiT0Level

Get the dividing level between clock ΦT0 and fc.

Prototype:
CG_DivideLevel
CG_GetPhiT0Level(void)

Parameters:
None

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

If the value “Reserved” is read from the register, the API will return **CG_DIVIDE_UNKNOWN**.

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

4.2.3.7 CG_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_OSC1**: fosc1 is selected as timer source,
- **CG_WARM_UP_SRC_OSC2**: fosc2 is selected as timer source,

Time: If ***Source*** is **CG_WARM_UP_SRC_OSC1** or **CG_WARM_UP_SRC_OSC2,** 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:
Setting_value = ((warm-up time) / (input cycle time by frequency))/16

Example of calculating register value for warm-up time:
/* set up warm time 100us, input cycle by frequency is 8M */
So value = 100*10E(-6)/(1/(8*10E(6)))/16=0x0320>>4=0x32

Return:
None.

4.2.3.8 CG_StartWarmUp

Start warm up timer.

Prototype:

```
void
CG_StartWarmUp (void)
```

Parameters:
None

Description:
This function will start the warm up timer.

Return:
None

4.2.3.9 CG_GetWarmUpState

Check that warm-up operation is in middle or completed.

Prototype:
WorkState
CG_GetWarmUpState (void)

Parameters:
None

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

Example of using warm-up timer:

```
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC1, 0x32);
/* start warm up */
CG_StartWarmUp();
/* check warm up is finished or not*/
While( CG_GetWarmUpState() == BUSY);
```

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

4.2.3.10 CG_SetFPLLValue

Set PLL multiplying value.

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

Parameters:
NewValue:

- **CG_FPLL_8M_MULTIPLY_5:** Input clock 8MHz, output clock 40MHz (5 multiplying).
- **CG_FPLL_10M_MULTIPLY_4:** Input clock 10MHz, output clock 40MHz (4 multiplying)

Description:
This function sets PLL multiplying value.

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

4.2.3.11 CG_GetFPLLValue

Get the value of PLL setting.

Prototype:
uint32_t
CG_GetFPLLValue(void)

Parameters:
None.

Description:
This function will get the PLL multiplying value.

Return:
The source of PLL multiplying value.

- **CG_FPLL_8M_MULTIPLY_5:** Input clock 8MHz, output clock 40MHz (5 multiplying).
- **CG_FPLL_10M_MULTIPLY_4:** Input clock 10MHz, output clock 40MHz (4 multiplying)

4.2.3.12 CG_SetPLL

Enable or disable the PLL circuit.

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

Parameters:
NewState:

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

Description:
This function will enable or disable the PLL circuit as the input parameter.
If the PLL is selected as fc, it can’t be disabled; in that case the API will return **ERROR**.

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

4.2.3.13 CG_GetPLLState

Get the state of PLL circuit.

Prototype:
FunctionalState
CG_GetPLLState(void)

Parameters:
None

Description:

This function will get the state of PLL circuit.

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

4.2.3.14 CG_SetFosc

Enable or disable the high-speed oscillator (osc1 or osc2).

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

Parameters:
Source: select source for fosc.

- **CG_FOSC_OSC1**: fosc1 is selected,
- **CG_FOSC_OSC2**: fosc2 is selected.

NewState: select source for fosc.

- **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.
When fgear is selected as system clock (fsys) , the high-speed oscillator (fosc) can't be disabled; in this case the API will return **ERROR**.

Note:
The source of **CG_FOSC_OSC2.**

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

4.2.3.15 CG_SetFoscSrc

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

Prototype:
void
CG_SetFoscSrc(CG_FoscSrc ***Source***)

Parameters:
Source: select source for fosc.

- **CG_FOSC_OSC1**: fosc1 is selected,
- **CG_FOSC_OSC2**: fosc2 is selected.

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

Return:

None

4.2.3.16 CG_GetFoscSrc

Get the source of the high-speed oscillator.

Prototype:
CG_FoscSrc
CG_GetFoscSrc(void)

Parameters:
None

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

Return:
The source of fosc
CG_FOSC_OSC1: fosc1 is selected.
CG_FOSC_OSC2: fosc2 is selected.

4.2.3.17 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_OSC1**: fosc1 is selected,
- **CG_FOSC_OSC2**: fosc2 is selected.

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

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

4.2.3.18 CG_SetPortM

Set portM for X1/X2 or general port.

Prototype:
void
CG_SetPortM(CG_PortMMode ***Mode***)

Parameters:
Mode:

- **CG_PORTM_AS_GPIO** : to set port M as general port,
- **CG_PORTM_AS_HOSC**: to set port M as Hosc.

Description:
This function will set port M as general port when ***Mode*** is **CG_PORTM_AS_GPIO** and set port M as Hosc when ***Mode*** is **CG_PORTM_AS_HOSC**.

Return:
None

4.2.3.19 CG_SetSTBYMode

Set the standby mode.

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

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

- **CG_STBY_MODE_STOP**: STOP mode. All the internal circuits including the internal oscillator are brought to a stop.
- **CG_STBY_MODE_IDLE**: IDLE mode. Only CPU stop in this mode.

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

Return:
None

4.2.3.20 CG_GetSTBYMode

Get the standby mode.

Prototype:
CG_STBYMode
CG_GetSTBYMode (void)

Parameters:
None

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

Return:
The low power mode:
CG_STBY_MODE_STOP: STOP mode.
CG_STBY_MODE_IDLE: IDLE mode
CG_STBY_MODE_UNKNOWN: Invalid data is read.

4.2.3.21 CG_SetPinStateInStopMode

Specify the pin status in stop mode

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

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

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

Description:
This function will specify the pin status in stop mode.

Return:
None

4.2.3.22 CG_GetPinStateInStopMode

Get the pin status in stop mode

Prototype:
FunctionalState
CG_GetPinStateInStopMode (void)

Parameters:
None

Description:
This function will get the pin status in stop mode.

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

4.2.3.23 CG_SetFcSrc

Set the clock source of fc

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

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

Description:

This function will set the clock source of fc.
The following conditions should be matched before calling this API
 a) high-speed oscillator is set to on
 b) If the input for parameter ***Source*** is **CG_FC_SRC_FPLL**, PLL circuit must be enabled earlier (by calling "**CG_SetPLL**(ENABLE)") together with condition a) matched.
 Otherwise, calling of this API will return **ERROR**

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

4.2.3.24 CG_GetFcSrc

Get the clock source of fc.

Prototype:
CG_FcSrc
CG_GetFosc (void)

Parameters:
None

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

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

4.2.3.25 CG_SetSTBYReleaseINTSrc

Set the INT source for releasing low power mode.

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

Parameters:
INTSource: select the INT source for releasing standby mode
This parameter can be one of the following values:

- **CG_INT_SRC_6** : INT6
- **CG_INT_SRC_7** : INT7
- **CG_INT_SRC_C** : INTC

ActiveState: select the active state for release trigger.
This parameter can be one of the following values:

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

- **CG_INT_ACTIVE_STATE_BOTH_EDGES**: active on both edges

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

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

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

Return:
None

4.2.3.26 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_6, CG_INT_SRC_7, CG_INT_SRC_C.

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

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

4.2.3.27 CG_ClearINTReq

Clear the INT request for releasing standby mode.

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_6, CG_INT_SRC_7, CG_INT_SRC_C

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

Return:
None

4.2.3.28 CG_GetNMIFlag

Get the NMI flag, which shows what triggered NMI

Prototype:
CG_NMIFactor
CG_GetNMIFlag (void)

Parameters:
None

Description:
This function will get the NMI flag, which shows what triggered NMI.

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

4.2.3.29 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 will get the reset flag which shows the trigger of reset and clear the reset flag.

Return:
Reset flag:
PowerOn (Bit 0) means reset from power-on.
ResetPin (Bit 1) means reset from Reset pin.
WDTReset (Bit 2) means reset from WDT.
VLTDReset (Bit 3) means reset from VLTD.
DebugReset (Bit 4) means reset from SYSRESETREQ.
OFDReset (Bit 5) means reset from OFD.

4.2.4 Data Structure Description

4.2.4.1 CG_NMIFactor

Data Fields:
uint32_t
All specifies CGNMI source generation state.

Bit Fields:
uint32_t

WDT(Bit 0) means generated from WDT.

4.2.4.2 CG_ResetFlag

Data Fields:
uint32_t
All specifies CG reset source.

Bit Fields:
uint32_t
PowerOn(Bit 0) means reset from power-on.
uint32_t
ResetPin(Bit 1) means reset from Reset Pin.
uint32_t
WDTReset(Bit 2) means reset from WDT.
uint32_t
VLTDReset (Bit 3) means reset from VLTD.
uint32_t
DebugReset(Bit 4) means reset from debugger.
uint32_t
OFDReset(Bit 5) means reset from OFD

5. ENC

5.1 Overview

The TMPM375FSDMG has a one-channel incremental encoder interface (ENC0), which can obtain the absolute position of the motor, based on input signals from the incremental encoder.

The encoder input circuit supports four operation modes including encoder mode, sensor mode (two types) and timer mode. And the functions are as follows:

- Supports incremental encoders and Hall sensor ICs. (signals of Hall sensor IC can be input directly)
- 24-bit general-purpose timer mode
- Multiply-by-4 (multiply-by-6) circuit
- Rotational direction detection circuit
- 24-bit counter
- Comparator enable/disable
- Interrupt request output:1
- Digital noise filters for input signals

The ENC API provides a set of functions for using the TMPM375FSDMG ENC modules. It includes ENC channel select, mode set, compare function set, software caputer set, ENC status read, ENC counter read and so on.

This driver is contained in TX03_Periph_Driver\src\tmpm375_enc.c, with TX03_Periph_Driver\inc\tmpm375_enc.h containing the API definitions for use by applications.

5.2 API Functions

5.2.1 Function List

- void ENC_Enable(TSB_EN_TypeDef * ***ENx***);
- void ENC_Disable(TSB_EN_TypeDef * ***ENx***);
- void ENC_Init(TSB_EN_TypeDef * ***ENx***, ENC_InitTypeDef * ***InitStruct***);
- void ENC_SetSWCapture(TSB_EN_TypeDef * ***ENx***, uint32_t ***ENC_Mode***);
- void ENC_ClearCounter (TSB_EN_TypeDef * ***ENx***);
- ENC_FlagStatus ENC_GetENCFlag (TSB_EN_TypeDef * ***ENx***);
- void ENC_SetCounterReload (TSB_EN_TypeDef ****ENx***, uint32_t ***ENC_Mode***, uint32_t ***PeriodValue***);
- void ENC_SetCompareValue(TSB_EN_TypeDef * ***ENx***,uint32_t ***ENC_Mode***, uint32_t ***CompareValue***);
- uint32_t ENC_GetCompareValue(TSB_EN_TypeDef * ***ENx***);
- uint32_t ENC_GetCounterValue(TSB_EN_TypeDef * ***ENx***);

5.2.2 Detailed Description

Functions listed above can be divided into three parts:

1) ENC setting by ENC_Init(), ENC_ClearCounter(), ENC_SetCounterReload(), ENC_SetSWCapture(), ENC_SetCompareValue().
2) ENC function enable/disable by ENC_Enable(), ENC_Disable().
3) ENC state or data read functions by ENC_GetENCFlag(), ENC_GetCounterValue(), ENC_GetCompareValue().

5.2.3 Function Documentation

5.2.3.1 ENC_Enable

Enable the specified encoder operation.

Prototype:
void
ENC_Enable(TSB_EN_TypeDef * ***ENx***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- **EN0**

Description:
This function will enable the specified encoder operation.

Return:
None

5.2.3.2 ENC_Disable

Disable the specified encoder operation.

Prototype:
void
ENC_Disable(TSB_EN_TypeDef * ***ENx***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- **EN0**

Description:
This function will disable the specified encoder operation.

Return:
None

5.2.3.3 ENC_Init

Initialize the specified encoder operation.

Prototype:

void
ENC_Init(TSB_EN_TypeDef * ***ENx***, ENC_InitTypeDef * ***InitStruct***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:
- **EN0**

InitStruct: The structure containing basic encoder configuration(see Data Structure for details).

Description:
This function will initialize the specified encoder operation.

Return:
None

5.2.3.4 ENC_SetSWCapture

Set the specified encoder to execute software capture (timer mode/sensor mode (at timer count)).

Prototype:
void
ENC_SetSWCapture(TSB_EN_TypeDef * ***ENx***, uint32_t ***ENC_Mode***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:
- **EN0**

ENC_Mode: Select the encoder operation mode.
This parameter can be one of the following values:
- **ENC_TIMER_MODE**
- **ENC_SENSOR_TIME_MODE**

Description:
This function will set the specified encoder to execute software capture (timer mode/sensor mode (at timer count)).

Return:
None

5.2.3.5 ENC_ClearCounter

Clear pulse counter for the specified encoder.

Prototype:
void
ENC_ClearCounter (TSB_EN_TypeDef * ***ENx***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:
- **EN0**

Description:
This function will clear pulse counter for the specified encoder.

Return:
None

5.2.3.6 ENC_GetENCFlag

Get the encoder compare flag/reverse error flag/Z-detected/rotation direction.

Prototype:
ENC_FlagStatus
ENC_GetENCFlag (TSB_EN_TypeDef * ***ENx***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- **EN0**

Description:
This function will get the encoder compare flag/reverse error flag/Z-detected/rotation direction.

Return:
The encoder flag
ZPhaseDetectFlag(bit12) means ENC Z-phase detect flag.

RotationDirection (bit13) means ENC rotation direction.

ReverseErrorFlag (bit14) means ENC sensor mode (at time count) reverse error flag.

CompareFlag (bit15) means ENC compare flag.

5.2.3.7 ENC_SetCounterReload

Set the encoder counter period.

Prototype:
void
ENC_SetCounterReload (TSB_EN_TypeDef * ***ENx***, uint32_t ***PeriodValue***)
Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- **EN0**

PeriodValue: Set the encoder counter period.
This parameter can be **0x0000 - 0xFFFF**

Description:
This function will set the encoder counter period.

Return:
None

5.2.3.8 ENC_SetCompareValue

Set the encoder counter compare value.

Prototype:
void
ENC_SetCompareValue(TSB_EN_TypeDef * ***ENx***,uint32_t ***ENC_Mode***,
uint32_t ***CompareValue***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- ➢ **EN0**

ENC_Mode: Select the encoder operation mode.
This parameter can be one of the following values:

- ➢ **ENC_ENCODER _MODE**
- ➢ **ENC_SENSOR_EVENT_MODE**
- ➢ **ENC_SENSOR_TIME_MODE**
- ➢ **ENC_TIMER_MODE**

CompareValue:Set the encoder counter compare value
In sensor mode (event count) and encoder mode:
This parameter can be **0x0000 - 0xFFFF**.

In sensor mode (timer count) and timer mode:
This parameter can be **0x000000 - 0xFFFFFF**.

Description:
This function will set the encoder counter compare value.

Return:
None

5.2.3.9 ENC_GetCompareValue

Get the encoder counter compare value.

Prototype:
uint32_t
ENC_GetCompareValue(TSB_EN_TypeDef * **ENx**)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- ➢ **EN0**

Description:
This function will get the encoder counter compare value.

Return:
Compare value of the encoder.

5.2.3.10 ENC_GetCounterValue

Get the encoder counter/capture value.

Prototype:
uint32_t
ENC_GetCounterValue(TSB_EN_TypeDef * ***ENx***)

Parameters:
ENx: Select ENC channel.
This parameter can be one of the following values:

- **EN0**

Description:
This function will get the encoder counter/capture value.

Return:
Value of the encoder counter.

5.2.4 Data Structure Description

5.2.4.1 ENC_InitTypeDef

Data Fields:

uint32_t

ModeType encoder input mode selection, which can be:

- **ENC_ENCODER_MODE**
- **ENC_SENSOR_EVENT_MODE**
- **ENC_SENSOR_TIME_MODE**
- **ENC_TIMER_MODE**

uint32_t

PhaseType 2-phase / 3-phase input selection (sensor mode), which can be:

- **ENC_TWO_PHASE**
- **ENC_THREE_PHASE**

uint32_t

EdgeType edge selection of ENCZ (timer mode), which can be:

- **ENC_RISING_EDGE**
- **ENC_FALLING_EDGE**

uint32_t

CompareStatus enable or disable the encoder compare function, which can be:

- **ENC_COMPARE_DISABLE**
- **ENC_COMPARE_ENABLE**

uint32_t

ZphaseStatus enable or disable the Z-phase (Encoder mode/timer mode), which can be:

- **ENC_ZPHASE_DISABLE**
- **ENC_ZPHASE_ENABLE**

uint32_t

FilterValue Set the noise filter effect value, which can be:

- **ENC_NO_FILTER**
- **ENC_FILTER_VALUE31**
- **ENC_FILTER_VALUE63**
- **ENC_FILTER_VALUE127**

uint32_t

PulseDivFactor Set the encoder pulse division factor, which can be:

- **ENC_PULSE_DIV1**
- **ENC_PULSE_DIV2**
- **ENC_PULSE_DIV4**
- **ENC_PULSE_DIV8**
- **ENC_PULSE_DIV16**
- **ENC_PULSE_DIV32**
- **ENC_PULSE_DIV64**
- **ENC_PULSE_DIV128**

5.2.4.2 ENC_FlagStatus

Data Fields:

uint32_t

All specifies ENC all flag status

uint32_t

ZPhaseDetectFlag(bit12) means ENC Z-phase detect flag.

uint32_t

RotationDirection (bit13) means ENC rotation direction.

uint32_t

ReverseErrorFlag (bit14) means ENC sensor mode (at time count) reverse error flag.

uint32_t

CompareFlag (bit15) means ENC compare flag.

6. FC

6.1 Overview

TMPM375FSDMG device contains flash memory.
For TMPM375FSDMG,the size of flash is 64Kbyte.

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

The block configuration of flash memory please refers to the MCU data sheet.

This driver is contained in \Libraries\TX03_Periph_Driver\src\tmpm375_fc.c with \Libraries\TX03_Periph_Driver \inc\tmpm375_fc.h containing the API definitions for use by applications.

6.2 API Functions

6.2.1 Function List

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

6.2.2 Detailed Description

Functions listed above can be divided into five parts:

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

6.2.3 Function Documentation

6.2.3.1 FC_SetBufferState

Set the value of CR register.

Prototype:
void
FC_ SetBufferState (FunctionalState ***NewState***)

Parameters:
NewState: Select the state of CR register.
This parameter can be one of the following values:

- **DISABLE**: Disable Instruction Buffer(with Buffer clear)
- **ENABLE**: Enable Instruction Buffer.

Description:
After Flash programing or flash erasing, it should clear instruction buffer by this function.

6.2.3.2 FC_SetSecurityBit

Set the value of SECBIT register.

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

Parameters:
NewState: Select the state of SECBIT register.
This parameter can be one of the following values:

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

Description:

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

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

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

Return:
None

6.2.3.3 FC_GetSecurityBit

Get the value of SECBIT register.

Prototype:
FunctionalState
FC_GetSecurityBit(void)

Parameters:
None

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

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

6.2.3.4 FC_GetBusyState

Get the status of the flash auto operation.

Prototype:
WorkState
FC_GetBusyState (void)

Parameters:
None

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

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

6.2.3.5 FC_GetBlockProtectState

Get the block protection status.

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

Parameters:
BlockNum:The flash block number

- **FC_BLOCK_0** for block 0
- **FC_BLOCK_1** for block 1

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

Return:

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

6.2.3.6 FC_ProgramBlockProtectState

Program the protection bits.

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

Parameters:
BlockNum:The flash block number

- **FC_BLOCK_0** for block 0
- **FC_BLOCK_1** for block 1

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

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

6.2.3.7 FC_EraseBlockProtectState

Erase the protection bits.

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

Parameters:
BlockGroup:The flash block group

- **FC_BLOCK_GROUP_0** for block 0, block 1.

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

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

6.2.3.8 FC_WritePage

Write data to the specified page.

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

Parameters:
PageAddr:The page start address
Data: The pointer to data buffer to be written into the page. The data size should be 128Byte.

Description:
This API is used to write data to specified page.
The TMPM375FSDMG contains 32 words in a page. The flash can only be written page by page.
The automatic page programming is allowed only once for a page already erased. No programming can be performed twice or more time irrespective of data value whether it is “1” or “0”.
***Note**: An attempt to rewrite a page two or more times without erasing the content can cause damages to the device.

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

6.2.3.9 FC_EraseBlock

Erase the content of specified block.

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

Parameters:
BlockAddr: The block starts address.

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

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

6.2.3.10 FC_EraseChip

Erase the content of the entire chip.

Prototype:
FC_Result
FC_EraseChip(void)

Parameters:

None

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

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

6.2.4 Data Structure Description

None.

7. GPIO

7.1 Overview

For TOSHIBA TMPM375FSDMG general-purpose I/O ports, inputs and outputs can be specified in units of bits. Besides the general-purpose input/output function, all ports perform specified function.

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

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/ tmpm375 _gpio.c, with /Libraries/TX03_Periph_Driver/inc/tmpm375 _gpio.h containing the macros, data types, structures and API definitions for use by applications.

7.2 API Functions

7.2.1 Function List

- uint8_t GPIO_ReadData(GPIO_Port ***GPIO_x***)
- uint8_t GPIO_ReadDataBit(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***)
- void GPIO_WriteData(GPIO_Port ***GPIO_x***, uint8_t ***Data***)
- void GPIO_WriteDataBit(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***, uint8_t ***BitValue***)
- void GPIO_Init(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  GPIO_InitTypeDef **** GPIO_InitStruct***)
- void GPIO_SetOutput(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***)
- void GPIO_SetInput(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***);
- void GPIO_SetOutputEnableReg(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***)

- void GPIO_SetInputEnableReg(GPIO_Port **GPIO_x**, uint8_t **Bit_x**,
  FunctionalState ***NewState***)
- void GPIO_SetPullUp(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***)
- void GPIO_SetPullDown(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***)
- void GPIO_SetOpenDrain(GPIO_Port ***GPIO_x***, uint8_t ***Bit_x***,
  FunctionalState ***NewState***)
- void GPIO_EnableFuncReg(GPIO_Port ***GPIO_x***, uint8_t ***FuncReg_x***, uint8_t ***Bit_x***)
- void GPIO_DisableFuncReg(GPIO_Port ***GPIO_x***, uint8_t ***FuncReg_x***, uint8_t ***Bit_x***)

7.2.2 Detailed Description

Functions listed above can be divided into three parts:

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

7.2.3 Function Documentation

7.2.3.1 GPIO_ReadData

Read specified GPIO Data register.

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

Description:
This function will read specified GPIO Data register.

Return:
The value read from DATA register.

7.2.3.2 GPIO_ReadDataBit

Read specified GPIO pin.

Prototype:
uint8_t
GPIO_ReadDataBit(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***)

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

Description:

This function will read specified GPIO pin.

Return:

The value read from GPIO pin as:

- **GPIO_BIT_VALUE_0**: Value 0,
- **GPIO_BIT_VALUE_1**: Value 1.

7.2.3.3 GPIO_WriteData

Write specified value to GPIO Data register.

Prototype:

void
GPIO_WriteData(GPIO_Port ***GPIO_x***,
uint8_t ***Data***)

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

Description:

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

Return:
None

7.2.3.4 GPIO_WriteDataBit

Write specified value of single bit to GPIO pin.

Prototype:

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

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

7.2.3.5 GPIO_Init

Initialize GPIO port function.

Prototype:

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

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

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

Return:
None

7.2.3.6 GPIO_SetOutput

Set specified GPIO pin as output port.

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,

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

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

Return:
None

7.2.3.7 GPIO_SetInput

Set specified GPIO Pin as input port.

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

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

Return:
None

7.2.3.8 GPIO_SetOutputEnableReg

Enable or disable specified GPIO Pin output function.

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

NewState:

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

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

Return:
None

7.2.3.9 GPIO_SetInputEnableReg

Enable or disable specified GPIO Pin input function.

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.

- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

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

Return:
None

7.2.3.10 GPIO_SetPullUp

Enable or disable specified GPIO Pin pull-up function.

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

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

Bit_x: Select GPIO pin, which can be set as:
- **GPIO_BIT_0:** GPIO pin 0,
- **GPIO_BIT_1:** GPIO pin 1,
- **GPIO_BIT_2:** GPIO pin 2,
- **GPIO_BIT_3:** GPIO pin 3,

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

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

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

Return:
None

7.2.3.11 GPIO_SetPullDown

Enable or disable specified GPIO Pin pull-down function.

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

Parameters:
GPIO_x: Select GPIO port, which can be set as:
- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

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

Return:
None

7.2.3.12 GPIO_SetOpenDrain

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

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

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.
- **GPIO_PJ :** GPIO port J.
- **GPIO_PK:** GPIO port K.
- **GPIO_PM:** GPIO port M.

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

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

NewState:

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

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

Return:
None

7.2.3.13 GPIO_EnableFuncReg

Enable specified GPIO function.

Prototype:

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

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.

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

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

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

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

Description:

This function will enable GPIO pin specified function.

Return:

None

7.2.3.14 GPIO_DisableFuncReg

Disable specified GPIO function.

Prototype:

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

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.

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

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

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

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

Description:
This function will disable GPIO pin specified function.

Return:
None

7.2.4 Data Structure Description

7.2.4.1 GPIO_InitTypeDef

Data Fields:

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

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

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

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

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

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

uint8_t

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

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

8. SBI

8.1 Overview

The TMPM375FSDMG contains Serial Bus Interface (SBI), The TMPM375FSDMG has 1 channel (SBI), SBI can work in I2C bus mode with multi-master capability or in I2C free mode (master is fixed to send data and slave is fixed to receive data).

In I2C bus mode, the SBI is connected to external devices via SCL and SDA.
Data can be transferred in free data format by the SBI channel. In free data format, data is always sent by master-transmitter and received by slave-receiver.

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

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm375_sbi.c, with /Libraries/TX03_Periph_Driver/inc/tmpm375_sbi.h containing the macros, data types, structures and API definitions for use by applications.

8.2 API Functions

8.2.1 Function List

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

8.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Configure and control the common functions of each SBI channel are handled by SBI_Enable(), SBI_Disable(), SBI_SetI2CACK(), SBI_SetI2CBitNum(), and SBI_InitI2C().

2) Transfer control of each SBI channel is handled by SBI_ClearI2CINTReq(), SBI_GenerateI2Cstart(), SBI_GenerateI2Cstop(),SBI_SetSendData(), SBI_GetReceiveData().
3) The status indication of each SBI channel is handled by SBI_GetI2CState().
4) SBI_SWReset(), SBI_SetIdleMode() and SBI_EnableI2CfreeDataMode() handle other specified functions.

8.2.3 Function Documentation

8.2.3.1 SBI_Enable

Enable the specified SBI channel.

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
None

8.2.3.2 SBI_Disable

Disable the specified SBI channel.

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
None

8.2.3.3 SBI_SetI2CACK

Enable or disable the generation of ACK clock.

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

Parameters:
SBIx is the specified SBI channel.

NewState sets the generation of ACK clock, which can be:

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

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

Return:
None

8.2.3.4 SBI_InitI2C

Initialize the specified SBI channel in I2C mode.

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

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

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

Return:
None

8.2.3.5 SBI_SetI2CBitNum

Specify the number of bits per transfer.

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

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

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

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

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

Return:
None

8.2.3.6 SBI_SWReset

Reset the state of the specified SBI channel.

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
None

8.2.3.7 SBI_ClearI2CINTReq

Clear SBI interrupt request in I2C bus mode.

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
None

8.2.3.8 SBI_GenerateI2CStart

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

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
None

8.2.3.9 SBI_GenerateI2CStop

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

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
None

8.2.3.10 SBI_GetI2CState

Get the SBI channel state in I2C bus mode.

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

Parameters:
SBIx is the specified SBI channel.

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

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

8.2.3.11 SBI_SetIdleMode

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

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

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

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

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

Return:
None

8.2.3.12 SBI_SetSendData

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

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

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

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

Return:
None

8.2.3.13 SBI_GetReceiveData

Get data received from the specified SBI channel.

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

Parameters:
SBIx is the specified SBI channel.

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

Return:
Data which has been received

8.2.3.14 SBI_SetI2CFreeDataMode

Set SBI channel working in I2C free data mode.

Prototype:
void
SBI_SetI2CFreeDataMode(TSB_SBI_TypeDef* ***SBIx***,
FunctionalState ***NewState***)

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

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

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

Return:
None

8.2.4 Data Structure Description

8.2.4.1 SBI_InitI2CTypeDef

Data Fields:

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

uint32_t
I2CDataLen Specify data length of the SBI channel in I2C mode, which can be set as:

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

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

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

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

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

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

8.2.4.2 SBI_I2CState

Data Fields:

uint32_t
All specifies state data in I2C mode

Bit Fields:

uint32_t
LastRxBit specifies last received bit monitor.

uint32_t
GeneralCall specifies general call detected monitor.

uint32_t
SlaveAddrMatch specifies slave address match monitor.

uint32_t
ArbitrationLost specifies arbitration last detected monitor.

uint32_t
INTReq specifies Interrupt request monitor.

uint32_t
BusState specifies bus busy flag.

uint32_t
TRx specifies transfer or Receive selection monitor.

uint32_t
MasterSlave specifies master or slave selection monitor.

9. OFD

9.1 Overview

TMPM375FSDMG has an oscillation frequency detector circuit (OFD), which can generate a reset signal when abnormal states of clock such as a harmonic, a sub harmonic or stopped state is detected.

The OFD driver APIs provide a set of functions to enable or disable the OFD function, configure detection frequency, get the OFD status and so on.

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm375_ofd.c, with /Libraries/TX03_Periph_Driver/inc/tmpm375_ofd.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 Function List

- void OFD_SetRegWriteMode(FunctionalState NewState);
- void OFD_Enable(void);
- void OFD_Disable(void);
- void OFD_SetDetectionFrequency(OFD_PLL_State State,
  uint32_t HigherDetectionCount,
  uint32_t LowerDetectionCount);

9.2.2 Detailed Description

Initialize and configure OFD function by OFD_SetRegWriteMode(), OFD_SetDetectionFrequency (),OFD_Enable () and OFD_Disable ().

9.2.3 Function Documentation

9.2.3.1 OFD_SetRegWriteMode

Enable or disable the writing of OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF.

Prototype:

void
OFD_SetRegWriteMode(FunctionalState ***NewState***)

Parameters:

NewState is the new state of writing OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF registers.

This parameter can be one of the following values:

- **ENABLE:**OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF registers can be written.

- **DISABLE:**OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/ OFDMXPLLOFF registers can't be written.

Description:
This function will enable writing of OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF registers when ***NewState*** is **ENABLE**, and disable writing of OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF registers when ***NewState*** is **DISABLE**.

Return:
None

9.2.3.2 OFD_Enable

Enable the OFD function.

Prototype:
void
OFD_Enable(void)

Parameters:
None.

Description:
This function will enable the OFD function.

Return:
None

9.2.3.3 OFD_Disable

Disable the OFD function.

Prototype:
void
OFD_Disable(void)

Parameters:
None.

Description:
This function will disable the OFD function.

Return:
None

9.2.3.4 OFD_SetDetectionFrequency

Set the count value of detection frequency.

Prototype:

void
OFD_SetDetectionFrequency(OFD_PLL_State **State**,
Uint32_t ***HigherDetectionCount***,
Uint32_t ***LowerDetectionCount***)

Parameters:

State: Select the state of PLL.
This parameter can be one of the following values:

- **OFD_PLL_ON:** Select the CG PLL on state
- **OFD_PLL_OFF:** Select the CG PLL off state

HigherDetectionCount: the count value of higher detection frequency.
LowerDetectionCount: the count value of lower detection frequency.

Description:

This function will set the count value of detection frequency, both higher detection frequency and lower detection frequency, both PLL on and PLL off.

Return:

None

9.2.4 Data Structure Description

None.

10. TMRB

10.1 Overview

TOSHIBA TMPM375FSDMG contains 4 channels of multi-functional 16-bit timer/event counter (TMRB0, TMRB4, TMRB5, TMRB7). Each channel can operate in the following modes:

- 16-bit interval timer mode
- 16-bit event counter mode
- 16-bit programmable pulse generation mode (PPG)
- External trigger Programmable pulse generation mode (PPG)

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

- Pulse width measurement
- One-shot pulse generation from an external trigger pulse

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

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm375_tmrb.c, with /Libraries/TX03_Periph_Driver/inc/tmpm375_tmrb.h containing the macros, data types, structures and API definitions for use by applications.

10.2 API Functions

10.2.1 Function List

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

- void TMRB_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***);

10.2.2 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_SetDoubleBuf(), TMRB_SetExtStartTrg() and TMRB_SetClkInCoreHalt () handle other specified functions.

10.2.3 Function Documentation

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

10.2.3.1 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

10.2.3.2 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

10.2.3.3 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

10.2.3.4 TMRB_Init

Initialize the specified TMRB channel.

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

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

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

Return:
None

Note:
If the TMRB Channels don’t have TBxIN, ***InitStruct->mode*** can only be **TMRB_INTERVAL_TIMER**.

10.2.3.5 TMRB_SetCaptureTiming

Configure the capture timing.

Prototype:

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

Parameters:

TBx: Select the TMRB channel.
This parameter can be one of the following values:
TSB_TB0, TSB_TB4, TSB_TB7.

CaptureTiming specifies TMRB capture timing, which can be

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

Note:
When TBx = TSB_TB0, CaptureTiming can't be **TMRB_CAPTURE_IN_RISING** and **TMRB_CAPTURE_IN_RISING_FALLING**.

When TBx = TSB_TB4 or TSB_TB7, CaptureTiming can't be **TMRB_CAPTURE_TIMPLS_RISING_FALLING**.

Description:

If ***CaptureTiming*** is set as **TMRB_CAPTURE_IN_RISING**, then at the time of the rising edge of input port TBxIN, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel.
If ***CaptureTiming*** is set as **TMRB_CAPTURE_IN_RISING_FALLING**, then at the time of the rising edge of input port TBxIN, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TBxIN pin input.
If ***CaptureTiming*** is set as **TMRB_CAPTURE_TIMPLS_RISING_FALLING**, then at the time of the rising edge of TIMPLS input, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TIMPLS input.

Return:

None

10.2.3.6 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, and it should be:
TSB_TB7.

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

10.2.3.7 TMRB_GetINTFactor

Indicate what causes the interrupt.

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

Parameters:
TBx is the specified TMRB channel.

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

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

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

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

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

 if (factor.Bit.OverFlow) {
      // Do C
```

```
    }
```

10.2.3.8 TMRB_SetINTMask

Mask the specified TMRB interrupt.

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

Parameters:

TBx is the specified TMRB channel.

INTMask specifies the interrupt to be masked, which can be

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

Description:

If **TMRB_MASK_MATCH_TRAILINGTIMING_INT** is selected, the interrupt of the specified TMRB channel will not happen when the value in up-counter and trailing timing are match.

If **TMRB_MASK_MATCH_LEADINGTIMING_INT** is selected, the interrupt of the specified TMRB channel will not happen when the value in up-counter and leading timing are match.

If **TMRB_MASK_OVERFLOW_INT** is selected, the interrupt of the specified TMRB channel will not happen even if there is an occurrence of overflow.

If **TMRB_NO_INT_MASK** is selected, all interrupt masks will be cleared.

Return:
None

10.2.3.9 TMRB_ChangeLeadingTiming

Change the value of leading timing for the specified channel.

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

Parameters:

TBx is the specified TMRB channel.

LeadingTiming specifies the value of leading timing, max is 0xFFFF.

Description:

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

Return:
None

Note:
LeadingTiming cannot exceed ***TrailingTiming***.

10.2.3.10 TMRB_ChangeTrailingTiming

Change the value of trailing timing for the specified channel.

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

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

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

Return:
None

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

10.2.3.11 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.

10.2.3.12 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 according to the value of ***CapReg***.

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.
  TBx is the specified TMRB channel.
- **TMRB_CAPTURE_1:** specifying capture register1.
  TBx is the specified TMRB channel, and it should be:
  TSB_TB0, TSB_TB4, TSB_TB7.

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

10.2.3.13 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

10.2.3.14 TMRB_SetIdleMode

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

Prototype:

```
void
TMRB_SetIdleMode(TSB_TB_TypeDef* TBx,
                 FunctionalState NewState)
```

Parameters:
TBx is the specified TMRB channel.

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

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

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

Return:
None

10.2.3.15 TMRB_SetDoubleBuf

Enable or disable double buffering for the specified TMRB channel and set the timing to write to timer register 0 and 1 when double buffer enabled.

Prototype:
void
TMRB_SetDoubleBuf(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***,
uint8_t ***WriteRegMode***)

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

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

WriteRegMode specifies timing to write to timer register 0 and 1 when double buffer enabled, which can be

- **TMRB_WRITE_REG_SEPARATE:** Timer register 0 and 1 can be written separately, even in case writing preparation is ready for only one register.
- **TMRB_WRITE_REG_SIMULTANEOUS:** In case both registers are not ready to be written, timer registers 0 and 1 can't be written.

Description:
The register TBxRG0 (***LeadingTiming***) and TBxRG1 (***TrailingTiming***) and their buffers are assigned to the same address. If double buffering is disabled, the same value is written to the registers and their buffers.
If double buffering is enabled, the value is only written to each register buffer. Therefore, to write an initial value to the registers, TBxRG0 (***LeadingTiming***) and TBxRG1 (***TrailingTiming***), the double buffering must be set to **DISABLE**. Then **ENABLE** double buffering and write the following data to the register, which can be loaded when the corresponding interrupt occurs automatically.

Return:
None

10.2.3.16 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, and it should be:
TSB_TB4, TSB_TB7.

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

10.2.3.17 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

10.2.4 Data Structure Description

10.2.4.1 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).

Note: Channels which don't have TBxIN ***InitStruct->mode*** can only be **TMRB_INTERVAL_TIMER**.

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

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

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

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

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

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

10.2.4.2 TMRB_FFOutputTypeDef

Data Fields:

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

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

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

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

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

10.2.4.3 TMRB_INTFactor

Data Fields:

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

11. SIO/UART

11.1 Overview

This device has two serial I/O channels. SIO0 can operate in UART mode (asynchronous communication) and I/O Interface mode(synchronous communication). SIO1 can only operate in UART mode (asynchronous communication).
About UART mode (asynchronous communication) which can be 7-bit length, 8-bit length and 9-bit length. In 9-bit UART mode, a wakeup function can be used when the master controller can start up slave controllers via the serial link (multi-controller system).

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

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm375_uart.c, with /Libraries/TX03_Periph_Driver/inc/tmpm375_uart.h containing the macros, data types, structures and API definitions for use by applications.

11.2 API Functions

11.2.1 Function List

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

11.2.2 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(), UART_DefaultConfig(), UART_SetInputClock(), SIO_Enable(), SIO_Disable(), SIO_SetInuptClock() and SIO_Init().
2) Transfer control and error check of each UART channel are handled by UART_GetBufState(), UART_GetRxData(), UART_SetTxData(), UART_GetErrState(), SIO_GetRxData() and SIO_SetTxData.
3) UART_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().

11.2.3 Function Documentation

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

11.2.3.1 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

11.2.3.2 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

11.2.3.3 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.

11.2.3.4 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

11.2.3.5 UART_Init

Initialize and configure the specified UART channel.

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

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

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

Return:
None

11.2.3.6 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, the data value range is 0x00 to 0x1FF.

11.2.3.7 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 UART mode. The data value range is 0x00 to 0x1FF.

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

11.2.3.8 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

11.2.3.9 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.

11.2.3.10 UART_SetWakeUpFunc

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

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

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

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

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

Return:
None

11.2.3.11 UART_SetIdleMode

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

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

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

This parameter can be one of the following values:

ENABLE or **DISABLE**

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

Return:
None

11.2.3.12 UART_SetInputClock

Set the specified UART channel InputClock
Prototype:
void
UART_SetInputClock(TSB_SC_TypeDef * ***UARTx***, uint8_t ***ClkDivider***)
Parameters:
UARTx is the specified UART channel.
ClkDivider is the input clock divider for prescaler
This parameter can be one of the following values:
UART_DIVIDE_1_1or **UART_DIVIDE_1_2**
Description:
This function will Set InputClock for the specified UART channel selected by ***UARTx*** when ***ClkDivider*** is **UART_DIVIDE_1_1**, and when ***ClkDivider*** is **UART_DIVIDE_1_2**.

Return:
None

11.2.3.13 UART_FIFOConfig

Enable or disable FIFO.

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

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

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

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

Return:
None

11.2.3.14 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

11.2.3.15 UART_TRxAutoDisable

Controls automatic disabling of transmission and reception.

Prototype:
void
UART_TRxAutoDisable (TSB_SC_TypeDef* ***UARTx***,
UART_TRxAutoDisable ***TRxAutoDisable***);

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

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

Description:
Controls automatic disabling of transmission and reception.

Return:
None

11.2.3.16 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

11.2.3.17 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

11.2.3.18 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

11.2.3.19 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

11.2.3.20 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

11.2.3.21 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

11.2.3.22 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

11.2.3.23 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

11.2.3.24 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

11.2.3.25 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

11.2.3.26 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.

11.2.3.27 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.

11.2.3.28 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.

11.2.3.29 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.

11.2.3.30 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

11.2.3.31 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

11.2.3.32 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

11.2.3.33 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

11.2.3.34 SIO_Init

Initialize and configure the specified SIO channel.

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

Parameters:
SIOx is the specified SIO channel.
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

11.2.3.35 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

11.2.3.36 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

11.2.4 Data Structure Description

11.2.4.1 UART_InitTypeDef

Data Fields:

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

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

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

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

- **UART_EVEN_PARITY** for even parity
- **UART_ODD_PARITY** for odd parity

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

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

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

- **UART_NONE_FLOW_CTRL** for no flow control

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

11.2.4.2 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.

12. VLTD

12.1 Overview

The voltage detection circuit detects any decrease in the supply voltage and generates reset signal.

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

All driver APIs are contained in /Libraries/TX03_Periph_Driver/src/tmpm375_vltd.c, with /Libraries/TX03_Periph_Driver/inc/tmpm375_vltd.h containing the macros, data types, structures and API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- void VLTD_Enable(void);
- void VLTD_Disable(void);
- void VLTD_SetVoltage(uint32_t ***Voltage***);

12.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) Enable or disable VLTD are handled by VLTD_Enable() and VLTD_Disable().
2) Select detection voltage by VLTD_SetVoltage().

12.2.3 Function Documentation

12.2.3.1 VLTD_Enable

Enable the VLTD function.

Prototype:
void
VLTD_Enable(void)

Parameters:
None

Description:
This function will enable the VLTD function.

Return:
None

12.2.3.2 VLTD_Disable

Disable the VLTD function.

Prototype:
void
VLTD_Disable(void)

Parameters:
None

Description:
This function will disable the VLTD function.

Return:
None

12.2.3.3 VLTD_SetVoltage

Select the detection voltage.

Prototype:
void
VLTD_SetVoltage(uint32_t ***Voltage***)

Parameters:
Voltage is the value detection voltage.
This parameter can be one of the following values:

- **VLTD_DETECT_VOLTAGE_41:** Detection voltage = 4.1V ± 0.2V
- **VLTD_DETECT_VOLTAGE_44:** Detection voltage = 4.4V ± 0.2V
- **VLTD_DETECT_VOLTAGE_46:** Detection voltage = 4.6V ± 0.2V

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

Return:
None

12.2.4 Data Structure Description

None

13. WDT

13.1 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\tmpm375_wdt.c, with \Libraries/TX03_Periph_Driver\inc\tmpm375_wdt.h containing the API definitions for use by applications.

13.2 API Functions

13.2.1 Function List

- void WDT_SetDetectTime(uint32_t ***DetectTime***)
- void WDT_SetIdleMode(FunctionalState ***NewState***)
- void WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)
- void WDT_Init(WDT_InitTypeDef * ***InitStruct***)
- void WDT_Enable(void)
- void WDT_Disable(void)
- void WDT_WriteClearCode(void)

13.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) The Watchdog Timer basic function are handled by the WDT_SetDetectTime(),WDT_SetOverflowOutput(), WDT_Init(), WDT_Enable(), WDT_Disable(), and WDT_WriteClearCode() functions.
2) Run or stop the WDT counter when enter IDLE mode is handled by the WDT_SetIdleMode().

13.2.3 Function Documentation

13.2.3.1 WDT_SetDetectTime

Set detection time for WDT.

Prototype:
void
WDT_SetDetectTime(uint32_t ***DetectTime***)

Parameters:
DetectTime**:** Set the detection time
This parameter can be one of the following values:

- **WDT_DETECT_TIME_EXP_15:** ***DetectTime*** is 2^15/fsys
- **WDT_DETECT_TIME_EXP_17:** ***DetectTime*** is 2^17/fsys
- **WDT_DETECT_TIME_EXP_19:** ***DetectTime*** is 2^19/fsys
- **WDT_DETECT_TIME_EXP_21:** ***DetectTime*** is 2^21/fsys
- **WDT_DETECT_TIME_EXP_23:** ***DetectTime*** is 2^23/fsys
- **WDT_DETECT_TIME_EXP_25:** ***DetectTime*** is 2^25/fsys

Description:
This function will set detection time for WDT.

Return:
None

13.2.3.2 WDT_SetIdleMode

Run or stop the WDT counter when the system enters IDLE mode.

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

Parameters:
NewState: Run or stop WDT counter.
This parameter can be one of the following values:

- **ENABLE**: Run the WDT counter.
- **DISABLE**. Stop the WDT counter.

Description:
This function will run the WDT counter when the system enters IDLE mode when ***NewState*** is **ENABLE**, and stop the WDT counter when the system enters IDLE mode when ***NewState*** is **DISABLE**.

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

Return:
None

13.2.3.3 WDT_SetOverflowOutput

Set WDT to generate NMI interrupt or reset when the counter overflows.

Prototype:
void
WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)

Parameters:
OverflowOutput: Select function of WDT when counter overflow.
This parameter can be one of the following values:

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

Description:

This function will set WDT to generate NMI interrupt if the counter overflows when ***OverflowOutput*** is **WDT_NMIINT,** and set WDT to generate reset if the counter overflows when ***OverflowOutput*** is **WDT_WDOUT**.

Return:
None

13.2.3.4 WDT_Init

Initialize and configure WDT.

Prototype:
void
WDT_Init (WDT_InitTypeDef* ***InitStruct***)

Parameters:
InitStruct: The structure containing basic WDT configuration including detect time and WDT output when counter overflow. (Refer to "Data structure Description" for details)

Description:
This function will initialize and configure the WDT detection time and the output of WDT when the counter overflows. **WDT_SetDetectTime()** and **WDT_SetOverflowOutput()** will be called by it.

Return:
None

13.2.3.5 WDT_Enable

Enable the WDT function.

Prototype:
void
WDT_Enable(void)

Parameters:
None

Description:
This function will enable WDT.

Return:
None

13.2.3.6 WDT_Disable

Disable the WDT function.

Prototype:
void
WDT_Disable(void)

Parameters:

None

Description:
This function will disable WDT.

Return:
None

13.2.3.7 WDT_WriteClearCode

Write the clear code.

Prototype:
void
WDT_WriteClearCode (void)

Parameters:
None

Description:
This function will clear the WDT counter.

Return:
None

13.2.4 Data Structure Description

13.2.4.1 WDT_InitTypeDef

Data Fields:

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

- **WDT_DETECT_TIME_EXP_15:** ***DetectTime*** is 2^15/fsys
- **WDT_DETECT_TIME_EXP_17:** ***DetectTime*** is 2^17/fsys
- **WDT_DETECT_TIME_EXP_19:** ***DetectTime*** is 2^19/fsys
- **WDT_DETECT_TIME_EXP_21:** ***DetectTime*** is 2^21/fsys
- **WDT_DETECT_TIME_EXP_23:** ***DetectTime*** is 2^23/fsys
- **WDT_DETECT_TIME_EXP_25:** ***DetectTime*** is 2^25/fsys

uint32_t
OverflowOutput Select the action when the WDT counter overflows, which can be set as:

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

14. PMD

14.1 Overview

The TMPM375FSDMG contains 1 channel programmable motor driver (PMD). The PMD of this product can control a three-phase motor such as vector motors in conjunction with a Vector Engine (VE+) and an analog/digital converter (ADC). Pulse-width modulation circuits, conduction control and synchronous trigger generators can be activated by commands from the Vector Engine. The synchronous trigger generation circuit can command the AD converter to start ADC conversion.

The PMD Module consists of two blocks of a wave generation circuit and a sync trigger generation circuit. The wave generation circuit includes a pulse width modulation circuit, a conduction control circuit, a protection control circuit, a dead time control circuit.

- The pulse width modulation circuit generates independent 3-phase PWM waveforms with the same PWM frequency.
- The conduction control circuit determines the output pattern for each of the upper and lower sides of the U, V and W phases.
- The protection control circuit controls emergency output stop by EMG input and OVV input.
- The dead time control circuit prevents a short circuit which may occur when the upper side and lower side are switched.
- The sync trigger generation circuit generates sync trigger signals to the AD converter.

This driver is contained in TX03_Periph_Driver\src\tmpm375_pmd.c, with TX03_Periph_Driver\inc\tmpm375_pmd.h containing the API definitions for use by applications.

14.2 API Functions

14.2.1 Function List

- void PMD_Enable(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_Disable(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetPortControl(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***PortMode***);
- void PMD_Init(TSB_PMD_TypeDef * ***PMDx***,
  PMD_InitTypeDef * ***InitStruct***);
- void PMD_ChangePWMCycle(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***CycleTiming***);
- uint32_t PMD_GetCntFlag(TSB_PMD_TypeDef * ***PMDx***);
- uint16_t PMD_GetCntValue(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetCompareValue(TSB_PMD_TypeDef * ***PMDx***,

uint32_t ***PMDPhase***,
uint32_t ***Timing***);

- void PMD_SetPortOutputMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Mode***);
- void PMD_SetOutputPhasePolarity(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***OutputPhase***,
  uint32_t ***Polarity***);
- void PMD_SetReflectTime(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***ReflectedTime***);
- void PMD_EnableEMG(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_DisableEMG(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetEMGNoiseElimination(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***NoiseElimination***);
- void PMD_SetToolBreakOutput(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Status***);
- void PMD_SetEMGMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Mode***);
- void PMD_EMGRelease(TSB_PMD_TypeDef * ***PMDx***);
- uint32_t PMD_GetEMGAbnormalLevel(TSB_PMD_TypeDef * ***PMDx***);
- uint32_t PMD_GetEMGCondition(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetDeadTime(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Time***);
- void PMD_SetAllPhaseCompareValue(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***UPhaseTiming***,
  uint32_t ***VPhaseTiming***,
  uint32_t ***WPhaseTiming***)
- void PMD_ChangeDutyMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***DutyMode***);
- Result PMD_SetPortOutput(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***PMDPhase***,
  uint8_t ***Output***);
- void PMD_SetTrgCmpValue(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***TRGCMP0Timing***,
  uint32_t ***TRGCMP1Timing***,
  uint32_t ***TRGCMP2Timing***,
  uint32_t ***TRGCMP3Timing***);
- void PMD_SetTrgMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***PMDTrg***,
  uint32_t ***Mode***);
- void PMD_SetTrgUpdate(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***PMDTrg***,
  uint32_t ***UpdateTiming***);
- void PMD_SetEMGTrg(TSB_PMD_TypeDef * ***PMDx***,
  FunctionalState ***NewState***);
- void PMD_SetTrgOutput(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***TrgMode***,
  uint32_t ***TrgChannel***);
- void PMD_SetSelectMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Mode***);
- void PMD_EnableOVV(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_DisableOVV(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetOVVNoiseElimination(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***NoiseElimination***);
- void PMD_SetADCMonitorInput(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Monitor***,
  FunctionalState ***NewState***);

- void PMD_SetOVVMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Mode***);
- void PMD_SetOVVInputSrc(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Source***);
- void PMD_SetOVVAutoRelease(TSB_PMD_TypeDef * ***PMDx***,
  FunctionalState ***NewState***);
- uint32_t PMD_GetOVVAbnormalLevel(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetAutoSwitchCtrl(TSB_PMD_TypeDef * ***PMDx***,
  FunctionalState ***NewState***);
- void PMD_SetPWMEdge(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***PMDPhase***,
  uint32_t ***Edge***);
- void PMD_SetBufferUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***UpdateTime***);
- void PMD_SetTrgSyncTime(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***SyncTime***);
- void PMD_SetTrgUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***UpdateTime***);

14.2.2 Detailed Description

Functions listed above can be divided into seven parts:

1) Common configuration and control of each PMD channel are handled by PMD_Enable(), PMD_Disable(), PMD_SetPortControl(), PMD_Init(), PMD_ChangePWMCycle(), PMD_SetCompareValue(), PMD_SetAllPhaseCompareValue(), PMD_ChangeDutyMode(),PMD_SetSelectMode(), PMD_SetAutoSwitchCtrl(),PMD_SetPWMEdge(), PMD_SetBufferUpdateTime().
2) PMD port output settings are handled by PMD_SetPortOutputMode(), PMD_SetOutputPhasePolarity(), PMD_SetReflectTime(), PMD_SetPortOutput().
3) PMD EMG functions are handled by PMD_EnableEMG(),PMD_DisableEMG(), PMD_SetEMGNoiseElimination(), PMD_SetToolBreakOutput(), PMD_SetEMGMode(), PMD_EMGRelease().
4) The status indication of each PMD channel is handled by PMD_GetCntFlag(), PMD_GetCntValue(), PMD_GetEMGAbnormalLevel(), PMD_GetEMGCondition(), PMD_GetOVVAbnormalLevel(),PMD_GetOVVCondition().
5) PMD dead time control is handled by PMD_SetDeadTime().
6) PMD ADC trigger generation circuit is handled by PMD_SetTrgCmpValue(), PMD_SetTrgMode(), PMD_SetTrgUpdate(), PMD_SetEMGTrg(),PMD_SetTrgOutput(),PMD_SetTrgSyncTime(), PMD_SetTrgUpdateTime().
7) PMD OVV functions are handled by PMD_EnableOVV(),PMD_DisableOVV(), PMD_SetOVVNoiseElimination(), PMD_SetADCMonitorInput(), PMD_SetOVVMode(), PMD_SetOVVAutoRelease(),PMD_SetOVVInputSrc().

14.2.3 Function Documentation

Note**: In all of the following APIs, parameter “TSB_PMD_TypeDef * ***PMDx” can be ***PMD1.***

14.2.3.1 PMD_Enable

Enable the specified PMD channel.

Prototype:

void
PMD_Enable (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will enable the specified PMD channel.

Return:
None

14.2.3.2 PMD_Disable

Disable the specified PMD channel.

Prototype:
void
PMD_Disable (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will disable the specified PMD channel.

Return:
None

14.2.3.3 PMD_SetPortControl

Set PMD port control of the specified PMD channel.

Prototype:
void
PMD_SetPortControl (TSB_PMD_TypeDef * ***PMDx***
uint32_t ***PortMode***)

Parameters:
PMDx: Select the PMD channel.

PortMode: The port output mode of PMD.
This parameter can be one of the following values:

- **PMD_PORT_MODE_0:** Upper phases = High-Z / Lower phases = High-Z
- **PMD_PORT_MODE_1:** Upper phases = High-Z / Lower phases = PMD output
- **PMD_PORT_MODE_2:** Upper phases = PMD output / Lower phases = High-Z
- **PMD_PORT_MODE_3:** Upper phases = PMD output / Lower phases = PMD output

Description:
This function will set PMD port control of the specified PMD channel.

Return:
None

14.2.3.4 PMD_Init

Initialize the specified PMD channel.

Prototype:
void
PMD_Init (TSB_PMD_TypeDef * ***PMDx***,
PMD_InitTypeDef * ***InitStruct***)

Parameters:
PMDx: Select the PMD channel.

InitStruct: The structure containing basic PMD configuration.
(Refer to “Data Structure Description” for details).

Description:
This function will initialize the specified PMD channel.

Return:
None

14.2.3.5 PMD_ChangePWMCycle

Change the PWM cycle of the specified PMD channel.

Prototype:
void
PMD_ChangePWMCycle (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***CycleTiming***)

Parameters:
PMDx: Select the PMD channel.

CycleTiming: PWM cycle, from 0x0000 to 0xFFFF.

Description:
This function will change the PWM cycle of the specified PMD channel.

Return:
None

***Note**:
If a value less than 0x10 is set, the register assumes 0x10 is set

14.2.3.6 PMD_GetCntFlag

Get the PWM counter flag of the specified PMD channel.

Prototype:
uint32_t
PMD_GetCntFlag (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will get the PWM counter flag of the specified PMD channel.

Return:
The PWM counter flag.
The value returned can be one of the following values:
PMD_COUNTER_UP: The PWM counter is up-counting
PMD_COUNTER_DOWN : The PWM counter is down-counting

14.2.3.7 PMD_GetCntValue

Get the count value of the specified PMD channel.

Prototype:
uint16_t
PMD_GetCntValue (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will get the count value of the specified PMD channel.

Return:
Count value of the specified PMD channel.

14.2.3.8 PMD_SetCompareValue

Set the compare value of the specified phase of the specified PMD channel.

Prototype:
void
PMD_SetCompareValue (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDPhase***,
uint32_t ***Timing***)

Parameters:
PMDx: Select the PMD channel.

PMDPhase: Select the phase of PMD channel.
This parameter can be one of the following values:
- **PMD_PHASE_U:** U-phase
- **PMD_PHASE_V:** V-phase
- **PMD_PHASE_W:** W-phase
- **PMD_PHASE_ALL:** All phases

Timing: Compare value, from 0x0000 to 0xFFFF.

Description:
This function will set the compare value of the specified phase of the specified PMD channel.

Return:
None

14.2.3.9 PMD_SetPortOutputMode

Set the mode of port output of the specified PMD channel.

Prototype:
void
PMD_SetPortOutputMode (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Mode***)

Parameters:
PMDx: Select the PMD channel.

Mode: The mode of port output.
This parameter can be one of the following values:
- **PMD_PORT_OUTPUT_MODE_0:** PMDxMDCR<SYNTMD>=0
- **PMD_PORT_OUTPUT_MODE_1:** PMDxMDCR<SYNTMD>=1

Description:
This function will set the mode of port output of the specified PMD channel.

***Note**:
PMDxMDCR<SYNTMD>, PMDxMDPOT<POLH><POLL>, PMDxMDOUT <UPWN><VPWN><WPWN> <UOC> <VOC> <WOC> set the port output. (x can be 1)

PMD_SetPortOutputMode() set PMDxMDCR<SYNTMD>.
PMD_SetOutputPhasePolarity() set PMDxMDPOT<POLH><POLL>.
PMD_SetPortOutput () set PMDxMDOUT<UPWN><VPWN> <WPWN> <UOC> <VOC> <WOC>.

The details about the port output is in the below diagram.

MTPDxMDCR<SYNTMD>=0

Polarity: high-active(MTPDxMDPOT<POLH><POLL>="11")

MDOUT output control		MTPDxMDOUT <WPWM><VPWM><UPWM> H/L/PWM output selection			
		0 : H/L output		1 : PWM output	
<WOC[1]> <VOC[1]> <UOC[1]> (Upper)	<WOC[0]> <VOC[0]> <UOC[0]> (Lower)	Upper output	Lower output	Upper output	Lower output
0	0	L	L	$\overline{\text{PWM}}$	PWM
0	1	L	H	L	PWM
1	0	H	L	PWM	L
1	1	H	H	PWM	$\overline{\text{PWM}}$

MTPDxMDCR<SYNTMD>=0

Polarity: low-active(MTPDxMDPOT<POLH><POLL>="00")

MDOUT output control		MTPDxMDOUT <WPWM><VPWM><UPWM> H/L/PWM output selection			
		0 : H/L output		1 : PWM output	
<WOC[1]> <VOC[1]> <UOC[1]> (Upper)	<WOC[0]> <VOC[0]> <UOC[0]> (Lower)	Upper output	Lower output	Upper output	Lower output
0	0	H	H	PWM	$\overline{\text{PWM}}$
0	1	H	L	H	$\overline{\text{PWM}}$
1	0	L	H	$\overline{\text{PWM}}$	H
1	1	L	L	$\overline{\text{PWM}}$	PWM

MTPDxMDCR<SYNTMD>=1

Polarity: high-active(MTPDxMDPOT<POLH><POLL>="11")

MDOUT output control		MTPDxMDOUT <WPWM><VPWM><UPWM> H/L/PWM output selection			
		0 : H/L output		1 : PWM output	
<WOC[1]> <VOC[1]> <UOC[1]> (Upper)	<WOC[0]> <VOC[0]> <UOC[0]> (Lower)	Upper output	Lower output	Upper output	Lower output
0	0	L	L	$\overline{\text{PWM}}$	PWM
0	1	L	H	L	$\overline{\text{PWM}}$
1	0	H	L	PWM	L
1	1	H	H	PWM	$\overline{\text{PWM}}$

MTPDxMDCR<SYNTMD>=1

Polarity: low-active(MTPDxMDPOT<POLH><POLL>="00")

MDOUT output control		MTPDxMDOUT <WPWM><VPWM><UPWM> H/L/PWM output selection			
		0 : H/L output		1 : PWM output	
<WOC[1]> <VOC[1]> <UOC[1]> (Upper)	<WOC[0]> <VOC[0]> <UOC[0]> (Lower)	Upper output	Lower output	Upper output	Lower output
0	0	H	H	PWM	$\overline{\text{PWM}}$
0	1	H	L	H	PWM
1	0	L	H	$\overline{\text{PWM}}$	H
1	1	L	L	$\overline{\text{PWM}}$	PWM

Return:
None

14.2.3.10 PMD_SetOutputPhasePolarity

Set polarity of the specified output port phase of the specified PMD channel.

Prototype:
void
PMD_SetOutputPhasePolarity (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***OutputPhase***,
uint32_t ***Polarity***)

Parameters:
PMDx: Select the PMD channel.

OutputPhase: Select the specified output port phase.

- **PMD_OUTPUT_PHASE_UPPER:** Upper phase output port
- **PMD_OUTPUT_PHASE_LOWER:** Lower phase output port

Polarity: The value of the bit.

- **PMD_POLARITY_LOW:** Low active
- **PMD_POLARITY_HIGH:** High active

Description:
This function will set polarity of the specified output port phase of the specified PMD channel.

***Note**:
1 Refer to function PMD_SetPortOutputMode() for more details.
2 When calling this function, the PMD must be disabled.

Return:
None

14.2.3.11 PMD_SetReflectTime

Choose the timing when port outputs of U-, V- and W- phase output setting is reflected of the specified PMD channel.

Prototype:
void
PMD_SetReflectTime (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***ReflectedTime***)

Parameters:
PMDx: Select the PMD channel.

ReflectedTime: Select the reflected time.

- **PMD_REFLECTED_TIME_WRITE:** Reflect when write
- **PMD_REFLECTED_TIME_MIN:** Reflect when PWM counter MDCNT= "1"(minimum)
- **PMD_REFLECTED_TIME_MAX:** Reflect when PWM counter MDCNT= PMDxMDPRD<MDPRD>(maximum)
- **PMD_REFLECTED_TIME_MIN_MAX:** Reflect when PWM counter MDCNT="1"(minimum) or PMDxMDPRD<MDPRD>(maximum)

Description:
This function will choose the timing when port outputs of U-, V- and W- phase output setting is reflected of the specified PMD channel.

***Note**:
When calling this function, the PMD must be disabled.

Return:
None

14.2.3.12 PMD_EnableEMG

Enable EMG protection of the specified PMD channel.

Prototype:
void
PMD_EnableEMG (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will enable EMG protection of the specified PMD channel.

Return:

None

14.2.3.13 PMD_DisableEMG

Disable EMG protection of the specified PMD channel.

Prototype:
void
PMD_DisableEMG (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will disable EMG protection of the specified PMD channel.

Return:
None

14.2.3.14 PMD_SetEMGNoiseElimination

Set the noise elimination time for abnormal condition detection input of the specified PMD channel.

Prototype:
void
PMD_SetEMGNoiseElimination (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***NoiseElimination***)

Parameters:
PMDx: Select the PMD channel.

NoiseElimination: Select the noise elimination time.

- **PMD_NOISE_ELIMINATION_NONE:** Noise filter is not used
- **PMD_NOISE_ELIMINATION_16:** Input noise elimination time 16/fsys[s]
- **PMD_NOISE_ELIMINATION_32:** Input noise elimination time 32/fsys[s]
- **PMD_NOISE_ELIMINATION_48:** Input noise elimination time 48/fsys[s]
- **PMD_NOISE_ELIMINATION_64:** Input noise elimination time 64/fsys[s]
- **PMD_NOISE_ELIMINATION_80:** Input noise elimination time 80/fsys[s]
- **PMD_NOISE_ELIMINATION_96:** Input noise elimination time 96/fsys[s]
- **PMD_NOISE_ELIMINATION_112:** Input noise elimination time 112/fsys[s]
- **PMD_NOISE_ELIMINATION_128:** Input noise elimination time 128/fsys[s]
- **PMD_NOISE_ELIMINATION_144:** Input noise elimination time 144/fsys[s]
- **PMD_NOISE_ELIMINATION_160:** Input noise elimination time 160/fsys[s]
- **PMD_NOISE_ELIMINATION_176:** Input noise elimination time 176/fsys[s]
- **PMD_NOISE_ELIMINATION_192:** Input noise elimination time 192/fsys[s]
- **PMD_NOISE_ELIMINATION_208:** Input noise elimination time 208/fsys[s]
- **PMD_NOISE_ELIMINATION_224:** Input noise elimination time 224/fsys[s]
- **PMD_NOISE_ELIMINATION_240:** Input noise elimination time 240/fsys[s]

Description:
This function will set the noise elimination time for abnormal condition detection input of the specified PMD channel.

Return:

None

14.2.3.15 PMD_SetToolBreakOutput

Choose PMD output status at tool break of the specified PMD channel.

Prototype:
void
PMD_SetToolBreakOutput (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Status***)

Parameters:
PMDx: Select the PMD channel.

Status: PMD output status at tool break.

- **PMD_BREAK_STATUS_PMD:** PMD output is continued
- **PMD_BREAK_STATUS_HIGH_IMPEDANCE:** High-impedance

Description:
This function will choose PMD output status at tool break of the specified PMD channel.

Return:
None

14.2.3.16 PMD_SetEMGMode

Set EMG protection mode of the specified PMD channel.

Prototype:
void
PMD_SetEMGMode (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Mode***)

Parameters:
PMDx: Select the PMD channel.

Mode: EMG protection mode.

- **PMD_EMG_MODE_0:** All phases High-Z.
- **PMD_EMG_MODE_1:** All upper-phase ON / all lower-phase High-Z.
- **PMD_EMG_MODE_2:** All upper phase High-Z / all lower phase ON.
- **PMD_EMG_MODE_3:** All phase High-Z.

Description:
This function will set EMG protection mode of the specified PMD channel.

Return:
None

14.2.3.17 PMD_EMGRelease

Release EMG protection status of the specified PMD channel.

Prototype:

void
PMD_EMGRelease (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will release EMG protection status of the specified PMD channel

***Note**:
PMDxMDOUT<UPWN><VPWN><WPWN> and PMDxMDOUT<UOC> <VOC> <WOC> will be cleared to 0 after the function be called. (x can be 1)

Return:
None

14.2.3.18 PMD_GetEMGAbnormalLevel

Get the level of abnormal condition input of the specified PMD channel.

Prototype:
uint32_t
PMD_GetEMGAbnormalLevel (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will get the level of abnormal condition input of the specified PMD channel.

Return:
The level of abnormal condition input.
The value returned can be one of the following values:
PMD_ABNORMAL_LEVEL_L: Abnormal condition input level is "L"
PMD_ABNORMAL_LEVEL_H : Abnormal condition input level is "H"

14.2.3.19 PMD_GetEMGCondition

Get the EMG protection condition of the specified PMD channel.

Prototype:
uint32_t
PMD_GetEMGCondition (TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will get the EMG protection condition of the specified PMD channel.

Return:
The EMG protection condition.
The value returned can be 0 or 1.
0 means normal operation.

1 means during in EMG protection.

14.2.3.20 PMD_SetDeadTime

Set dead time of the specified PMD channel.

Prototype:

```
void
PMD_SetDeadTime (TSB_PMD_TypeDef * PMDx,
                 uint32_t Time)
```

Parameters:

PMDx: Select the PMD channel.

Time: Dead time, from 0x00 to 0xFF.

Description:

This function will set dead time of the specified PMD channel.

***Note**:*

When calling this function, the PMD must be disabled.

Return:

None

14.2.3.21 PMD_SetAllPhaseCompareValue

Set the compare values of the all phases of the specified PMD channel.

Prototype:

```
void
PMD_SetAllPhaseCompareValue (TSB_PMD_TypeDef * PMDx,
                             uint32_t UPhaseTiming,
                             uint32_t VPhaseTiming,
                             uint32_t WPhaseTiming)
```

Parameters:

PMDx: Select the PMD channel.

UPhaseTiming: Compare value of phase U, from 0x0000 to 0xFFFF.

VPhaseTiming: Compare value of phase V, from 0x0000 to 0xFFFF.

WPhaseTiming: Compare value of phase W, from 0x0000 to 0xFFFF.

Description:

This function will set the compare values of the all phases of the specified PMD channel.

Return:

None

14.2.3.22 PMD_ChangeDutyMode

Change duty mode of the specified PMD channel.

Prototype:
void
PMD_ChangeDutyMode (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***DutyMode***)

Parameters:
PMDx: Select the PMD channel.

DutyMode: The duty mode of PMD.

- **PMD_DUTY_MODE_U_PHASE:** U-phase in common
- **PMD_DUTY_MODE_3_PHASE:** 3-phase independent

Description:
This function will change duty mode of the specified PMD channel.

Return:
None

14.2.3.23 PMD_SetPortOutput

Set the specified output of the specified phase of the specified PMD channel.

Prototype:
Result
PMD_SetPortOutput (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDPhase***,
uint8_t ***Output***)

Parameters:
PMDx: Select the PMD channel.

PMDPhase: Select the phase of PMD channel.
This parameter can be one of the following values:

- **PMD_PHASE_U:** U-phase
- **PMD_PHASE_V:** V-phase
- **PMD_PHASE_W:** W-phase
- **PMD_PHASE_ALL:** All phases

Output: Select the output.
This parameter can be one of the following values:

- **PMD_OUTPUT_L_L:** Upper output is L, Lower output is L.
- **PMD_OUTPUT_L_H:** Upper output is L, Lower output is H.
- **PMD_OUTPUT_H_L:** Upper output is H, Lower output is L.
- **PMD_OUTPUT_H_H:** Upper output is H, Lower output is H.
- **PMD_OUTPUT_PWM_IPWM:**Upper output is PWM, Lower output is IPWM.
- **PMD_OUTPUT_IPWM_PWM:**Upper output is IPWM, Lower output is PWM.
- **PMD_OUTPUT_H_PWM:**Upper output is H, Lower output is PWM.
- **PMD_OUTPUT_L_PWM:**Upper output is L, Lower output is PWM.
- **PMD_OUTPUT_PWM_L:**Upper output is PWM, Lower output is L.
- **PMD_OUTPUT_H_IPWM:**Upper output is H, Lower output is IPWM.
- **PMD_OUTPUT_L_IPWM:**Upper output is L, Lower output is IPWM.
- **PMD_OUTPUT_IPWM_H:**Upper output is IPWM, Lower output is H.

Description:
This function will set the specified output of the specified phase of the specified PMD channel.

Return:
Success or not
The value returned can be one of the following values:
SUCCESS: PMD output is set successfully.
ERROR: PMD output setting is failed.

***Note**:
1. IPWM is the inverting PWM.
2. Refer to function PMD_SetPortOutputMode() for details.

14.2.3.24 PMD_SetTrgCmpValue

Set the ADC trigger compare registers' value of the specified PMD channel.

Prototype:
void
PMD_SetTrgCmpValue(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***TRGCMP0Timing***,
uint32_t ***TRGCMP1Timing***,
uint32_t ***TRGCMP2Timing***,
uint32_t ***TRGCMP3Timing***)

Parameters:
PMDx: Select the PMD channel.

TRGCMP0Timing: Value of ADC trigger compare register 0, from 0x0001 to [MDPRD set value – 1].

TRGCMP1Timing: Value of ADC trigger compare register 1, from 0x0001 to [MDPRD set value – 1].

TRGCMP2Timing: Value of ADC trigger compare register 2, from 0x0001 to [MDPRD set value – 1].

TRGCMP3Timing: Value of ADC trigger compare register 3, from 0x0001 to [MDPRD set value – 1].

Description:
This function will set the ADC trigger compare registers' value of the specified PMD channel.

Return:
None

***Note**: PMDnTRGCMPx (x can be 1) should be set in a range of 1 to [MDPRD set value – 1].

14.2.3.25 PMD_SetTrgMode

Set trigger mode of the specified PMD channel.

Prototype:

```
void
PMD_SetTrgMode (TSB_PMD_TypeDef * PMDx,
                uint32_t PMDTrg,
                uint32_t Mode)
```

Parameters:

PMDx: Select the PMD channel.

PMDTrg: Select the PMD Trigger.
This parameter can be one of the following values:
- **PMD_ADC_TRG_0:** Select trigger 0
- **PMD_ADC_TRG_1:** Select trigger 1

Mode: PMD trigger mode.
This parameter can be one of the following values:
- **PMD_TRG_MODE_0:** Trigger output disabled
- **PMD_TRG_MODE_1:** Trigger output at down-count match
- **PMD_TRG_MODE_2:** Trigger output at up-count match
- **PMD_TRG_MODE_3:** Trigger output at up-/down-count match
- **PMD_TRG_MODE_4:** Trigger output at PWM carrier peak
- **PMD_TRG_MODE_5:** Trigger output at PWM carrier bottom
- **PMD_TRG_MODE_6:** Trigger output at PWM carrier peak/bottom
- **PMD_TRG_MODE_7:** Trigger output disabled

Description:

This function will set trigger mode of the specified PMD channel.

Return:

None

14.2.3.26 PMD_SetTrgUpdate

Set trigger buffer update timing of the specified PMD channel.

Prototype:

```
void
PMD_SetTrgUpdate (TSB_PMD_TypeDef * PMDx,
                  uint32_t PMDTrg,
                  uint32_t UpdateTiming)
```

Parameters:

PMDx: Select the PMD channel.

PMDTrg: Select the PMD Trigger.
This parameter can be one of the following values:
- **PMD_ADC_TRG_0:** Select trigger 0
- **PMD_ADC_TRG_1:** Select trigger 1
- **PMD_ADC_TRG_2:** Select trigger 2
- **PMD_ADC_TRG_3:** Select trigger 3

Mode: PMDTRG0 to PMDTRG1 buffer update timing.
This parameter can be one of the following values:
- **PMD_TRG_UPDATE_SYNC:** Sync to PWM

- **PMD_TRG_UPDATE_ASYNC:** The value written to PMDTRGx is immediately reflected

Description:
This function will set trigger buffer update timing of the specified PMD channel.

Return:
None

14.2.3.27 PMD_SetEMGTrg

Enable or disable trigger output in EMG protection state of the specified PMD channel.

Prototype:
void
PMD_SetEMGTrg (TSB_PMD_TypeDef * ***PMDx***,
FunctionalState ***NewState***)

Parameters:
PMDx: Select the PMD channel.

NewState: Output enable in EMG protection state.

- **ENABLE:** Enable trigger output in the protection state
- **DIABLE:** Disable trigger output in the protection state

Description:
This function will enable or disable trigger output in EMG protection state of the specified PMD channel.

Return:
None

14.2.3.28 PMD_SetTrgOutput

Set trigger output of the specified PMD channel.

Prototype:
void
PMD_SetTrgOutput(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***TrgMode***,
uint32_t ***TrgChannel***);

Parameters:
PMDx: Select the PMD channel.

TrgMode: Select the trigger output mode.

- **PMD_TRG_FIXED_OUTPUT:** Fixed trigger output
- **PMD_TRG_VARIABLE_OUTPUT:** Fixed trigger output

TrgChannel: Trigger output select.
when ***TrgMode*** == **PMD_TRG_FIXED_OUTPUT**:

- **PMD_TRG_OUTPUT_0:** Output from PMDTRG0
- **PMD_TRG_OUTPUT_1:** Output from PMDTRG1
- **PMD_TRG_OUTPUT_2:** Output from PMDTRG2

- **PMD_TRG_OUTPUT_3:** Output from PMDTRG3

when ***TrgMode*** == **PMD_TRG_VARIABLE_OUTPUT**:

- **PMD_TRG_OUTPUT_0:** Output from PMDTRG0
- **PMD_TRG_OUTPUT_1:** Output from PMDTRG1
- **PMD_TRG_OUTPUT_2:** Output from PMDTRG2
- **PMD_TRG_OUTPUT_3:** Output from PMDTRG3
- **PMD_TRG_OUTPUT_4:** Output from PMDTRG4
- **PMD_TRG_OUTPUT_5:** Output from PMDTRG5

Description:
This function will set trigger output of the specified PMD channel.

Return:
None

14.2.3.29 PMD_SetSelectMode

Set the select mode of the specified PMD channel.

Prototype:
void
PMD_SetSelectMode(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Mode***);

Parameters:
PMDx: Select the PMD channel.

Mode: The select mode of PMD.

- **PMD_BUS_MODE:** Load the second buffer of each double-buffered register with the register value set via the bus (bus mode)
- **PMD_VE_MODE:** Load the second buffer of each double-buffered register with the register value set from the Vector Engine (VE mode)

Description:
This function will set the select mode of the specified PMD channel.

Return:
None

14.2.3.30 PMD_EnableOVV

Enable OVV protection of the specified PMD channel.

Prototype:
void
PMD_EnableOVV(TSB_PMD_TypeDef * ***PMDx***);

Parameters:
PMDx: Select the PMD channel.

Description:
This function will enable OVV protection of the specified PMD channel.

Return:
None

14.2.3.31 PMD_DisableOVV

Disable OVV protection of the specified PMD channel.

Prototype:
void
PMD_DisableOVV(TSB_PMD_TypeDef * ***PMDx***);

Parameters:
PMDx: Select the PMD channel.

Description:
This function will disable OVV protection of the specified PMD channel.

Return:
None

14.2.3.32 PMD_SetOVVNoiseElimination

Set the noise elimination time for abnormal condition detection input of the specified PMD channel.

Prototype:
void
PMD_SetOVVNoiseElimination(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***NoiseElimination***)

Parameters:
PMDx: Select the PMD channel.

NoiseElimination: Select the noise elimination time.

- **PMD_NOISE_ELIMINATION_16:** Input noise elimination time 16/fsys[s]
- **PMD_NOISE_ELIMINATION_32:** Input noise elimination time 32/fsys[s]
- **PMD_NOISE_ELIMINATION_48:** Input noise elimination time 48/fsys[s]
- **PMD_NOISE_ELIMINATION_64:** Input noise elimination time 64/fsys[s]
- **PMD_NOISE_ELIMINATION_80:** Input noise elimination time 80/fsys[s]
- **PMD_NOISE_ELIMINATION_96:** Input noise elimination time 96/fsys[s]
- **PMD_NOISE_ELIMINATION_112:** Input noise elimination time 112/fsys[s]
- **PMD_NOISE_ELIMINATION_128:** Input noise elimination time 128/fsys[s]
- **PMD_NOISE_ELIMINATION_144:** Input noise elimination time 144/fsys[s]
- **PMD_NOISE_ELIMINATION_160:** Input noise elimination time 160/fsys[s]
- **PMD_NOISE_ELIMINATION_176:** Input noise elimination time 176/fsys[s]
- **PMD_NOISE_ELIMINATION_192:** Input noise elimination time 192/fsys[s]
- **PMD_NOISE_ELIMINATION_208:** Input noise elimination time 208/fsys[s]
- **PMD_NOISE_ELIMINATION_224:** Input noise elimination time 224/fsys[s]
- **PMD_NOISE_ELIMINATION_240:** Input noise elimination time 240/fsys[s]

Description:
This function will set the noise elimination time for abnormal condition detection input of the specified PMD channel.

Return:
None

14.2.3.33 PMD_SetADCMonitorInput

Enable or disable ADC monitor interrupt input of OVV protection of the specified PMD channel.

Prototype:

```
void
PMD_SetADCMonitorInput(TSB_PMD_TypeDef * PMDx,
                       uint32_t Monitor,
                       FunctionalState NewState);
```

Parameters:

PMDx: Select the PMD channel.

Monitor: Select the ADC monitor for OVV protection.

- **PMD_ADC_MONITOR_A:** ADC A monitor interrupt input
- **PMD_ADC_MONITOR_B:** ADC B monitor interrupt input

NewState: ADC monitor interrupt input enable in OVV protection state.

- **ENABLE:** Enable ADC monitor interrupt input in the protection state
- **DIABLE:** Disable ADC monitor interrupt input in the protection state

Description:

This function will enable or disable ADC monitor interrupt input of OVV protection of the specified PMD channel.

Return:

None

14.2.3.34 PMD_SetOVVMode

Set OVV protection mode of the specified PMD channel.

Prototype:

```
void
PMD_SetOVVMode(TSB_PMD_TypeDef * PMDx,
               uint32_t Mode);
```

Parameters:

PMDx: Select the PMD channel.

Mode: OVV protection mode.

- **PMD_OVV_MODE_0:** No output control
- **PMD_OVV_MODE_1:** All upper phases ON, all lower phases OFF
- **PMD_OVV_MODE_2:** All upper phases OFF, all lower phases ON
- **PMD_OVV_MODE_3:** All phases OFF (ON = High, OFF = Low [when <POLL>,<POLH> = 1 (active high)])

Description:

This function will set OVV protection mode of the specified PMD channel.

Return:

None

14.2.3.35 PMD_SetOVVInputSrc

Set OVV input source of the specified PMD channel.

Prototype:
void
PMD_SetOVVInputSrc(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Source***);

Parameters:
PMDx: Select the PMD channel.

Source: OVV input source.
- **PMD_OVV_PORT_INPUT:** Use port input as the OVV input signal
- **PMD_OVV_ADC_MONITOR:** Use ADC monitor signal as the OVV input signal

Description:
This function will set OVV input source of the specified PMD channel.

Return:
None

14.2.3.36 PMD_SetOVVAutoRelease

Enable or disable automatic release of OVV protection of the specified PMD channel.

Prototype:
void
PMD_SetOVVAutoRelease(TSB_PMD_TypeDef * ***PMDx***,
FunctionalState ***NewState***);

Parameters:
PMDx: Select the PMD channel.

NewState: Automatic release enable in OVV protection state.
- **ENABLE:** Enable trigger automatic releasein the protection state
- **DIABLE:** Disable trigger automatic releasein the protection state

Description:
This function will enable or disable automatic release of OVV protection of the specified PMD channel.

Return:
None

14.2.3.37 PMD_GetOVVAbnormalLevel

Get the level of abnormal condition OVV input of the specified PMD channel.

Prototype:
uint32_t
PMD_GetOVVAbnormalLevel(TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will get the level of abnormal condition OVV input of the specified PMD channel.

Return:
The level of abnormal condition input.
The value returned can be one of the following values:
PMD_ABNORMAL_LEVEL_L: Abnormal condition input level is "L"
PMD_ABNORMAL_LEVEL_H : Abnormal condition input level is "H"

14.2.3.38 PMD_GetOVVCondition

Get the OVV protection condition of the specified PMD channel.

Prototype:
uint32_t
PMD_GetOVVCondition(TSB_PMD_TypeDef * ***PMDx***)

Parameters:
PMDx: Select the PMD channel.

Description:
This function will get the OVV protection condition of the specified PMD channel.

Return:
The OVV protection condition.
The value returned can be one of the following values:
PMD_OVV_NORMAL: OVV protection control circuit is in normal operation
PMD_OVV_PROTECTED: OVV protection control circuit is in protection

14.2.3.39 PMD_SetAutoSwitchCtrl

Enable or disable automatic switching between VE register and PMD register of the specified PMD channel.

Prototype:
void
PMD_SetAutoSwitchCtrl(TSB_PMD_TypeDef * ***PMDx***,
FunctionalState ***NewState***);

Parameters:
PMDx: Select the PMD channel.

NewState: Switching control state.

- **ENABLE:** Enable switching control state
- **DIABLE:** Disable switching control state

Description:
This function will enable or disable automatic switching between VE register and PMD register of the specified PMD channel.

Return:

None

14.2.3.40 PMD_SetPWMEdge

Set PWM edge of the specified phase of the specified PMD channel.

Prototype:
void
PMD_SetPWMEdge(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDPhase***,
uint32_t ***Edge***);

Parameters:
PMDx: Select the PMD channel.

PMDPhase: Select the phase of PMD channel.
This parameter can be one of the following values:

- **PMD_PHASE_U:** U-phase
- **PMD_PHASE_V:** V-phase
- **PMD_PHASE_W:** W-phase
- **PMD_PHASE_ALL:** All phases

Edge: Select the phase edge.

- **PMD_PWM_EDGE_UNFIXED:** Edge unfixed.
- **PMD_PWM_RISING_EDGE_FIXED:** PWM rising-edge fixed.
- **PMD_PWM_FALLING_EDGE_FIXED:** PWM falling-edge fixed.

Description:
This function will set PWM edge of the specified phase of the specified PMD channel.

Return:
None

14.2.3.41 PMD_SetBufferUpdateTime

Choose the double buffer update timing of the specified PMD channel.

Prototype:
void
PMD_SetBufferUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***UpdateTime***);

Parameters:
PMDx: Select the PMD channel.

UpdateTime: Select the update time..

- **PMD_UPDATE_TIME_WRITE:** Depends on write setting
- **PMD_UPDATE_TIME_MIN:** Updates at PWM carrier bottom
- **PMD_UPDATE_TIME_MAX:** Updates at PWM carrier peak
- **PMD_UPDATE_TIME_MIN_MAX:** Updates at both PWM carrier peak and bottom

Description:

This function will choose the double buffer update timing of the specified PMD channel.

Return:
None

14.2.3.42 PMD_SetTrgSyncTime

Choose the trigger synchronous timing of the specified PMD channel.

Prototype:
void
PMD_SetTrgSyncTime (TSB_PMD_TypeDef * ***PMDx***,
uint32_t***SyncTime***);

Parameters:
PMDx: Select the PMD channel.

SyncTime: Select the synchronous time.

- **PMD_TRGGER_TIME_ASYNC:** Transfer timing is asynchronous
- **PMD_TRGGER_TIME_ENC:** Selects transfer timing when INTENC (ENC interrupt request) occurs
- **PMD_TRGGER_TIME_TMRB:** Selects transfer timing when INTTB00 (TMRB interrupt request) occurs

Description:
This function will choose the trigger synchronous timing of the specified PMD channel.

Return:
None

14.2.3.43 PMD_SetTrgUpdateTime

Choose the trigger buffer update timing of the specified PMD channel.

Prototype:
void
PMD_SetTrgUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***UpdateTime***);

Parameters:
PMDx: Select the PMD channel.

UpdateTime: Select the update time..

- **PMD_UPDATE_TIME_WRITE:** Depends on write setting
- **PMD_UPDATE_TIME_MIN:** Updates at PWM carrier bottom
- **PMD_UPDATE_TIME_MAX:** Updates at PWM carrier peak
- **PMD_UPDATE_TIME_MIN_MAX:** Updates at both PWM carrier peak and bottom

Description:
This function will choose the trigger buffer update timing of the specified PMD channel.

Return:
None

14.2.4 Data Structure Description

14.2.4.1 PMD_InitTypeDef

Data Fields:

uint32_t

CycleMode: Specify PWM cycle extension mode, which can be:

- **PMD_PWM_NORMAL_CYCLE:** Normal cycle
- **PMD_PWM_4_FOLD_CYCLE:** 4-fold cycle

uint32_t

DutyMode: Choose DUTY mode, which can be:

- **PMD_DUTY_MODE_U_PHASE:** U-phase in common
- **PMD_DUTY_MODE_3_PHASE:** 3-phase independent

uint32_t

IntTiming: Choose PWM interrupt timing when PWM mode 1 (triangle wave) is set, which can be:

- **PMD_PWM_INT_TIMING_MINIMUM:** When PWM count MDCNT="1" is set, (minimum) interrupt request occurs
- **PMD_PWM_INT_TIMING_MAXIMUM:** When PWM count MDCNT= PMDxMDPRD<MDPRD> is set, (maximum) interrupt request **occurs**

uint32_t

IntCycle: Choose PWM interrupt cycle, which can be:

- **PMD_PWM_INT_CYCLE_HALF:** Every PWM 0.5 cycle (can be set in PWM mode1 (triangle wave)
- **PMD_PWM_INT_CYCLE_1:** Every PWM 1 cycle
- **PMD_PWM_INT_CYCLE_2:** Every PWM 2 cycles
- **PMD_PWM_INT_CYCLE_4:** Every PWM 4 cycles

uint32_t

CarrierMode: Specify PWM carrier wave, which can be:

- **PMD_CARRIER_WAVE_MODE_0:** PWM mode 0 (edge **PWM**, sawtooth)
- **PMD_CARRIER_WAVE_MODE_1:** PWM mode 1 (center PWM, triangle wave)

uint32_t

CycleTiming: Set PWM cycle, which can be 0x0000 to 0xFFFF

***Note**:
If a value less than 0x10 is set, the register assumes 0x10 is set