

# **TOSHIBA**

## **TOSHIBA TX03 Peripheral Driver User Guide (TMPM37AFSQG)**

Ver 1  
Sep, 2017

**TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION**

CMDR-M37AUG-01E

## **RESTRICTIONS ON PRODUCT USE**

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

---

## Index

<b>Index .....</b>	<b>2</b>
<b>1. Introduction .....</b>	<b>1</b>
<b>2. Organization of TOSHIBA TX03 Peripheral Driver.....</b>	<b>1</b>
<b>3. ADC .....</b>	<b>2</b>
3.1 Overview .....	2
3.2 API Functions .....	2
3.2.1 Function List .....	2
3.2.2 Detailed Description.....	3
3.2.3 Function Documentation .....	3
3.2.4 Data Structure Description .....	12
<b>4. CG.....</b>	<b>15</b>
4.1 Overview .....	15
4.2 API Functions .....	15
4.2.1 Function List .....	15
4.2.2 Detailed Description.....	16
4.2.3 Function Documentation .....	16
4.2.4 Data Structure Description .....	28
<b>5. FC .....</b>	<b>30</b>
5.1 Overview .....	30
5.2 API Functions .....	30
5.2.1 Function List .....	30
5.2.2 Detailed Description.....	30
5.2.3 Function Documentation .....	31
5.2.4 Data Structure Description .....	35
<b>6. GPIO.....</b>	<b>35</b>
6.1 Overview .....	35
6.2 API Functions .....	35
6.2.1 Function List .....	35
6.2.2 Detailed Description.....	36
6.2.3 Function Documentation .....	36
6.2.4 Data Structure Description .....	46
<b>7. SBI .....</b>	<b>48</b>
7.1 Overview .....	48
7.2 API Functions .....	48
7.2.1 Function List .....	48
7.2.2 Detailed Description.....	48
7.2.3 Function Documentation .....	49
7.2.4 Data Structure Description .....	54
<b>8. OFD .....</b>	<b>57</b>
8.1 Overview .....	57
8.2 API Functions .....	57
8.2.1 Function List .....	57
8.2.2 Detailed Description.....	57
8.2.3 Function Documentation .....	57
8.2.4 Data Structure Description .....	59
<b>9. TMRB .....</b>	<b>60</b>
9.1 Overview .....	60
9.2 API Functions .....	60
9.2.1 Function List .....	60
9.2.2 Detailed Description.....	61
9.2.3 Function Documentation .....	61
9.2.4 Data Structure Description .....	69
<b>10. SIO/UART.....</b>	<b>72</b>

10.1	Overview .....	72
10.2	API Functions .....	72
10.2.1	Function List .....	72
10.2.2	Detailed Description.....	73
10.2.3	Function Documentation .....	73
10.2.4	Data Structure Description .....	87
<b>11.</b>	<b>VLTD .....</b>	<b>90</b>
11.1	Overview .....	90
11.2	API Functions .....	90
11.2.1	Function List .....	90
11.2.2	Detailed Description.....	90
11.2.3	Function Documentation .....	90
11.2.4	Data Structure Description .....	91
<b>12.</b>	<b>WDT.....</b>	<b>92</b>
12.1	Overview .....	92
12.2	API Functions .....	92
12.2.1	Function List .....	92
12.2.2	Detailed Description.....	92
12.2.3	Function Documentation .....	92
12.2.4	Data Structure Description .....	95
<b>13.</b>	<b>PMD.....</b>	<b>96</b>
13.1	Overview .....	96
13.2	API Functions .....	96
13.2.1	Function List .....	96
13.2.2	Detailed Description.....	98
13.2.3	Function Documentation .....	98
13.2.4	Data Structure Description .....	120

## 1. Introduction

TOSHIBA TX03 Peripheral Driver is a set of drivers for all peripherals found on the TOSHIBA TX03 series microcontrollers. TMPM37AFSQG Peripheral Driver is an important part of TOSHIBA TX03 Peripheral Driver, which is designed for TMPM37AFSQG series MCUs.

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

The design goals of TOSHIBA TMPM37AFSQG 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 TMPM37AFSQG Peripheral Drivers.

### **/Libraries/ TX03\_CMSIS**

This folder contains the device peripheral access layer of TMPM37AFSQG CMSIS files.

### **/Libraries/TX03\_Periph\_Driver**

This folder contains all the source code of the drivers, the core of TOSHIBA TMPM37AFSQG Peripheral Driver.

### **/Libraries/TX03\_Periph\_Driver/inc**

This folder contains all the header files of TMPM37AFSQG Peripheral Drivers for each peripheral.

### **/Libraries/TX03\_Periph\_Driver/src**

This folder contains all the source files of TMPM37AFSQG Peripheral Drivers for each peripheral.

### **/Project**

This folder contains template project and examples for using TMPM37AFSQG Peripheral Driver.

### **/Project/Template**

This folder contains template project of TOSHIBA TMPM37AFSQG Peripheral Driver.

### **/Project/Examples**

This folder contains a set of examples for using TMPM37AFSQG Peripheral Driver

### **/Utilities/TMPM37A-EVAL**

This folder contains the configuration and driver files for hardware resources (e.g. led, key) on TMPM37AFSQG boards.

## 3. ADC

### 3.1 Overview

TOSHIBA TMPM37AFSQG 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 TMPM37AFSQG 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\tmpm37a\_adc.c, with TX03\_Periph\_Driver\inc\tmpm37a\_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**,

- ◆ 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 PMD<sub>x</sub>. (x = 6 to 11).

This parameter must be inputted with macro as the format below:

- **TRG\_ENABLE(PMD\_PROG<sub>y</sub>):** Enable PMD Trigger Program Select Register with Program y. (**y = 0 to 5**)
- **TRG\_DISABLE(PMD\_PROG<sub>y</sub>):** 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 **SEL<sub>x</sub>**, 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\_CMPCR<sub>x</sub>

**CMPCR<sub>x</sub>** Select Compare Control Register.

which can be:

- **ADC\_CMPCR\_0:** ADxCMPCR0
- **ADC\_CMPCR\_1:** ADxCMPCR1

ADC\_REG<sub>x</sub>

**ResultREG<sub>x</sub>** 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** Condition to compare ADxREG<sub>m</sub> with ADxCMP<sub>n</sub>. (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 TPM37AFSQG 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\tpm37a\_cg.c, with TX03\_Periph\_Driver\incl\tpm37a\_cg.h containing the API definitions for use by applications.

The following symbols fosc, fosc1, fosc2, fpll, fc, fgear, fsys, fperiph,  $\Phi$ T0 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>.  
 **$\Phi$ T0** : 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 2<sup>nd</sup> 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( $\Phi T0$ ) 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( $\Phi T0$ ) source.

**Return:**  
None

## 4.2.3.4 CG\_GetPhiT0Src

Get the PhiT0 ( $\Phi T0$ ) source.

**Prototype:**

CG\_PhiT0Src

CG\_GetPhiT0Src (void)

**Parameters:**

None

**Description:**

This function will get the PhiT0( $\Phi$ T0) source.

**Return:**

**CG\_PHIT0\_SRC\_FGEAR** means PhiT0 source is fgear.

**CG\_PHIT0\_SRC\_FC** means PhiT0 source is fc.

#### 4.2.3.5 CG\_SetPhiT0Level

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

**Prototype:**

Result

CG\_SetPhiT0Level (CG\_DivideLevel ***DividePhiT0FromFc***)

**Parameters:**

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

This parameter can be one of the following values:

- **CG\_DIVIDE\_1**:  $\Phi$ T0 = fc
- **CG\_DIVIDE\_2**:  $\Phi$ T0 = fc/2
- **CG\_DIVIDE\_4**:  $\Phi$ T0 = fc/4
- **CG\_DIVIDE\_8**:  $\Phi$ T0 = fc/8
- **CG\_DIVIDE\_16**:  $\Phi$ T0 = fc/16
- **CG\_DIVIDE\_32**:  $\Phi$ T0 = fc/32
- **CG\_DIVIDE\_64**:  $\Phi$ T0 = fc/64
- **CG\_DIVIDE\_128**:  $\Phi$ T0 = fc/128
- **CG\_DIVIDE\_256**:  $\Phi$ T0 = fc/256
- **CG\_DIVIDE\_512**:  $\Phi$ T0 = fc/512

**Description:**

This function will set the dividing level of prescaler clock.

**Return:**

**SUCCESS** means the setting has been written to registers successfully.

**ERROR** means the setting has not been written to registers.

#### 4.2.3.6 CG\_GetPhiT0Level

Get the dividing level between clock  $\Phi$ T0 and fc.

**Prototype:**

CG\_DivideLevel

CG\_GetPhiT0Level(void)

**Parameters:**

None

**Description:**

This function will get the dividing level of prescaler clock.

If the value “Reserved” is read from the register, the API will return **CG\_DIVIDE\_UNKNOWN**.

**Return:**

Dividing level between clock  $\Phi T0$  and  $f_c$ , the value will be one of the following:

**CG\_DIVIDE\_1:**  $\Phi T0 = f_c$

**CG\_DIVIDE\_2:**  $\Phi T0 = f_c/2$

**CG\_DIVIDE\_4:**  $\Phi T0 = f_c/4$

**CG\_DIVIDE\_8:**  $\Phi T0 = f_c/8$

**CG\_DIVIDE\_16:**  $\Phi T0 = f_c/16$

**CG\_DIVIDE\_32:**  $\Phi T0 = f_c/32$

**CG\_DIVIDE\_64:**  $\Phi T0 = f_c/64$

**CG\_DIVIDE\_128:**  $\Phi T0 = f_c/128$

**CG\_DIVIDE\_256:**  $\Phi T0 = f_c/256$

**CG\_DIVIDE\_512:**  $\Phi T0 = f_c/512$

**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:

$$\text{Setting\_value} = ((\text{warm-up time}) / (\text{input cycle time by frequency}))/16$$

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\_RISING**: 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. FC

### 5.1 Overview

TMPM37AFSQG device contains flash memory.  
For TMPM37AFSQG, 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\tmpm37a\_fc.c with \Libraries\TX03\_Periph\_Driver\inc\tmpm37a\_fc.h containing the API definitions for use by applications.

### 5.2 API Functions

#### 5.2.1 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)

#### 5.2.2 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()

## 5.2.3 Function Documentation

### 5.2.3.1 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.

### 5.2.3.2 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

### 5.2.3.3 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.

### 5.2.3.4 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.

### 5.2.3.5 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

## 5.2.3.6 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.

## 5.2.3.7 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.

## 5.2.3.8 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 TPM37AFSG 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.

## 5.2.3.9 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.

## 5.2.3.10 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.

## 5.2.4 Data Structure Description

None.

## 6. GPIO

### 6.1 Overview

For TOSHIBA TMPM37AFSQG 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/ tmpm37a \_gpio.c, with /Libraries/TX03\_Periph\_Driver/inc/tmpm37a \_gpio.h containing the macros, data types, structures and API definitions for use by applications.

### 6.2 API Functions

#### 6.2.1 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**)

## 6.2.2 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.

## 6.2.3 Function Documentation

### 6.2.3.1 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.

### 6.2.3.2 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.

### 6.2.3.3 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

## 6.2.3.4 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

## 6.2.3.5 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

## 6.2.3.6 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

### 6.2.3.7 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

### 6.2.3.8 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

## 6.2.3.9 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

## 6.2.3.10 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

## 6.2.3.11 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

## 6.2.3.12 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



## 6.2.3.13 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

## 6.2.3.14 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

## 6.2.4 Data Structure Description

### 6.2.4.1 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.

## 7. SBI

### 7.1 Overview

The TMPM37AFSQQ contains Serial Bus Interface (SBI). The TMPM37AFSQQ 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/tmpm37a\_sbi.c, with /Libraries/TX03\_Periph\_Driver/inc/tmpm37a\_sbi.h containing the macros, data types, structures and API definitions for use by applications.

### 7.2 API Functions

#### 7.2.1 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**);

#### 7.2.2 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.

## 7.2.3 Function Documentation

### 7.2.3.1 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

### 7.2.3.2 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

### 7.2.3.3 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

## 7.2.3.4 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

## 7.2.3.5 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

### 7.2.3.6 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

### 7.2.3.7 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

## 7.2.3.8 SBI\_Generatel2CStart

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

**Prototype:**

void  
SBI\_Generatel2CStart(TSB\_SBI\_TypeDef\* **SBIx**)

**Parameters:**

**SBIx** is the specified SBI channel.

**Description:**

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

**Return:**

None

## 7.2.3.9 SBI\_Generatel2CStop

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

**Prototype:**

void  
SBI\_Generatel2CStop(TSB\_SBI\_TypeDef\* **SBIx**)

**Parameters:**

**SBIx** is the specified SBI channel.

**Description:**

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

**Return:**

None

## 7.2.3.10 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.



## 7.2.3.11 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

## 7.2.3.12 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

## 7.2.3.13 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

## 7.2.3.14 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

## 7.2.4 Data Structure Description

### 7.2.4.1 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.

## 7.2.4.2 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.

## 8. OFD

### 8.1 Overview

TMPM37AFSQG 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/tmpm37a\_ofd.c, with /Libraries/TX03\_Periph\_Driver/inc/tmpm37a\_ofd.h containing the macros, data types, structures and API definitions for use by applications.

### 8.2 API Functions

#### 8.2.1 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);

#### 8.2.2 Detailed Description

Initialize and configure OFD function by OFD\_SetRegWriteMode(), OFD\_SetDetectionFrequency (),OFD\_Enable () and OFD\_Disable ().

#### 8.2.3 Function Documentation

##### 8.2.3.1 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/O  
FDMXPLLOFF 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

### 8.2.3.2 OFD\_Enable

Enable the OFD function.

**Prototype:**

void  
OFD\_Enable(void)

**Parameters:**

None.

**Description:**

This function will enable the OFD function.

**Return:**

None

### 8.2.3.3 OFD\_Disable

Disable the OFD function.

**Prototype:**

void  
OFD\_Disable(void)

**Parameters:**

**None.**

**Description:**

This function will disable the OFD function.

**Return:**

None

### 8.2.3.4 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

## 8.2.4 Data Structure Description

None.

## 9. TMRB

### 9.1 Overview

TOSHIBA TMPM37AFSQG 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/tmpm37a\_tmr.c, with /Libraries/TX03\_Periph\_Driver/inc/tmpm37a\_tmr.h containing the macros, data types, structures and API definitions for use by applications.

### 9.2 API Functions

#### 9.2.1 Function List

- ◆ void TMRB\_Enable(TSB\_TB\_TypeDef \* **TBx**);
- ◆ void TMRB\_Disable(TSB\_TB\_TypeDef \* **TBx**);
- ◆ void TMRB\_SetRunState(TSB\_TB\_TypeDef \* **TBx**, uint32\_t **Cmd**);
- ◆ void TMRB\_Init(TSB\_TB\_TypeDef \* **TBx**, TMRB\_InitTypeDef \* **InitStruct**);
- ◆ void TMRB\_SetCaptureTiming(TSB\_TB\_TypeDef \* **TBx**, uint32\_t **CaptureTiming**);
- ◆ void TMRB\_SetFlipFlop(TSB\_TB\_TypeDef \* **TBx**, TMRB\_FFOutputTypeDef \* **FFStruct**);
- ◆ TMRB\_INTFactor TMRB\_GetINTFactor(TSB\_TB\_TypeDef \* **TBx**);
- ◆ void TMRB\_SetINTMask(TSB\_TB\_TypeDef \* **TBx**, uint32\_t **INTMask**);
- ◆ void TMRB\_ChangeLeadingTiming(TSB\_TB\_TypeDef \* **TBx**, uint32\_t **LeadingTiming**);
- ◆ void TMRB\_ChangeTrailingTiming(TSB\_TB\_TypeDef \* **TBx**, uint32\_t **TrailingTiming**);
- ◆ uint16\_t TMRB\_GetUpCntValue(TSB\_TB\_TypeDef \* **TBx**);
- ◆ uint16\_t TMRB\_GetCaptureValue(TSB\_TB\_TypeDef \* **TBx**, uint8\_t **CapReg**);
- ◆ void TMRB\_ExecuteSWCapture(TSB\_TB\_TypeDef \* **TBx**);
- ◆ void TMRB\_SetIdleMode(TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**);



- ◆ void TMRB\_SetDoubleBuf(TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**,  
uint8\_t **WriteRegMode**);
- ◆ void TMRB\_SetExtStartTrg(TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**,  
uint8\_t **TrgMode**);
- ◆ void TMRB\_SetClkInCoreHalt(TSB\_TB\_TypeDef \* **TBx**, uint8\_t **ClkState**);

## 9.2.2 Detailed Description

Functions listed above can be divided into four parts:

- 1) Configure and control the common functions of each TMRB channel are handled by TMRB\_Enable(), TMRB\_Disable(), TMRB\_Init(), TMRB\_SetRunState(), TMRB\_ChangeLeadingTiming() and TMRB\_ChangeTrailingTiming().
- 2) Capture function of each TMRB channel is handled by TMRB\_SetCaptureTiming(), and TMRB\_ExecuteSWCapture().
- 3) The status indication of each TMRB channel is handled by TMRB\_GetINTFactor(), TMRB\_GetUpCntValue() and TMRB\_GetCaptureValue().
- 4) TMRB\_SetFlipFlop(), TMRB\_SetINTMask(), TMRB\_SetIdleMode(), TMRB\_SetDoubleBuf(), TMRB\_SetExtStartTrg() and TMRB\_SetClkInCoreHalt () handle other specified functions.

## 9.2.3 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.**

### 9.2.3.1 TMRB\_Enable

Enable the specified TMRB channel.

**Prototype:**

void  
TMRB\_Enable(TSB\_TB\_TypeDef\* **TBx**)

**Parameters:**

**TBx** is the specified TMRB channel.

**Description:**

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

**Return:**

None

### 9.2.3.2 TMRB\_Disable

Disable the specified TMRB channel.

**Prototype:**

void  
TMRB\_Disable(TSB\_TB\_TypeDef\* **TBx**)

**Parameters:**

**TBx** is the specified TMRB channel.

**Description:**

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

**Return:**  
None

### 9.2.3.3 TMRB\_SetRunState

Start or stop counter of the specified TB channel.

**Prototype:**  
void  
TMRB\_SetRunState(TSB\_TB\_TypeDef\* **TBx**,  
uint32\_t **Cmd**)

**Parameters:**  
**TBx** is the specified TMRB channel.  
**Cmd** sets the state of up-counter, which can be:  
➤ **TMRB\_RUN**: starting counting  
➤ **TMRB\_STOP**: stopping counting

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

**Return:**  
None

### 9.2.3.4 TMRB\_Init

Initialize the specified TMRB channel.

**Prototype:**  
void  
TMRB\_Init(TSB\_TB\_TypeDef\* **TBx**,  
TMRB\_InitTypeDef\* **InitStruct**)

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

## 9.2.3.5 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

## 9.2.3.6 TMRB\_SetFlipFlop

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

### Prototype:

```
void  
TMRB_SetFlipFlop(TSB_TB_TypeDef* TBx,  
                 TMRB_FFOutputTypeDef* FFStruct)
```

**Parameters:**

**TBx** is the specified TMRB channel, 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

## 9.2.3.7 TMRB\_GetINTFactor

Indicate what causes the interrupt.

**Prototype:**

```
TMRB_INTFactor  
TMRB_GetINTFactor(TSB_TB_TypeDef* TBx)
```

**Parameters:**

**TBx** is the specified TMRB channel.

**Description:**

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

```
}
```

## 9.2.3.8 TMRB\_SetINTMask

Mask the specified TMRB interrupt.

### Prototype:

```
void  
TMRB_SetINTMask(TSB_TB_TypeDef* TBx,  
                uint32_t INTMask)
```

### Parameters:

**TBx** is the specified TMRB channel.

**INTMask** specifies the interrupt to be masked, which can be

- **TMRB\_MASK\_MATCH\_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

## 9.2.3.9 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**.

## 9.2.3.10 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.

## 9.2.3.11 TMRB\_GetUpCntValue

Get up-counter value of the specified TMRB channel.

**Prototype:**

```
uint16_t  
TMRB_GetUpCntValue(TSB_TB_TypeDef* TBx)
```

**Parameters:**

**TBx** is the specified TMRB channel.

**Description:**

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

**Return:**

The value of up-counter.

## 9.2.3.12 TMRB\_GetCaptureValue

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

**Prototype:**

```
uint16_t  
TMRB_GetCaptureValue(TSB_TB_TypeDef* TBx,  
                     uint8_t CapReg)
```

**Parameters:**

**TBx** is 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

### 9.2.3.13 TMRB\_ExecuteSWCapture

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

**Prototype:**

```
void  
TMRB_ExecuteSWCapture(TSB_TB_TypeDef* TBx)
```

**Parameters:**

**TBx** is the specified TMRB channel.

**Description:**

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

**Return:**

None

### 9.2.3.14 TMRB\_SetIdleMode

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

### 9.2.3.15 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

### 9.2.3.16 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

## 9.2.3.17 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

## 9.2.4 Data Structure Description

### 9.2.4.1 TMRB\_InitTypeDef

**Data Fields:**

uint32\_t

**Mode** selects TMRB working mode between **TMRB\_INTERVAL\_TIMER** (internal interval timer mode) and **TMRB\_EVENT\_CNT** (external event counter).

**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**.

## 9.2.4.2 TMRB\_FFOutputTypeDef

**Data Fields:**

uint32\_t

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

- **TMRB\_FLIPFLOP\_INVERT**: setting output reversed by using software.
- **TMRB\_FLIPFLOP\_SET**: setting output to be high level.
- **TMRB\_FLIPFLOP\_CLEAR**: setting output to be low level.

uint32\_t

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

- **TMRB\_DISALBE\_FLIPFLOP**, which disables the flip-flop output reverse trigger,
- **TMRB\_FLIPFLOP\_TAKE\_CAPTURE\_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\_CAPTURE\_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.

### 9.2.4.3 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

**Reserved**: 29 -

## 10. SIO/UART

### 10.1 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/tmpm37a\_uart.c, with /Libraries/TX03\_Periph\_Driver/inc/tmpm37a\_uart.h containing the macros, data types, structures and API definitions for use by applications.

### 10.2 API Functions

#### 10.2.1 Function List

- ◆ void UART\_Enable(TSB\_SC\_TypeDef\* **UARTx**)
- ◆ void UART\_Disable(TSB\_SC\_TypeDef\* **UARTx**)
- ◆ WorkState UART\_GetBufState(TSB\_SC\_TypeDef\* **UARTx**, uint8\_t **Direction**)
- ◆ void UART\_SWReset(TSB\_SC\_TypeDef\* **UARTx**)
- ◆ void UART\_Init(TSB\_SC\_TypeDef\* **UARTx**, UART\_InitTypeDef\* **InitStruct**)
- ◆ uint32\_t UART\_GetRxData(TSB\_SC\_TypeDef\* **UARTx**)
- ◆ void UART\_SetTxData(TSB\_SC\_TypeDef\* **UARTx**, uint32\_t **Data**)
- ◆ void UART\_DefaultConfig(TSB\_SC\_TypeDef\* **UARTx**)
- ◆ UART\_Err UART\_GetErrState(TSB\_SC\_TypeDef\* **UARTx**)
- ◆ void UART\_SetWakeUpFunc(TSB\_SC\_TypeDef\* **UARTx**,  
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)

## 10.2.2 Detailed Description

Functions listed above can be divided into four parts:

- 1) Initialize and configure the common functions of each UART channel are handled by UART\_Enable(), UART\_Disable(), UART\_SetInputClock(), UART\_Init(), UART\_DefaultConfig(), UART\_SetInputClock(), SIO\_Enable(), SIO\_Disable(), SIO\_SetInputClock() and SIO\_Init().
- 2) Transfer control and error check of each UART channel are handled by UART\_GetBufState(), UART\_GetRxData(), UART\_SetTxData(), UART\_GetErrState(), SIO\_GetRxData() and SIO\_SetTxData.
- 3) UART\_SWReset(), UART\_SetWakeUpFunc() and UART\_SetIdleMode() handle other specified functions.
- 4) FIFO operation functions are UART\_FIFOConfig(), UART\_SetFIFOTransferMode(), UART\_TRxAutoDisable(), UART\_RxFIFOINTCtrl(), UART\_TxFIFOINTCtrl(), UART\_RxFIFOByteSel(), UART\_RxFIFOFillLevel(), UART\_RxFIFOINTSel(), UART\_RxFIFOClear(), UART\_TxFIFOFillLevel(), UART\_TxFIFOINTSel(), UART\_TxFIFOClear(), UART\_TxBufferClear(), UART\_GetRxFIFOFillLevelStatus(), UART\_GetRxFIFOOverRunStatus(), UART\_GetTxFIFOFillLevelStatus() and UART\_GetTxFIFOUnderRunStatus().

## 10.2.3 Function Documentation

**Note:** in all of the following APIs, parameter “TSB\_SC\_TypeDef\* **UARTx**” can be one of the following values:

**UART0, UART1.**

parameter “TSB\_SC\_TypeDef\* **SIOx**” can be one of the following values:  
**SIO0.**

### 10.2.3.1 UART\_Enable

Enable the specified UART channel.

**Prototype:**

void  
UART\_Enable(TSB\_SC\_TypeDef\* **UARTx**)

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

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

**Return:**

None

## 10.2.3.2 UART\_Disable

Disable the specified UART channel.

**Prototype:**

void  
UART\_Disable(TSB\_SC\_TypeDef\* **UARTx**)

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

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

**Return:**

None

## 10.2.3.3 UART\_GetBufState

Indicate the state of transmission or reception buffer.

**Prototype:**

WorkState  
UART\_GetBufState(TSB\_SC\_TypeDef\* **UARTx**,  
uint8\_t **Direction**)

**Parameters:**

**UARTx** is the specified UART channel.

**Direction** select the direction of transfer, which can be one of:

- **UART\_RX** for reception
- **UART\_TX** for transmission

**Description:**

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

**Return:**

**DONE** means that the buffer can be read or written.

**BUSY** means that the transfer is ongoing.

## 10.2.3.4 UART\_SWReset

Reset the specified UART channel.

**Prototype:**

void  
UART\_SWReset(TSB\_SC\_TypeDef\* **UARTx**)

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

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

**Return:**

None

## 10.2.3.5 UART\_Init

Initialize and configure the specified UART channel.

**Prototype:**

```
void  
UART_Init(TSB_SC_TypeDef* UARTx,  
          UART_InitTypeDef* InitStruct)
```

**Parameters:**

**UARTx** is the specified UART channel.

**InitStruct** is the structure containing basic UART configuration including baud rate, data bits per transfer, stop bits, parity, transfer mode and flow control (refer to "Data Structure Description" for details).

**Description:**

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

**Return:**

None

## 10.2.3.6 UART\_GetRxData

Get data received from the specified UART channel.

**Prototype:**

```
uint32_t  
UART_GetRxData(TSB_SC_TypeDef* UARTx)
```

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

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

**Return:**

Data which has been received, the data value range is 0x00 to 0x1FF.

## 10.2.3.7 UART\_SetTxData

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

**Prototype:**

```
void  
UART_SetTxData(TSB_SC_TypeDef* UARTx,  
               uint32_t Data)
```

**Parameters:**

**UARTx** is the specified UART channel.

**Data** is a frame to be sent, which can be 7-bit, 8-bit or 9-bit, depending on the 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

### 10.2.3.8 UART\_DefaultConfig

Initialize the specified UART channel in the default configuration.

**Prototype:**

```
void  
UART_DefaultConfig(TSB_SC_TypeDef* UARTx)
```

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

This function will initialize the selected UART channel in the following configuration:

Baud rate: 115200 bps

Data bits: 8 bits

Stop bits: 1 bit

Parity: None

Flow Control: None

Both transmission and reception are enabled. And baud rate generator is used as source clock.

**Return:**

None

### 10.2.3.9 UART\_GetErrState

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

**Prototype:**

```
UART_Err  
UART_GetErrState(TSB_SC_TypeDef* UARTx)
```

**Parameters:**



**UARTx** is the specified UART channel.

**Description:**

This function will check whether an error occurs at the last transfer and return the result, which can be **UART\_NO\_ERR**, meaning no error, **UART\_OVERRUN**, meaning overrun, **UART\_PARITY\_ERR**, meaning even or odd parity error, **UART\_FRAMING\_ERR**, meaning framing error, and **UART\_ERRS**, meaning more than one error above.

**Return:**

**UART\_NO\_ERR** means there is no error in the last transfer.

**UART\_OVERRUN** means that overrun occurs in the last transfer.

**UART\_PARITY\_ERR** means either even parity or odd parity fails.

**UART\_FRAMING\_ERR** means there is framing error in the last transfer.

**UART\_ERRS** means that 2 or more errors occurred in the last transfer.

## 10.2.3.10 UART\_SetWakeUpFunc

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

## 10.2.3.11 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

## 10.2.3.12 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\_1** or **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

## 10.2.3.13 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

## 10.2.3.14 UART\_SetFIFOTransferMode

Transfer mode setting.

**Prototype:**

```
void  
UART_SetFIFOTransferMode (TSB_SC_TypeDef* UARTx,  
                          uint32_t TransferMode);
```

**Parameters:**

**UARTx** is the specified UART channel.

**TransferMode** Transfer mode.

This parameter can be one of the following values:

**UART\_TRANSFER\_PROHIBIT**, **UART\_TRANSFER\_HALFDPX\_RX**, **UART\_TRANSFER\_HALFDPX\_TX** or **UART\_TRANSFER\_FULLDPX**.

**Description:**

Transfer mode setting.

**Return:**

None

## 10.2.3.15 UART\_TRxAutoDisable

Controls automatic disabling of transmission and reception.

**Prototype:**

```
void  
UART_TRxAutoDisable (TSB_SC_TypeDef* UARTx,  
                     UART_TRxAutoDisable TRxAutoDisable);
```

**Parameters:**

**UARTx** is the specified UART channel.

**TRxAutoDisable** Disabling transmission and reception or not

This parameter can be one of the following values:

**UART\_RTXCNT\_NONE** or **UART\_RTXCNT\_AUTODISABLE** .

**Description:**

Controls automatic disabling of transmission and reception.

**Return:**

None

## 10.2.3.16 UART\_RxFIFOINTCtrl

Enable or disable receive interrupt for receive FIFO.

**Prototype:**

```
void  
UART_RxFIFOINTCtrl (TSB_SC_TypeDef* UARTx,  
                    FunctionalState NewState);
```

**Parameters:**

**UARTx** is the specified UART channel.

**NewState** is new state of receive interrupt for receive FIFO.

This parameter can be one of the following values:

**ENABLE or DISABLE**

**Description:**

Enable or disable receive interrupt for receive FIFO.

**Return:**

None

## 10.2.3.17 UART\_TxFIFOINTCtrl

Enable or disable transmit interrupt for transmit FIFO.

**Prototype:**

void

UART\_TxFIFOINTCtrl (TSB\_SC\_TypeDef\* **UARTx**,  
FunctionalState **NewState**);

**Parameters:**

**UARTx** is the specified UART channel.

**NewState** is new state of transmit interrupt for transmit FIFO.

This parameter can be one of the following values:

**ENABLE or DISABLE**

**Description:**

Enable or disable transmit interrupt for transmit FIFO.

**Return:**

None

## 10.2.3.18 UART\_RxFIFOByteSel

Bytes used in receive FIFO.

**Prototype:**

void

UART\_RxFIFOByteSel (TSB\_SC\_TypeDef\* **UARTx**,  
uint32\_t **BytesUsed**);

**Parameters:**

**UARTx** is the specified UART channel.

**BytesUsed** is bytes used in receive FIFO.

This parameter can be one of the following values:

**UART\_RXFIFO\_MAX or UART\_RXFIFO\_RXFLEVEL**

**Description:**

Bytes used in receive FIFO.

**Return:**

None

## 10.2.3.19 UART\_RxFIFOFillLevel

Receive FIFO fill level to generate receive interrupts.

**Prototype:**

```
void  
UART_RxFIFOFillLevel (TSB_SC_TypeDef* UARTx,  
                      uint32_t RxFIFOLevel);
```

**Parameters:**

**UARTx** is the specified UART channel.

**RxFIFOLevel** is receive FIFO fill level.

This parameter can be one of the following values:

**UART\_RXFIFO4B\_FLEVLE\_4\_2B**, **UART\_RXFIFO4B\_FLEVLE\_1\_1B**,  
**UART\_RXFIFO4B\_FLEVLE\_2\_2B** or **UART\_RXFIFO4B\_FLEVLE\_3\_1B**.

**Description:**

Receive FIFO fill level to generate receive interrupts.

**Return:**

None

## 10.2.3.20 UART\_RxFIFOINTSel

Select RX interrupt generation condition.

**Prototype:**

```
void  
UART_RxFIFOINTSel (TSB_SC_TypeDef* UARTx,  
                   uint32_t RxINTCondition);
```

**Parameters:**

**UARTx** is the specified UART channel.

**RxINTCondition** is RX interrupt generation condition.

This parameter can be one of the following values:

**UART\_RFIS\_REACH\_FLEVEL** or **UART\_RFIS\_REACH\_EXCEED\_FLEVEL**

**Description:**

Select RX interrupt generation condition.

**Return:**

None

## 10.2.3.21 UART\_RxFIFOClear

Receive FIFO clear.

**Prototype:**

```
void  
UART_RxFIFOClear (TSB_SC_TypeDef* UARTx);
```

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Receive FIFO clear.

**Return:**

None

## 10.2.3.22 UART\_TxFIFOFillLevel

Transmit FIFO fill level to generate transmit interrupts.

**Prototype:**

void

UART\_TxFIFOFillLevel (TSB\_SC\_TypeDef\* **UARTx**,  
uint32\_t **TxFIFOLevel**);

**Parameters:**

**UARTx** is the specified UART channel.

**TxFIFOLevel** is transmit FIFO fill level.

This parameter can be one of the following values:

**UART\_TXFIFO4B\_FLEVLE\_0\_0B**, **UART\_TXFIFO4B\_FLEVLE\_1\_1B**,  
**UART\_TXFIFO4B\_FLEVLE\_2\_0B** or **UART\_TXFIFO4B\_FLEVLE\_3\_1B**.

**Description:**

Transmit FIFO fill level to generate transmit interrupts.

**Return:**

None

## 10.2.3.23 UART\_TxFIFOINTSel

Select TX interrupt generation condition.

**Prototype:**

void

UART\_TxFIFOINTSel (TSB\_SC\_TypeDef\* **UARTx**,  
uint32\_t **TxINTCondition**);

**Parameters:**

**UARTx** is the specified UART channel.

**TxINTCondition** is TX interrupt generation condition.

This parameter can be one of the following values:

**UART\_TFIS\_REACH\_FLEVEL** or **UART\_TFIS\_REACH\_NOREACH\_FLEVEL**.

**Description:**

Select TX interrupt generation condition.

**Return:**

None

## 10.2.3.24 UART\_TxFIFOClear

TransmitFIFO clear.

**Prototype:**

void

UART\_TxFIFOClear (TSB\_SC\_TypeDef\* **UARTx**);

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Transmit FIFO clear.

**Return:**

None

## 10.2.3.25 UART\_TxBufferClear

Transmit buffer clear.

**Prototype:**

void

UART\_TxBufferClear (TSB\_SC\_TypeDef\* **UARTx**);

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Transmit buffer clear.

**Return:**

None

## 10.2.3.26 UART\_GetRxFIFOFillLevelStatus

Status of receive FIFO fill level.

**Prototype:**

uint32\_t

UART\_GetRxFIFOFillLevelStatus (TSB\_SC\_TypeDef\* **UARTx**);

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Status of receive FIFO fill level.

**Return:**

**UART\_TRXFIFO\_EMPTY:** TX FIFO fill level is empty.

**UART\_TRXFIFO\_1B:** TX FIFO fill level is 1 byte.

**UART\_TRXFIFO\_2B:** TX FIFO fill level is 2 bytes.

**UART\_TRXFIFO\_3B:** TX FIFO fill level is 3 bytes.

**UART\_TRXFIFO\_4B:** TX FIFO fill level is 4 bytes.

## 10.2.3.27 UART\_GetRxFIFOOverRunStatus

Receive FIFO overrun.

**Prototype:**

uint32\_t  
UART\_GetRxFIFOOverRunStatus (TSB\_SC\_TypeDef\* **UARTx**);

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Receive FIFO overrun.

**Return:**

**UART\_RXFIFO\_OVERRUN:** Flags for RX FIFO overrun.

## 10.2.3.28 UART\_GetTxFIFOFillLevelStatus

Status of transmit FIFO fill level.

**Prototype:**

uint32\_t  
UART\_GetTxFIFOFillLevelStatus (TSB\_SC\_TypeDef\* **UARTx**);

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Status of transmit FIFO fill level.

**Return:**

**UART\_TRXFIFO\_EMPTY:** TX FIFO fill level is empty.

**UART\_TRXFIFO\_1B:** TX FIFO fill level is 1 byte.

**UART\_TRXFIFO\_2B:** TX FIFO fill level is 2 bytes.

**UART\_TRXFIFO\_3B:** TX FIFO fill level is 3 bytes.

**UART\_TRXFIFO\_4B:** TX FIFO fill level is 4 bytes.

## 10.2.3.29 UART\_GetTxFIFOUnderRunStatus

Transmit FIFO under run

**Prototype:**

uint32\_t  
UART\_GetTxFIFOUnderRunStatus (TSB\_SC\_TypeDef\* **UARTx**);

**Parameters:**

**UARTx** is the specified UART channel.

**Description:**

Transmit FIFO under run

**Return:**

**UART\_TXFIFO\_UNDERRUN:** Flags for TX FIFO under-run.



## 10.2.3.30 UART\_SetInputClock

Selects input clock for prescaler.

**Prototype:**

void

UART\_SetInputClock (TSB\_SC\_TypeDef \* UARTx,  
uint32\_t clock)

**Parameters:**

**UARTx** is the specified UART channel.

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

This parameter can be one of the following values:

**0** : PhiT0/2

**1** : PhiT0

**Description:**

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

**Return:**

None

## 10.2.3.31 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

## 10.2.3.32 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

## 10.2.3.33 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

## 10.2.3.34 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

## 10.2.3.35 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

## 10.2.3.36 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

## 10.2.4 Data Structure Description

### 10.2.4.1 UART\_InitTypeDef

**Data Fields:**

uint32\_t

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

uint32\_t

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

- **UART\_DATA\_BITS\_7** for 7-bit mode
- **UART\_DATA\_BITS\_8** for 8-bit mode
- **UART\_DATA\_BITS\_9** for 9-bit mode

uint32\_t

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

- **UART\_STOP\_BITS\_1** for 1 stop bit
- **UART\_STOP\_BITS\_2** for 2 stop bits

uint32\_t

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

- **UART\_NO\_PARITY** for no parity

- **UART\_EVEN\_PARITY** for even parity
- **UART\_ODD\_PARITY** for odd parity

uint32\_t

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

- **UART\_ENABLE\_TX** for enabling transmission
- **UART\_ENABLE\_RX** for enabling reception

uint32\_t

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

- **UART\_NONE\_FLOW\_CTRL** for no flow control

\*: 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.

## 10.2.4.2 SIO\_InitTypeDef

**Data Fields:**

uint32\_t

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

- **SIO\_SCLKS\_TXDF\_RXDR** Data in the transfer buffer is sent to TXDx pin one bit at a time on the falling edge of SCLKx, data from RXDx pin is received in the receive buffer one bit at a time on the rising edge of SCLKx.
- **SIO\_SCLKS\_TXDR\_RXDF** Data in the transfer buffer is sent to TXDx pin one bit at a time on the rising edge of SCLKx, data from RXDx pin is received in the receive buffer one bit at a time on the falling edge of SCLKx.

uint32\_t

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

- **SIO\_TIDLE\_LOW** Set the status of TXDx pin keep a low level output.
- **SIO\_TIDLE\_HIGH** Set the status of TXDx pin keep a high level output.
- **SIO\_TIDLE\_LAST** Set the status of TXDx pin keep a last bit.

uint32\_t

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

- **SIO\_TXDEMP\_LOW** Set the status of TXDx pin is low level output.
- **SIO\_TXDEMP\_HIGH** Set the status of TXDx pin is high level output.

uint32\_t

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

- **SIO\_EHOLD\_FC\_2** Set a last bit hold time is 2/fc.
- **SIO\_EHOLD\_FC\_4** Set a last bit hold time is 4/fc.
- **SIO\_EHOLD\_FC\_8** Set a last bit hold time is 8/fc.
- **SIO\_EHOLD\_FC\_16** Set a last bit hold time is 16/fc.
- **SIO\_EHOLD\_FC\_32** Set a last bit hold time is 32/fc.
- **SIO\_EHOLD\_FC\_64** Set a last bit hold time is 64/fc.
- **SIO\_EHOLD\_FC\_128** Set a last bit hold time is 128/fc.

uint32\_t

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

- **SIO\_SINT\_TIME\_NONE** Interval time is None.
- **SIO\_SINT\_TIME\_SCLK\_1** Interval time is 1xSCLK.
- **SIO\_SINT\_TIME\_SCLK\_2** Interval time is 2xSCLK.
- **SIO\_SINT\_TIME\_SCLK\_4** Interval time is 4xSCLK.

- **SIO\_SINT\_TIME\_SCLK\_8** Interval time is 8xSCLK.
- **SIO\_SINT\_TIME\_SCLK\_16** Interval time is 16xSCLK.
- **SIO\_SINT\_TIME\_SCLK\_32** Interval time is 32xSCLK.
- **SIO\_SINT\_TIME\_SCLK\_64** Interval time is 64xSCLK.

uint32\_t

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

- **SIO\_TRANSFER\_PROHIBIT** Transfer prohibit.
- **SIO\_TRANSFER\_HALFDPX\_RX** Half duplex(Receive).
- **SIO\_TRANSFER\_HALFDPX\_TX** Half duplex(Transmit).
- **SIO\_TRANSFER\_FULDPX** Full duplex.

uint32\_t

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

- **SIO\_LSB\_FRIST** LSB first.
- **SIO\_MSB\_FRIST** MSB first.

uint32\_t

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

- **UART\_ENABLE\_TX** for enabling transmission.
- **UART\_ENABLE\_RX** for enabling reception.

uint32\_t

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

- **SIO\_WBUF\_DISABLE** Double buffer disable.
- **SIO\_WBUF\_ENABLE** Double buffer enable.

uint32\_t

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

- **SIO\_BR\_CLOCK\_TS0** Select the input clock to baud rate generator is TS0.
- **SIO\_BR\_CLOCK\_TS2** Select the input clock to baud rate generator is TS2.
- **SIO\_BR\_CLOCK\_TS8** Select the input clock to baud rate generator is TS8.
- **SIO\_BR\_CLOCK\_TS32** Select the input clock to baud rate generator is TS32.

uint32\_t

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

- **SIO\_BR\_DIVIDER\_16** Division ratio is 16.
- **SIO\_BR\_DIVIDER\_1** Division ratio is 1.
- **SIO\_BR\_DIVIDER\_2** Division ratio is 2.
- **SIO\_BR\_DIVIDER\_3** Division ratio is 3.
- **SIO\_BR\_DIVIDER\_4** Division ratio is 4.
- **SIO\_BR\_DIVIDER\_5** Division ratio is 5.
- **SIO\_BR\_DIVIDER\_6** Division ratio is 6.
- **SIO\_BR\_DIVIDER\_7** Division ratio is 7.
- **SIO\_BR\_DIVIDER\_8** Division ratio is 8.
- **SIO\_BR\_DIVIDER\_9** Division ratio is 9.
- **SIO\_BR\_DIVIDER\_10** Division ratio is 10.
- **SIO\_BR\_DIVIDER\_11** Division ratio is 11.
- **SIO\_BR\_DIVIDER\_12** Division ratio is 12.
- **SIO\_BR\_DIVIDER\_13** Division ratio is 13.
- **SIO\_BR\_DIVIDER\_14** Division ratio is 14.
- **SIO\_BR\_DIVIDER\_15** Division ratio is 15.

## 11. VLTD

### 11.1 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/tmpm37a\_vltd.c, with /Libraries/TX03\_Periph\_Driver/inc/tmpm37a\_vltd.h containing the macros, data types, structures and API definitions for use by applications.

### 11.2 API Functions

#### 11.2.1 Function List

- ◆ void VLTD\_Enable(void);
- ◆ void VLTD\_Disable(void);
- ◆ void VLTD\_SetVoltage(uint32\_t *Voltage*);

#### 11.2.2 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().

#### 11.2.3 Function Documentation

##### 11.2.3.1 VLTD\_Enable

Enable the VLTD function.

**Prototype:**

```
void  
VLTD_Enable(void)
```

**Parameters:**

None

**Description:**

This function will enable the VLTD function.

**Return:**

None

## 11.2.3.2 VLTD\_Disable

Disable the VLTD function.

**Prototype:**

void  
VLTD\_Disable(void)

**Parameters:**

None

**Description:**

This function will disable the VLTD function.

**Return:**

None

## 11.2.3.3 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 =  $4.1V \pm 0.2V$
- **VLTD\_DETECT\_VOLTAGE\_44**: Detection voltage =  $4.4V \pm 0.2V$
- **VLTD\_DETECT\_VOLTAGE\_46**: Detection voltage =  $4.6V \pm 0.2V$

**Description:**

This function will set the value of detection voltage.

**Return:**

None

## 11.2.4 Data Structure Description

None

## 12. WDT

### 12.1 Overview

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

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

This driver is contained in \Libraries\TX03\_Periph\_Driver\src\tmpm37a\_wdt.c, with \Libraries\TX03\_Periph\_Driver\inc\tmpm37a\_wdt.h containing the API definitions for use by applications.

### 12.2 API Functions

#### 12.2.1 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)

#### 12.2.2 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().

#### 12.2.3 Function Documentation

##### 12.2.3.1 WDT\_SetDetectTime

Set detection time for WDT.

**Prototype:**

void  
WDT\_SetDetectTime(uint32\_t **DetectTime**)

**Parameters:**

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



- WDT\_DETECT\_TIME\_EXP\_15: *DetectTime* is  $2^{15}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_17: *DetectTime* is  $2^{17}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_19: *DetectTime* is  $2^{19}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_21: *DetectTime* is  $2^{21}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_23: *DetectTime* is  $2^{23}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_25: *DetectTime* is  $2^{25}/\text{fsys}$

**Description:**

This function will set detection time for WDT.

**Return:**

None

### 12.2.3.2 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

### 12.2.3.3 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

## 12.2.3.4 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

## 12.2.3.5 WDT\_Enable

Enable the WDT function.

**Prototype:**  
void  
WDT\_Enable(void)

**Parameters:**  
None

**Description:**  
This function will enable WDT.

**Return:**  
None

## 12.2.3.6 WDT\_Disable

Disable the WDT function.

**Prototype:**  
void  
WDT\_Disable(void)

**Parameters:**

None

**Description:**

This function will disable WDT.

**Return:**

None

## 12.2.3.7 WDT\_WriteClearCode

Write the clear code.

**Prototype:**

void  
WDT\_WriteClearCode (void)

**Parameters:**

None

**Description:**

This function will clear the WDT counter.

**Return:**

None

## 12.2.4 Data Structure Description

### 12.2.4.1 WDT\_InitTypeDef

**Data Fields:**

uint32\_t

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

- **WDT\_DETECT\_TIME\_EXP\_15:** *DetectTime* is  $2^{15}/f_{sys}$
- **WDT\_DETECT\_TIME\_EXP\_17:** *DetectTime* is  $2^{17}/f_{sys}$
- **WDT\_DETECT\_TIME\_EXP\_19:** *DetectTime* is  $2^{19}/f_{sys}$
- **WDT\_DETECT\_TIME\_EXP\_21:** *DetectTime* is  $2^{21}/f_{sys}$
- **WDT\_DETECT\_TIME\_EXP\_23:** *DetectTime* is  $2^{23}/f_{sys}$
- **WDT\_DETECT\_TIME\_EXP\_25:** *DetectTime* is  $2^{25}/f_{sys}$

uint32\_t

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

- **WDT\_WDOUT:** Set WDT to generate reset when the counter overflows.
- **WDT\_NMIINT:** Set WDT to generate NMI interrupt when the counter overflows.

## 13. PMD

### 13.1 Overview

The TMPM37AFSQG 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\tmpm37a\_pmd.c, with TX03\_Periph\_Driver\inc\tmpm37a\_pmd.h containing the API definitions for use by applications.

### 13.2 API Functions

#### 13.2.1 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**);

## 13.2.2 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().

## 13.2.3 Function Documentation

**\*Note:** In all of the following APIs, parameter “TSB\_PMD\_TypeDef \* **PMDx**” can be **PMD1**.

### 13.2.3.1 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

### 13.2.3.2 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

### 13.2.3.3 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

## 13.2.3.4 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

## 13.2.3.5 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

## 13.2.3.6 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

### 13.2.3.7 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.

### 13.2.3.8 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

## 13.2.3.9 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			
<WOC[1]> <VOC[1]> <UOC[1]> (Upper)	<WOC[0]> <VOC[0]> ><UOC[0]> (Lower)	0 : H/L output		1 : PWM output	
		Upper output	Lower output	Upper output	Lower output
0	0	L	L	PWM	PWM
0	1	L	H	L	PWM
1	0	H	L	PWM	L
1	1	H	H	PWM	PWM

MTPDxMDCR<SYNTMD>=0

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

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

MTPDxMDCR<SYNTMD>=1

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

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

MTPDxMDCR<SYNTMD>=1

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

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

**Return:**  
None

### 13.2.3.10PMD\_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

### 13.2.3.11 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

### 13.2.3.12 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

## 13.2.3.13PMD\_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

## 13.2.3.14PMD\_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

## 13.2.3.15PMD\_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

## 13.2.3.16PMD\_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

## 13.2.3.17PMD\_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

### 13.2.3.18PMD\_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"

### 13.2.3.19PMD\_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.

## 13.2.3.20PMD\_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

## 13.2.3.21 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



## 13.2.3.22PMD\_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

## 13.2.3.23PMD\_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.

### 13.2.3.24PMD\_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].

### 13.2.3.25PMD\_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

### 13.2.3.26PMD\_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

### 13.2.3.27PMD\_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
- **DISABLE:** 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

### 13.2.3.28PMD\_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

### 13.2.3.29PMD\_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

### 13.2.3.30PMD\_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

## 13.2.3.31 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

## 13.2.3.32 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

## 13.2.3.33PMD\_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
- **DISABLE**: 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

## 13.2.3.34PMD\_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

## 13.2.3.35PMD\_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

## 13.2.3.36PMD\_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 release in the protection state
- **DIABLE**: Disable trigger automatic release in the protection state

**Description:**

This function will enable or disable automatic release of OVV protection of the specified PMD channel.

**Return:**

None

## 13.2.3.37PMD\_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"

### 13.2.3.38PMD\_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

### 13.2.3.39PMD\_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

## 13.2.3.40 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

## 13.2.3.41 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

## 13.2.3.42 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

## 13.2.3.43 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

## 13.2.4 Data Structure Description

### 13.2.4.1 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