TOSHIBA

TX00 Peripheral Driver User Guide (TMPM037)

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M037UG-01E

RESTRICTIONS ON PRODUCT USE

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

Index

1. Introduction 1
2. Organization of TOSHIBA TX00 Peripheral Driver 1
3. ADC 1
3.1 Overview 1
3.2 API Functions 2
3.2.1 Function List 2
3.2.2 Detailed Description 2
3.2.3 Function Documentation 3
3.2.4 Data Structure Description 18
4. CG 20
4.1 Overview 20
4.2 API Functions 20
4.2.1 Function List 20
4.2.2 Detailed Description 21
4.2.3 Function Documentation 21
4.2.4 Data Structure Description 37
5. DMAC 39
5.1 Overview 39
5.2 API Functions 39
5.2.1 Function List 39
5.2.2 Detailed Description 39
5.2.3 Function Documentation 40
5.2.4 Data Structure Description 50
6. FC 54
6.1 Overview 54
6.2 API Functions 55
6.2.1 Function List 55
6.2.2 Detailed Description 55
6.2.3 Function Documentation 55
6.2.4 Data Structure Description 62
7. GPIO 63
7.1 Overview 63
7.2 API Functions 63
7.2.1 Function List 63
7.2.2 Detailed Description 63

7.2.3 Function Documentation 64
7.2.4 Data Structure Description 76
8. I2C.. 79
8.1 Overview 79
8.2 API Functions 79
8.2.1 Function List 79
8.2.2 Detailed Description 80
8.2.3 Function Documentation 80
8.2.4 Data Structure Description 88
9. LVD 92
9.1 Overview 92
9.2 API Functions 92
9.2.1 Function List 92
9.2.2 Detailed Description 92
9.2.3 Function Documentation 92
9.2.4 Data Structure Description 96
10. TMR16A 97
10.1 Overview 97
10.2 API Functions 97
10.2.1 Function List 97
10.2.2 Detailed Description 97
10.2.3 Function Documentation 97
10.2.1 Data Structure Description 101
11. TMRB 102
11.1 Overview 102
11.2 API Functions 102
11.2.1 Function List 102
11.2.2 Detailed Description 103
11.2.3 Function Documentation 103
11.2.4 Data Structure Description 117
12. UART 120
12.1 Overview 120
12.2 API Functions 120
12.2.1 Function List 120
12.2.2 Detailed Description 121
12.2.3 Function Documentation 121
12.2.4 Data Structure Description 141

13. WDT 145

13.1 Overview 145

13.2 API Functions 145

13.2.1 Function List 145

13.2.2 Detailed Description 145

13.2.3 Function Documentation 145

13.2.4 Data Structure Description 149

1. Introduction

TX00 Peripheral Driver is a set of drivers for all peripherals found on the TX00 series of based microcontrollers. TMPM037 Peripheral Driver is an important part of TX00 Peripheral Driver, which is designed for TMPM037 series MCU.

TX00 Peripheral Driver contains a collection of macros, data types, and structures for each peripheral.

The design goals of TOSHIBA TMPM037 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 TX00 Peripheral Driver

/Libraries
This folder contains all CMSIS files and TMPM037 Peripheral Drivers.

/Libraries/ TX00_CMSIS
This folder contains the device peripheral access layer of TMPM037 CMSIS files.

/Libraries/TX00_Periph_Driver
This folder contains all the source code of the drivers, the core of TOSHIBA TMPM037 Peripheral Driver.

/Libraries/TX00_Periph_Driver/inc
This folder contains all the header files of TMPM037 Peripheral Drivers for each peripheral.

/Libraries/TX00_Periph_Driver/src
This folder contains all the source files of TMPM037 Peripheral Drivers for each peripheral.

/Project
This folder contains template project and examples for using TMPM037 Peripheral Driver.

/Project/Template
This folder contains template project of TOSHIBA TMPM037 Peripheral Driver.

/Project/Examples
This folder contains a set of examples for using TMPM037 Peripheral Driver

/Utilities/TMPM037-EVAL
This folder contains the configuration and driver files for hardware resources (e.g. led, key) on TMPM037 boards.

3. ADC

3.1 Overview

This device has 8 channels 10-bit A/D converter, the main functions include:

• Start by an internal or external timer trigger
• Fixed channel/scan mode
• Single/repeat mode
• AD monitoring 2ch

The ADC API provides a set of functions for using the TMPM037 ADC module. 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 TX00_Periph_Driver\src\tmpm037_adc.c(*), with TX00_Periph_Driver\inc\tmpm037_adc.h(*) containing the API definitions for use by applications.

3.2 API Functions

3.2.1 Function List

- void ADC_SWReset(void)
- void ADC_SetClk(uint32_t ***Conversion_Time***, uint32_t ***Prescaler_Output***)
- void ADC_Start(void)
- void ADC_SetScanMode(FunctionalState ***NewState***)
- void ADC_SetRepeatMode(FunctionalState ***NewState***)
- void ADC_SetINTMode(uint32_t ***INTMode***)
- ADC_State ADC_GetConvertState(void)
- void ADC_SetInputChannel(uint32_t ***InputChannel***)
- void ADC_SetChannelScanMode(ADC_ChannelScanMode ***ScanMode***)
- void ADC_SetIdleMode(FunctionalState ***NewState***)
- void ADC_SetVref(FunctionalState ***NewState***)
- void ADC_SetInputChannelTop(uint32_t ***TopInputChannel***)
- void ADC_StartTopConvert(void)
- void ADC_SetMonitor(uint8_t ***ADCMPx***, FunctionalState ***NewState***)
- void ADC_SetResultCmpReg(uint8_t ***ADCMPx***, uint32_t ***ResultComparison***)
- void ADC_SetMonitorINT(uint8_t ***ADCMPx***, ADC_ComparisonState ***NewState***)
- void ADC_SetHWTrg(uint32_t ***HwSource***, FunctionalState ***NewState***)
- void ADC_SetHWTrgTop(uint32_t ***HwSource***, FunctionalState ***NewState***)
- ADC_ResultTypeDef ADC_GetConvertResult (uint32_t ***ADREGx***)
- void ADC_SetCmpValue(uint8_t ***ADCMPx***, uint16_t ***value***)
- void ADC_SetDMAReq(uint8_t ***DMAReq***, FunctionalState ***NewState***)

3.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) ADC setting by ADC_SetClk(), ADC_SetScanMode(), ADC_SetRepeatMode(), ADC_SetINTMode(), ADC_SetInputChannel(), ADC_SetChannelScanMode(), ADC_SetVref(), ADC_SetInputChannelTop(), ADC_SetMonitor(), ADC_SetResultCmpReg(), ADC_SetMonitorINT(), ADC_SetHWTrg(), ADC_SetHWTrgTop(), ADC_SetCmpValue().
2) ADC function start by ADC_Start(), ADC_StartTopConvert().

3) ADC state or data read functions by ADC_GetConvertState(), ADC_GetConvertResult().
4) ADC_SWReset(), ADC_SetDMAReq() and ADC_SetIdleMode() handle other specified functions.

3.2.3 Function Documentation

3.2.3.1 ADC_SWReset

Software reset ADC function.

Prototype:

void

ADC_SWReset(void)

Parameters:

None

Description:

This function will software reset ADC.

***Note:**

A software reset initializes other bits. Re-setting a mode register is needed.

Return:

None

3.2.3.2 ADC_SetClk

Set A/D conversion time and prescaler output.

Prototype:

void

ADC_SetClk(uint32_t ***Conversion_Time***,

uint32_t ***Prescaler_Output***)

Parameters:

Conversion_Time: Select ADC sample hold time.

This parameter can be one of the following values:

- **ADC_CONVERSION_35_CLOCK:** 35.5 conversion clock
- **ADC_CONVERSION_42_CLOCK:** 42 conversion clock
- **ADC_CONVERSION_68_CLOCK:** 68 conversion clock
- **ADC_CONVERSION_81_CLOCK:** 81 conversion clock

Prescaler_Output: Select ADC prescaler output(ADCLK).

This parameter can be one of the following values:

- **ADC_FC_DIVIDE_LEVEL_1:** fc
- **ADC_FC_DIVIDE_LEVEL_2:** fc / 2
- **ADC_FC_DIVIDE_LEVEL_4:** fc / 4
- **ADC_FC_DIVIDE_LEVEL_6:** fc / 6
- **ADC_FC_DIVIDE_LEVEL_8:** fc / 8
- **ADC_FC_DIVIDE_LEVEL_12:** fc / 12
- **ADC_FC_DIVIDE_LEVEL_16:** fc / 16
- **ADC_FC_DIVIDE_LEVEL_24:** fc / 24
- **ADC_FC_DIVIDE_LEVEL_48:** fc / 48
- **ADC_FC_DIVIDE_LEVEL_96:** fc / 96

Description:

This function will set ADC conversion time by ***Conversion_Time*** and prescaler output by ***Prescaler_Output***.

*__Note:__

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

A clock count is required to satisfy the condition that described below.

VREFH AVDD	Conversion time
2.7 to 3.6V	05—1 us or longer
2.3 to 3.6V	21—3 us or longer

Return:

None

3.2.3.3 ADC_Start

Start ADC function.

Prototype:

void

ADC_Start(void)

Parameters:

None

Description:

This function will start AD conversion.

***Note:**

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

Fixed-channel single conversion mode

Channel scan single conversion mode

Fixed-channel repeat conversion mode

Channel scan repeat conversion mode

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

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

Return:

None

3.2.3.4 ADC_SetScanMode

Set ADC scan mode.

Prototype:

void

ADC_SetScanMode(FunctionalState ***NewState***)

Parameters:

NewState: Specify ADC scan mode

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable or disable ADC scan mode by ***NewState*** setting.

Return:

None

3.2.3.5 ADC_SetRepeatMode

Set ADC repeat mode.

Prototype:

void

ADC_SetRepeatMode(FunctionalState ***NewState***)

Parameters:

NewState: Specify ADC repeat mode

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will enable or disable ADC repeat mode by ***NewState*** setting.

Return:

None

3.2.3.6 ADC_SetINTMode

Set ADC interrupt mode in fixed channel repeat conversion mode.

Prototype:

void

ADC_SetINTMode(uint32_t ***INTMode***)

Parameters:

INTMode: Specify AD conversion interrupt mode.

The parameter can be one of the following values:

- **ADC_INT_SINGLE:** Generate in interrupt once every single conversion.
- **ADC_INT_CONVERSION_4:** Generate interrupt once every 4 conversions.
- **ADC_INT_CONVERSION_8:** Generate interrupt once every 8 conversions.

Description:

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

***Note:**

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

Examples for setting fixed channel repeat conversion mode:

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

Return:

None

3.2.3.7 ADC_GetConvertState

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

Prototype:

ADC_State

ADC_GetConvertState(void)

Parameters:

None

Description:

This function will read AD conversion completion / busy flag (both normal and top-priority).

Return:

A union with the state of AD conversion:

NormalBusy(Bit 0) : '1' means normal AD is converting

NormalComplete (Bit 1): '1' means normal AD conversion is completed.

TopBusy(Bit 2) : '1' means top-priority AD is converting

TopComplete (Bit 3): '1' means top-priority AD conversion is completed.

3.2.3.8 ADC_SetInputChannel

Set ADC input channel.

Prototype:

void

ADC_SetInputChannel(uint32_t ***InputChannel***)

Parameters:

InputChannel: Analog input channel, and the input channel also related with other settings.

This parameter can be one of the following values:

ADC_AN_0, ADC_AN_1, ADC_AN_2, ADC_AN_3,

ADC_AN_4, ADC_AN_5, ADC_AN_6, ADC_AN_7.

Description:

This function will specify ADC input channel by ***InputChannel*** setting. And the input channels also relate with mode setting.

In fixed channel mode (**ADC_SetScanMode(DISABLE)**), user can only select one of the 8 channels.

In channel scan mode (**ADC_SetScanMode(ENABLE)**), the input channels is different for 4 channel scan mode

(**ADC_SetChannelScanMode(ADC_SCAN_4CH)**)

and 8 channel scan mode (**ADC_SetChannelScanMode(ADC_SCAN_8CH)**).

***Note:**

Set channel scan mode: **ADC_SetScanMode(ENABLE)**.

Set 4 channel scan mode mode:

ADC_SetChannelScanMode(ADC_SCAN_4CH).

Set 8 channel scan mode mode:

ADC_SetChannelScanMode(ADC_SCAN_8CH).

Return:

None

3.2.3.9 ADC_SetChannelScanMode

Set ADC operation for scanning.

Prototype:

void

ADC_SetChannelScanMode(ADC_ChannelScanMode ***ScanMode***)

Parameters:

ScanMode: Specify operation mode for channel scanning.

The parameter can be one of the following values:

ADC_SCAN_4CH, ADC_SCAN_8CH

Description:

This function will specify different channel scan mode by ***ScanMode*** setting.

***Note:**

This function setting will change the input channel setting

ADC_SetInputChannel(), please refer to the description of **ADC_SetInputChannel()** for details.

Return:

None

3.2.3.10 ADC_SetIdleMode

Set ADC operation in IDLE mode.

Prototype:

void

ADC_SetIdleMode(FunctionalState ***NewState***)

Parameters:

NewState: Specify AD conversion in IDLE mode.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will specify ADC enable or disable in system IDLE mode by ***NewState*** setting.

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

Return:

None

3.2.3.11 ADC_SetVref

Set ADC Vref application control on or off.

Prototype:

void

ADC_SetVref(FunctionalState ***NewState***)

Parameters:

NewState: Specify AD conversion Vref application control.

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This function will specify Vref on or off by ***NewState***.

***Note:**

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

Return:

None

3.2.3.12 ADC_SetInputChannelTop

Set ADC top-priority conversion analog input channel select.

Prototype:

void

ADC_SetInputChannelTop(uint32_t ***TopInputChannel***)

Parameters:

TopInputChannel: Analog input channel for top-priority conversion.

This parameter can be one of the following values:

ADC_AN_0, ADC_AN_1, ADC_AN_2, ADC_AN_3,

ADC_AN_4, ADC_AN_5, ADC_AN_6, ADC_AN_7.

Description:

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

***Note:**

Only one channel of **ADC_AN_0**~**ADC_AN_7** can be selected as top-priority conversion input each time.

Return:

None

3.2.3.13 ADC_StartTopConvert

Start ADC top-priority conversion.

Prototype:

void

ADC_StartTopConvert(void)

Parameters:

None

Description:

This function will start top-priority conversion.

***Note:**

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

Return:

None

3.2.3.14 ADC_SetMonitor

Set ADC monitor function.

Prototype:

void

ADC_SetMonitor(uint8_t ***ADCMPx***,

FunctionalState ***NewState***)

Parameters:

ADCMPx: Select AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

NewState: Specify ADC monitor function

This parameter can be one of the following values:

ENABLE or **DISABLE**.

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

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

Return:

None

3.2.3.15 ADC_SetResultCmpReg

Set ADC result comparison register or comparison register.

Prototype:

void

ADC_SetResultCmpReg(uint8_t ***ADCMPx***,

uint32_t ***ResultComparison***)

Parameters:

ADCMPx: Select AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

ResultComparison: Select the AD conversion result storage register that is to be compared with the comparison register if ADC monitor function is enabled.

The parameter can be one of the following values:

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_SP

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will specify ADC monitor channel by ***ADCMPx*** setting and specify AD conversion result storage register that is to be compared with the comparison register by ***ResultComparison*** setting.

Return:

None

3.2.3.16 ADC_SetMonitorINT

Set ADC monitor interrupt.

Prototype:

void

ADC_SetMonitorINT(uint8_t ***ADCMPx***,

ADC_ComparisonState ***NewState***)

Parameters:

ADCMPx: Selet AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

NewState: Specify ADC monitor function interrupt condition

This parameter can be one of the following values:

ADC_COMPARISON_SMALLER or **ADC_COMPARISON_LARGER**.

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will specify ADC monitor interrupt by ***ADCMPx*** setting and specify ADC monitor function interrupt condition by ***NewState*** setting.

Return:

None

3.2.3.17 ADC_SetHWTrg

Hardware trigger for normal ADC enable or disable and Hardware Source for activating normal ADC setting.

Prototype:

void

ADC_SetHWTrg(uint32_t ***HwSource***,

FunctionalState ***NewState***)

Parameters:

HwSource: Hardware source for activating normal ADC.

This parameter can be one of the following values:

- **ADC_EXT_TRG:** External trigger
- **ADC_MATCH_TB_0:** Match with timer channel 0 register

NewState: enable or disable hardware source for activating normal ADC

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will specify hardware source for activating normal ADC setting by ***HwSource*** setting and specify hardware trigger for normal ADC monitor function

enable or disable by ***NewState*** setting.

This function also has relation with TB0 setting.

***Note:**

The TMPM037 disables the external trigger used for hardware activation. Therefore do not use the function as **ADC_SetHWTrg(ADC_EXT_TRG, NewState)**.

If AD conversion is executed with the match triggers <ADHTG>(Refer to TMPM037 datasheet) and <HADHTG>(Refer to TMPM037 datasheet) of a 16-bit timer set to "1" by using a source for triggering hardware, A/D conversion can be activated at specified intervals by performing three steps shown below when the timer is idle:

- Select a source for triggering hardware.
- Enable hardware activation of AD conversion.
- Start the timer.

Do not make a top-priority AD conversion setting and a normal AD conversion setting simultaneously. Do not use functions **ADC_SetHWTrg(ADC_MATCH_TB_0, ENABLE)** and **ADC_SetHWTrgTop(ADC_MATCH_TB_1, ENABLE)** simultaneously.

Return:

None

3.2.3.18 ADC_SetHWTrgTop

Hardware trigger for top-priority ADC enable or disable and hardware source for activating top-priority ADC setting.

Prototype:

void

ADC_SetHWTrgTop(uint32_t ***HwSource***,

FunctionalState ***NewState***)

Parameters:

HwSource: Hardware source for activating top-priority ADC.

This parameter can be one of the following values:

- **ADC_EXT_TRG:** External trigger
- **ADC_MATCH_TB_1:** Match with timer channel 1 register

NewState: enable or disable hardware source for activating top-priority ADC

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

Description:
This function will specify hardware source for activating top-priority ADC setting by ***HwSource*** setting and specify hardware trigger for top-priority ADC monitor function enable or disable by ***NewState*** setting.
This function also has relation with TB1 setting.

***Note:**
The TMPM037 disables the external trigger used for hardware activation.
Therefore do not use the function as
ADC_SetHWTrgTop(ADC_EXT_TRG, NewState).
Do not make a top-priority AD conversion setting and a normal AD conversion setting simultaneously. Do not use functions
ADC_SetHWTrg(ADC_MATCH_TB_0, ENABLE) and
ADC_SetHWTrgTop(ADC_MATCH_TB_1, ENABLE) simultaneously.

Return:
None

3.2.3.19 ADC_GetConvertResult

Read ADC register's result storage flag state, overrun state and result value.

Prototype:
ADC_ResultTypeDef
ADC_GetConvertResult(uint32_t ***ADREGx***)

Parameters:
ADREGx: Select ADC result register.
The parameter can be one of the following values:
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_SP

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

***Note:**

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

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

AD conversion result is stored in ***ADRGEx*** in different ADC mode as below table.

Table: Analog Input Channels and Related A/D Conversion Result Registers

Analog input channel (port A)	A/D conversion result register			
	Conversion modes other than shown to the right	Fixed channel repeat conversion mode (every one conversion)	Fixed channel repeat conversion mode (every four conversions)	Fixed channel repeat conversion mode (every eight conversions)
ADC_AN_0	ADC_REG_0	ADC_REG_0 fixed	ADC_REG_0—> ADC_REG_3	ADC_REG_0-> ADC_REG_7
ADC_AN_1	ADC_REG_1			
ADC_AN_2	ADC_REG_2			
ADC_AN_3	ADC_REG_3			
ADC_AN_4	ADC_REG_4			
ADC_AN_5	ADC_REG_5			
ADC_AN_6	ADC_REG_6			
ADC_AN_7	ADC_REG_7			

The ADC mode setting, please refer to relate APIs.

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

Return:

ADC result structure:

1. The state of ADC result stored state, which can be
   - **DONE**: AD conversion complete and result stored.
   - **BUSY**: During conversion.
2. The state of normal AD conversion complete, which can be
   - **ADC_NO_OVERRUN**: No Conversion over run.
   - **ADC_OVERRUN**: Conversion over run.
3. ADC result value.

3.2.3.20 ADC_SetCmpValue

Set ADC comparison register value.

Prototype:

void

ADC_SetCmpValue(uint8_t ***ADCMPx***,

uint16_t ***value***)

Parameters:

ADCMPx: Select AD compare register

The parameter can be one of the following values:

ADC_CMP_0 or **ADC_CMP_1**

value: The value set to ADC compare register, max is 0x03ff

Description:

This device has 2ch AD monitor channels: channel 0 and channel 1.

This function will set the ADC compare register value of ADC monitor channel which specify by ***ADCMPx*** setting.

The max setting value should not be larger than 0x03ff for ADC is only 10-bit.

***Note:**

ADC monitor function setting process:

1. **ADC_SetResultCmpReg(ADCMPx, ResultComparison)**
2. **ADC_SetCmpValue(ADCMPx, value)**
3. **ADC_SetMonitorINT(ADCMPx, ResultComparison)**
4. **ADC_SetMonitor(ADCMPx, ENABLE)**

After AD conversion finished, if the condition match **ADC_SetMonitorINT()** setting, ADC monitor interrupt will occurs(The interrupt enable, please refer to interrupt chapter of TMPM037 datasheet).

Return:

None

3.2.3.21 ADC_SetDMAReq

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

Prototype:

void

ADC_SetDMAReq (uint8_t ***DMAReq***,

FunctionalState ***NewState***)

Parameters:

DMAReq: Specify AD conversion DMA request type.

The parameter can be one of the following values:

- **ADC_DMA_REQ_NORMAL:** Specify normal AD conversion DMA activaation factor. (Triggered by INTAD)
- **ADC_DMA_REQ_TOP:** Specify top-priority AD conversion DMA activation factor.(Triggered by INTADHP)
- **ADC_DMA_REQ_MONITOR1:** Specify AD monitor function 0 DMA activation factor.(Triggered by INTADM0)
- **ADC_DMA_REQ_MONITOR2:** Specify AD monitor function 1 DMA activation factor.(Triggered by INTADM1)

NewState: Specify AD conversion DMA activation factor.

The parameter can be one of the following values:

- **ENABLE:** Enable specified DMA activation factor.
- **DISABLE:** Disable specified DMA activation factor

Description:

This function will enable or disable DMA activation factor for normal or top-priority AD conversion.

Return:

None

3.2.4 Data Structure Description

3.2.4.1 ADC_ResultTypeDef

Data Fields:

WorkState

ADCResultStored specifies ADC result storage flag, which can be set as:

- **BUSY**, which means that ADC result has not been stored to the result register;
- **DONE**, which means that which ADC result has been stored to the result register.

ADC_OverrunState

ADCOverrunState specifies ADC overrun flag, which can be set as:

- **ADC_NO_OVERRUN**, which means that ADC is not overrun;

- **ADC_OVERRUN**, which means that ADC is overrun.

uint16_t

ADCResultValue specifies ADC result value,

3.2.4.2 ADC_State

Data Fields for this union:

uint32_t

All specifies AD conversion state.

Bit Fields:

uint32_t

NormalBusy(Bit 0) Normal A/D conversion busy flag (MOD0<ADBFN>).
'1' means conversion is busy

uint32_t

NormalComplete (Bit 1) Normal AD conversion complete flag (MOD0<EOCFN>).
'1' means conversion is completed

uint32_t

TopBusy(Bit 2) Top-priority A/D conversion busy flag (MOD2<ADBFHP>).
'1' means conversion is busy

uint32_t

TopComplete (Bit 3) Top-priority AD conversion complete flag
(MOD2<EOCFHP>).
'1' means conversion is completed

uint32_t

Reserved (Bit 4 to Bit 31) reserved.

4. CG

4.1 Overview

The CG API provides a set of functions for using the TMPM037 CG module as the following:

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

This driver is contained in /TX00_Periph_Driver/src/tmpm037_cg.c, with /TX00_Periph_Driver/inc/tmpm037_cg.h containing the API definitions for use by applications.

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

fosc: Clock from the internal oscillator, or input from X1&X2 pin.
fpll: Clock multiplied by PLL(2 x).
fc: Clock specified by CGPLLSEL<PLLSEL> (high-speed clock)
fgear: Clock specified by CGSYSCR<GEAR[2:0]>.
fsys: Clock specified by CGSYSCR<GEAR[2:0]>.(system clock)
fperiph: Clock specified by CGSYSCR<FPSEL[2:0]>
ΦT0: Clock specified by CGSYSCR<PRCK[2:0]> (prescaler clock)

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(CG_FpllValue ***NewValue***)
- CG_FpllValue CG_GetFPLLValue(void)
- Result CG_SetPLL(FunctionalState ***NewState***)

- FunctionalState CG_GetPLLState(void)
- Result CG_SetFosc(CG_FoscSrc Source, FunctionalState ***NewState***)
- void CG_SetFoscSrc(CG_FoscSrc ***Source***)
- CG_FoscSrc CG_GetFoscSrc(void)
- FunctionalState CG_GetFoscState(CG_FoscSrc ***Source***)
- void CG_SetSTBYMode(CG_STBYMode ***Mode***)
- CG_STBYMode CG_GetSTBYMode(void)
- Result CG_SetFcSrc(CG_FcSrc ***Source***)
- CG_FcSrc CG_GetFcSrc(void)
- void CG_SetProtectCtrl(FunctionalState ***NewState***)
- void CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,
  CG_INTActiveState ***ActiveState***,
  FunctionalState ***NewState***)
- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***)
- void CG_ClearINTReq(CG_INTSrc ***INTSource***)
- CG_NMIFactor CG_GetNMIFlag(void)
- CG_ResetFlag CG_GetResetFlag(void)
- void CG_SetADCClkSupply(FunctionalState NewState)

4.2.2 Detailed Description

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

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

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

This parameter 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

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 clock fgear and fc.

Prototype:

CG_DivideLevel

CG_GetFgearLevel(void)

Parameters:

None

Description:

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

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

Return:

The dividing level between clock fgear and fc.

The value returned can be one of the following values:

CG_DIVIDE_1: fgear = fc

CG_DIVIDE_2: fgear = fc/2

CG_DIVIDE_4: fgear = fc/4

CG_DIVIDE_8: fgear = fc/8

CG_DIVIDE_16: fgear = fc/16

CG_DIVIDE_UNKNOWN: invalid data is read

4.2.3.3 CG_SetPhiT0Src

Set fperiph for PhiT0.

Prototype:

void

CG_SetPhiT0Src(CG_PhiT0Src ***PhiT0Src***)

Parameters:

PhiT0Src: Select PhiT0 source.

This parameter can be one of the following values:

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

Description:

This function selects the source for PhiT0.

Return:

None

4.2.3.4 CG_GetPhiT0Src

Get the PhiT0 source.

Prototype:

CG_PhiT0Src

CG_GetPhiT0Src(void)

Parameters:

None

Description:

This function will get the PhiT0 source.

Return:

CG_PHIT0_SRC_FGEAR means PhiT0 source is fgear.

CG_PHIT0_SRC_FC means PhiT0 source is fc.

4.2.3.5 CG_SetPhiT0Level

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

Prototype:

Result

CG_SetPhiT0Level(CG_DivideLevel ***DividePhiT0FromFc***)

Parameters:

DividePhiT0FromFc: divide level between PhiT0(ΦT0) and fc.

This parameter can be one of the following values:

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

Description:

This function will set the dividing level of prescaler clock.

Return:

SUCCESS means the setting has been written to registers successfully.

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

4.2.3.6 CG_GetPhiT0Level

Get the dividing level between clock ΦT0 and fc.

Prototype:

CG_DivideLevel

CG_GetPhiT0Level(void)

Parameters:

None

Description:

This function will get the dividing level of prescaler clock.

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

CG_DIVIDE_UNKNOWN.

Return:

Dividing level between clock ΦT0 and fc, the value will be one of the following:

CG_DIVIDE_1: ΦT0 = fc

CG_DIVIDE_2: ΦT0 = fc/2

CG_DIVIDE_4: ΦT0 = fc/4

CG_DIVIDE_8: ΦT0 = fc/8

CG_DIVIDE_16: ΦT0 = fc/16

CG_DIVIDE_32: ΦT0 = fc/32

CG_DIVIDE_64: ΦT0 = fc/64

CG_DIVIDE_128 : ΦT0 = fc/128

CG_DIVIDE_256 : ΦT0 = fc/256

CG_DIVIDE_512 : ΦT0 = fc/512

CG_DIVIDE_UNKNOWN : invalid data is read.

4.2.3.7 CG_SetWarmUpTime

Set the warm up time.

Prototype:

void

CG_SetWarmUpTime(CG_WarmUpSrc ***Source***,

uint16_t ***Time***)

Parameters:

Source: select source of warm-up counter.

- **CG_WARM_UP_SRC_OSC_INT_HIGH**: internal high-speed oscillator is selected as timer source.
- **CG_WARM_UP_SRC_OSC_EXT_HIGH**: external high-speed oscillator is selected as timer source.

Time:

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

Description:

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

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

Example of calculating register value for warm-up time:

/* When using high-speed oscillator 10MHz, and set warm-up time 5ms. */

So value = (warm-up time to set) / (input frequency cycle(s)) = 5ms / (1/10MHz) = 5000cycle = 0xC350.

Round lower 4 bit off, set 0xC35 to CGOSCCR<WUODR[11:0]>

Return:

None.

4.2.3.8 CG_StartWarmUp

Start operation of warm up timer for oscillator.

Prototype:

void

CG_StartWarmUp(void)

Parameters:

None

Description:

This function will start the warm up timer.

Return:

None

4.2.3.9 CG_GetWarmUpState

Check whether warm up is completed or not.

Prototype:

WorkState

CG_GetWarmUpState(void)

Parameters:

None

Description:

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

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

Return:

Warm up state:

DONE: means warm-up operation is finished.

BUSY: means warm-up operation is in progress.

4.2.3.10 CG_SetFPLLValue

Set PLL multiplying value

Prototype:

Result

CG_SetFPLLValue(uint32_t ***NewValue***)

Parameters:

NewValue:

- **CG_MUL_2_FPLL:** 2 times multiplying

Description:

This function sets PLL multiplying value.

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

***Note**: When the external high-speed oscillation is upper than 10MHz, you can't use PLL.

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.

If the other value is read from the register, it means the value is reserved.

Return:

The source of PLL multiplying value

- **CG_MUL_2_FPLL:** 2 times 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.

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 high-speed oscillator (fosc).

Prototype:

Result

CG_SetFosc(CG_FoscSrc ***Source***,

FunctionalState ***NewState***)

Parameters:

Source: select clock source of fosc.

This parameter can be one of the following values:

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

NewState

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

Description:

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

Return:

SUCCESS: operation is finished successfully.

ERROR: operation is not done.

4.2.3.15 CG_SetFoscSrc

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

Prototype:

void

CG_SetFoscSrc(CG_FoscSrc Source)

Parameters:

Source: select source for fosc.

This parameter can be one of the following values:

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

Description:

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

Return:

None

4.2.3.16 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_OSC_EXT: external high-speed oscillator is selected,

CG_FOSC_CLKIN_EXT: external clock input is selected.

CG_FOSC_OSC_INT: internal high-speed oscillator is selected.

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

Description:

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

Return:

The state of fosc

ENABLE: fosc is enabled.

DISABLE: fosc is disabled.

4.2.3.18 CG_SetSTBYMode

Set the specified low-power mode.

Prototype:

void

CG_SetSTBYMode(CG_STBYMode ***Mode***)

Parameters:

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

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

4.2.3.19 CG_GetSTBYMode

Get the low-power consumption mode.

Prototype:

CG_STBYMode

CG_GetSTBYMode(void)

Parameters:

None

Description:

This function will get the setting of standby mode.

If the value “Reserved” is read, “**CG_STBY_MODE_UNKNOWN**” will be returned.

Return:

The low power mode:

CG_STBY_MODE_STOP1: STOP1 mode.

CG_STBY_MODE_IDLE: IDLE mode

CG_STBY_MODE_UNKNOWN: Invalid data is read.

4.2.3.20 CG_SetFcSrc

Set the clock source of fc

Prototype:

Result

CG_SetFcSrc(CG_FcSrc ***Source***)

Parameters:

Source: the source for fc

This parameter can be one of the following values:

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

Description:

This function will set the clock source of fc.

Return:

SUCCESS: set clock souce for fc successfully

ERROR: clock source of fc is not changed.

4.2.3.21 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.

4.2.3.22 CG_SetProtectCtrl

Enable or disable to protect CG registers.

Prototype:

void

CG_SetProtectCtrl(FunctionalState ***NewState***)

Parameters:

NewState

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

Description:

This function enables or disables CG registers to be written.

Return:

None

4.2.3.23 CG_SetSTBYReleaseINTSrc

Set the INT source for releasing low power mode.

Prototype:

void

CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,

CG_INTActiveState ***ActiveState***,

FunctionalState ***NewState***)

Parameters:

INTSource: select the INT source for releasing standby mode

This parameter can be one of the following values:

- **CG_INT_SRC_0** : INT0
- **CG_INT_SRC_1** : INT1
- **CG_INT_SRC_2** : INT2
- **CG_INT_SRC_3** : INT3
- **CG_INT_SRC_4** : INT4
- **CG_INT_SRC_5** : INT5

ActiveState: select the active state for release trigger.

This parameter can be one of the following values:

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

NewState: enable or disable this release trigger

This parameter can be one of the following values:

1) **ENABLE**: clear standby mode when the interrupt occurs and the condition of active state is matched.
2) **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

4.2.3.24 CG_GetSTBYReleaseINTState

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

Prototype:

CG_INTActiveState

CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***)

Parameters:

INTSource: select the release INT source

This parameter can be one of the following values:

CG_INT_SRC_0, CG_INT_SRC_1, CG_INT_SRC_2,
CG_INT_SRC_3, CG_INT_SRC_4, CG_INT_SRC_5.

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

4.2.3.25 CG_ClearINTReq

Clears the input INT request.

Prototype:

void

CG_ClearINTReq(CG_INTSrc ***INTSource***)

Parameters:

INTSource: select the release INT source.

This parameter can be one of the following values:

CG_INT_SRC_0, CG_INT_SRC_1, CG_INT_SRC_2,
CG_INT_SRC_3, CG_INT_SRC_4, CG_INT_SRC_5.

Description:

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

Return:

None

4.2.3.26 CG_GetNMIFlag

Get the NMI flag that shows who triggered NMI

Prototype:

CG_NMIFactor

CG_GetNMIFlag (void)

Parameters:

None

Description:

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

Return:

NMI value:

WDT (Bit 0) Means generated from WDT.

DetectLowVoltage1 (Bit 2) Means generated from LVD, only lower than the setting voltage when voltage decreasing.

DetectLowVoltage2 (Bit 3) Means generated from LVD when returning from low voltage.

4.2.3.27 CG_GetResetFlag

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

Prototype:

CG_ResetFlag

CG_GetResetFlag(void)

Parameters:

None

Description:

This function gets the reset flag showing what triggered reset.

Return:

Reset flag:

PowerOnReset (Bit0) Reset from Power-On-Reset

PinReset (Bit1) Reset from RESET pin

WDTReset (Bit2) Reset from WDT

DebugReset (Bit4) Reset from SYSRESETREQ

LVDReset (Bit6) Reset from LVD

4.2.3.28 CG_SetADCClkSupply

Enable or disable supplying clock fsys for ADC.

Prototype:

void

CG_SetADCClkSupply(FunctionalState ***NewState***)

Parameters:

NewState: New state of clock fsys supply setting for ADC.

This parameter can be one of the following values:

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

Description:

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

Return:

None

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.

uint32_t

Reserved1 (bit1) Reserved

uint32_t

DetectLowVoltage1 (Bit 2) Means generated from LVD, only lower than the setting voltage when voltage decreasing.

uint32_t

DetectLowVoltage2 (Bit3) Means generated from LVD when returning from low voltage.

Reserved2 (Bit4~Bit31) Reserved

4.2.4.2 CG_ResetFlag

Data Fields:

uint32_t

All specifies CG reset source.

Bit Fields:

uint32_t

PowerOnReset (Bit0) Reset from Power-On-Reset

uint32_t

PinReset (Bit1) Reset from RESET pin

uint32_t

WDTReset (Bit2) Reset from WDT

uint32_t

Reververd1 (Bit3) Reserved

uint32_t

DebugReset (Bit4) Reset from SYSRESETREQ

uint32_t

Reververd2 (Bit5) Reserved

uint32_t

LVDReset (Bit6) Reset from LVD

uint32_t

Reserved3 (Bit7~bit31) Reserved

5. DMAC

5.1 Overview

TOSHIBA TMPM037 has only one DMA controller controlled by DMA request select registers, and each DMA controller has two channels. Each channel can operate in one of four transferring types (memory to memory, memory to peripheral, peripheral to memory, peripheral to peripheral).The priority of channel 0 is higher than channel 1.

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

This driver is contained in \Libraries\TX00_Periph_Driver\src\tmpm037_dmac.c, with \Libraries/TX00_Periph_Driver\inc\tmpm037_dmac.h containing the API definitions for use by applications.

5.2 API Functions

5.2.1 Function List

- void DMAC_Enable(void);
- void DMAC_Disable(void);
- DMAC_INTReq DMAC_GetINTReq(void);
- DMAC_TxINTReq DMAC_GetTxINTReq(DMAC_Channel ***Chx***);
- void DMAC_ClearTxINTReq(DMAC_Channel ***Chx***, DMAC_INTSrc ***INTSource***);
- DMAC_TxINTReq DMAC_GetRawTxINTReq(DMAC_Channel ***Chx***);
- WorkState DMAC_GetChannelTxState(DMAC_Channel ***Chx***);
- void DMAC_SetSWBurstReq(DMAC_ReqNum ***BurstReq***);
- DMAC_BurstReqState DMAC_GetSWBurstReqState(void);
- void DMAC_SetLinkedList(DMAC_Channel ***Chx***, uint32_t ***LinkedAddr***);
- WorkState DMAC_GetFIFOState(DMAC_Channel ***Chx***);
- void DMAC_SetDMAHalt(DMAC_Channel ***Chx***, FunctionalState ***NewState***);
- void DMAC_SetLockedTx(DMAC_Channel ***Chx***, FunctionalState ***NewState***)
- void DMAC_SetTxINTConfig(DMAC_Channel ***Chx***, DMAC_INTSrc ***INTSource***, FunctionalState ***NewState***);
- void DMAC_SetDMAChannel(DMAC_Channel ***Chx***, FunctionalState ***NewState***);
- void DMAC_Init(DMAC_Channel ***Chx***, DMAC_InitTypeDef * ***InitStruct***);

5.2.2 Detailed Description

Functions listed above can be divided into five parts:

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

5.2.3 Function Documentation

5.2.3.1 DMAC_Enable

Enable the DMA circuit.

Prototype:

void

DMAC_Enable(void);

Description:

This function will Enable DMA circuit.

***Note:**

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

Return:

None

5.2.3.2 DMAC_Disable

Disable the DMA circuit.

Prototype:

void

DMAC_Disable(void);

Description:

This function will disable DMA circuit.

Return:

None

5.2.3.3 DMAC_GetINTReq

Get DMA Channel interrupt request state.

Prototype:

DMAC_INTReq

DMAC_GetINTReq(void);

Description:

This function will get DMA Channel interrupt request state.

Return:

The state of interrupt request.

5.2.3.4 DMAC_GetTxINTReq

Get the specified DMA Channel transfer interrupt request state.

Prototype:

DMAC_TxINTReq

DMAC_GetTxINTReq(DMAC_Channel ***Chx***);

Parameters:

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

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

Description:

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

Return:

The request states of DMA transfer interrupt.

The value returned can be one of the followings:

DMAC_TX_NO_REQ means there is no transfer interrupt request,

DMAC_TX_END_REQ means there is a transfer end interrupt request,

DMAC_TX_ERR_REQ means there is a transfer error interrupt request,

DMAC_TX_REQS means there is more than one interrupt request.

5.2.3.5 DMAC_ClearTxINTReq

Clear the transfer interrupt request.

Prototype:

void

DMAC_ClearTxINTReq(DMAC_Channel ***Chx***,

DMAC_INTSrc ***INTSource***);

Parameters:

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

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

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

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

Description:

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

Return:

None

5.2.3.6 DMAC_GetRawTxINTReq

Get the specified DMA Channel transfer raw interrupt request state.

Prototype:

DMAC_TxINTReq

DMAC_GetRawTxINTReq(DMAC_Channel ***Chx***);

Parameters:

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

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

Description:

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

Return:

The request states of DMA transfer raw interrupt.

The value returned can be one of the followings:

DMAC_TX_NO_REQ means there is no transfer raw interrupt request,

DMAC_TX_END_REQ means there is a transfer end interrupt request,

DMAC_TX_ERR_REQ means there is a transfer error interrupt request,

DMAC_TX_REQS means there is more than one interrupt request.

5.2.3.7 DMAC_GetChannelTxState

Get the specified DMA Channel transfer state.

Prototype:

WorkState

DMAC_GetChannelTxState(DMAC_Channel ***Chx***);

Parameters:

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

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

Description:

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

Return:

The DMA transfer status.

The value returned can be one of the followings:

BUSY or **DONE**

5.2.3.8 DMAC_SetSWBurstReq

Set DMA burst transfer requests by software.

Prototype:

void

DMAC_SetSWBurstReq(DMACA_ReqNum ***BurstReq***);

Parameters:

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

- **DMAC_SIO0_UART0_RX** for SIO0/UART0 Reception,
- **DMAC_SIO0_UART0_TX** for SIO0/UART0 Transmission,
- **DMAC_SIO1_UART1_RX** for SIO1/UART1 Reception,
- **DMAC_SIO1_UART1_TX** for SIO1/UART1 Transmission,
- **DMAC_SIO2_UART2_RX** for SIO2/UART2 Reception,
- **DMAC_SIO2_UART2_TX** for SIO2/UART2 Transmission,
- **DMAC_SIO3_UART3_RX** for SIO3/UART3 Reception,
- **DMAC_SIO3_UART3_TX** for SIO3/UART3 Transmission,
- **DMAC_I2C0_RX_TX** for I2C Reception and Transmission,
- **DMAC_SIO4_UART4_RX** for SIO4/UART4 Reception,
- **DMAC_SIO4_UART4_TX** for SIO4/UART4 Transmission,
- **DMAC_TMRB0_3** for TMRB channel 0 to channel 3
- **DMAC_TMRB4_7** for TMRB channel 4 to channel 3,
- **DMAC_TOP_PRIORITY_ADC** for Top priority A/D,
- **DMAC_AD_CONVERT_COMPLETE** for AD convert complete.

Description:

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

Return:

None

5.2.3.9 DMAC_GetSWBurstReqState

Get DMA software burst request state.

Prototype:

DMAC_BurstReqState

DMAC_GetSWBurstReqState(void);

Description:

This function will get DMA software burst request state.

Return:

The DMA burst request status.

5.2.3.10 DMAC_SetLinkedList

Set specified DMA Channel Linked List Item Register.

Prototype:

void

DMAC_SetLinkedList(DMAC_Channel ***Chx***,

uint32_t ***LinkedAddr***);

Parameters:

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

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

LinkedAddr: The start address of the next transfer information.

Max 0xFFFFFFF0.

Description:

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

***Note:**

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

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

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

1) DMACxCnSrcAddr

2) DMACxCnDestAddr

3) DMACxCnLLI

4) DMACxCnControl

Return:

None

5.2.3.11 DMAC_GetFIFOState

Indicates whether data is present in the channel FIFO

Prototype:

WorkState

DMAC_GetFIFOState(DMAC_Channel ***Chx***);

Parameters:

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

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

Description:

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

Return:

The FIFO status

The value returned can be one of the followings:

BUSY or **DONE**

5.2.3.12 DMAC_SetDMAHalt

Set whether ignore DMA request.

Prototype:

void

DMAC_SetDMAHalt(DMAC_Channel ***Chx***,

FunctionalState ***NewState***);

Parameters:

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

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

NewState: New state of DMA halt.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will set DMA specified Channel ignore DMA request.

Return:

None

5.2.3.13 DMAC_SetLockedTx

Set whether locked transfer.

Prototype:

void

DMAC_SetLockedTx(DMAC_Channel ***Chx***,

FunctionalState ***NewState***);

Parameters:

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

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

NewState: New state of DMA transfer.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

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

Return:

None

5.2.3.14 DMAC_SetTxINTConfig

Enable or disable the specified DMA Channel transfer interrupt.

Prototype:

void

DMAC_SetTxINTConfig(DMAC_Channel ***Chx***,

DMAC_INTSrc ***INTSource***,

FunctionalState ***NewState***);

Parameters:

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

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

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

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

NewState: New states of DMA transfer interrupt.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable or disable DMA specified Channel transfer end interrupt when ***INTSource*** is **DMAC_INT_TX_END** and enable or disable DMA specified Channel transfer error interrupt when ***INTSource*** is **DMAC_INT_TX_ERR**.

Return:

None

5.2.3.15 DMAC_SetDMAChannel

Enable or disable the specified DMA Channel.

Prototype:

void

DMAC_SetDMAChannel(DMAC_Channel ***Chx***,

FunctionalState ***NewState***);

Parameters:

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

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

NewState: New state of DMA channel.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

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

Return:

None

5.2.3.16 DMAC_Init

Initialize the specified DMA channel.

Prototype:

```
void
DMAC_Init(DMAC_Channel Chx,
                    DMAC_InitTypeDef * InitStruct);
```

Parameters:

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

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

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

Description:

This function will initialize and configure the source address, source address incremented state, transfer source bit width, transfer source burst size, destination address, destination address incremented state, transfer destination bit width, transfer destination burst size, transfer size, and transfer direction,

transfer peripheral and transfer interrupt state for the DMA specified channel.

***Note:**

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

Return:

None

5.2.4 Data Structure Description

5.2.4.1 DMAC_InitTypeDef

Data Fields:

uint32_t

TxDirection Set transfer direction, which can be set as:

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

uint32_t

SrcAddr Set source address.

uint32_t

DstAddr Set destination address.

FunctionalState

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

ENABLE or **DISABLE**.

FunctionalState

DstIncrementState Specifies whether the destination address is incremented or not,

which can be set as:

ENABLE or **DISABLE**.

DMAC_BitWidth

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

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

- **DMAC_WORD** means transfer source bit width set as word.

DMAC_BitWidth

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

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

DMAC_BurstSize

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

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

DMAC_BurstSize

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

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

uint32_t

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

DMAC_ReqNum

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

- **DMAC_SIO0_UART0_RX** for SIO3/UART3 Reception.
- **DMAC_SIO0_UART0_TX** for SIO3/UART3 Transmission.
- **DMAC_SIO1_UART1_RX** for SIO3/UART3 Reception.
- **DMAC_SIO1_UART1_TX** for SIO3/UART3 Transmission.
- **DMAC_SIO2_UART2_RX** for SIO3/UART3 Reception.
- **DMAC_SIO2_UART2_TX** for SIO0/UART0 Transmission.
- **DMAC_SIO3_UART3_RX** for SIO0/UART0 Reception.
- **DMAC_SIO3_UART3_TX** for SIO0/UART0 Transmission.
- **DMAC_I2C0_RX_TX** for I2C Reception and Transmission.
- **DMAC_SIO4_UART4_RX** for SIO4/UART4 Reception.
- **DMAC_SIO4_UART4_TX** for SIO4/UART4 Transmission.

- **DMAC_TMRB0_3** for TMRB channel 0 to channel 3.
- **DMAC_TMRB4_7** for TMRB channel 4 to channel 7.
- **DMAC_TOP_PRIORITY_ADC** for Top-priority AD conversion completion.
- **DMAC_AD_CONVERT_COMPLETE** for AD conversion completion.

DMAC_ReqNum

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

- **DMAC_SIO0_UART0_RX** for SIO3/UART3 Reception.
- **DMAC_SIO0_UART0_TX** for SIO3/UART3 Transmission.
- **DMAC_SIO1_UART1_RX** for SIO3/UART3 Reception.
- **DMAC_SIO1_UART1_TX** for SIO3/UART3 Transmission.
- **DMAC_SIO2_UART2_RX** for SIO3/UART3 Reception.
- **DMAC_SIO2_UART2_TX** for SIO0/UART0 Transmission.
- **DMAC_SIO3_UART3_RX** for SIO0/UART0 Reception.
- **DMAC_SIO3_UART3_TX** for SIO0/UART0 Transmission.
- **DMAC_I2C0_RX_TX** for I2C Reception and Transmission.
- **DMAC_SIO4_UART4_RX** for SIO4/UART4 Reception.
- **DMAC_SIO4_UART4_TX** for SIO4/UART4 Transmission.
- **DMAC_TMRB0_3** for TMRB channel 0 to channel 3.
- **DMAC_TMRB4_7** for TMRB channel 4 to channel 7.
- **DMAC_TOP_PRIORITY_ADC** for Top-priority AD conversion completion.
- **DMAC_AD_CONVERT_COMPLETE** for AD conversion completion.

FunctionalState

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

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

5.2.4.2 DMAC_INTReq

Data Fields for this union:

uint32_t

ALL DMAC channel interrupt status.

- **CH0_INTReq**: DMAC channel0 interrupt status.
- **CH1_INTReq**: DMAC channel1 interrupt status.

5.2.4.3 DMAC_ BurstReqState

Data Fields for this union:

uint32_t

ALL DMAC request status.

- **DMAC_SIO0_UART0_RX** for SIO3/UART3 Reception.
- **DMAC_SIO0_UART0_TX** for SIO3/UART3 Transmission.
- **DMAC_SIO1_UART1_RX** for SIO3/UART3 Reception.
- **DMAC_SIO1_UART1_TX** for SIO3/UART3 Transmission.
- **DMAC_SIO2_UART2_RX** for SIO3/UART3 Reception.
- **DMAC_SIO2_UART2_TX** for SIO0/UART0 Transmission.

- **DMAC_SIO3_UART3_RX** for SIO0/UART0 Reception.
- **DMAC_SIO3_UART3_TX** for SIO0/UART0 Transmission.
- **DMAC_I2C0_RX_TX** for I2C Reception and Transmission.
- **DMAC_SIO4_UART4_RX** for SIO4/UART4 Reception.
- **DMAC_SIO4_UART4_TX** for SIO4/UART4 Transmission.
- **DMAC_TMRB0_3** for TMRB channel 0 to channel 3.
- **DMAC_TMRB4_7** for TMRB channel 4 to channel 7.
- **DMAC_TOP_PRIORITY_ADC** for Top-priority AD conversion completion.
- **DMAC_AD_CONVERT_COMPLETE** for AD conversion completion.

6. FC

6.1 Overview

TMPM037 device contains flash memory, the flash size is 128Kbytes.

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\TX00_Periph_Driver\src\tmpm037_fc.c with \Libraries\TX00_Periph_Driver \inc\ tmpm037_fc.h containing the API definitions for use by applications.

6.2 API Functions

6.2.1 Function List

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

6.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) The security function restricts flash ROM data readout and debugging.

   FC_SetSecurityBit(), FC_GetSecurityBit().

2) The functions get the automatic operation status and each block protection status:

   FC_GetBusyState(), FC_GetBlockProtectState().

3) The functions change the protection status of each block:

   FC_ProgramBlockProtectState(), FC_EraseBlockProtectState().

4) Use automatic operation command to write or erase the content of flash.

   FC_WritePage(), FC_EraseBlock(), FC_EraseChip().

6.2.3 Function Documentation

6.2.3.1 FC_SetSecurityBit

Set the value of SECBIT register.

Prototype:

void

FC_SetSecurityBit (FunctionalState ***NewState***)

Parameters:

NewState: Select the state of SECBIT register.

This parameter can be one of the following values:

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

Description:

1) All the protection bits (the FCPSRA<BLK [3:0]> bits) used for the write/erase-protection function are set to “1”.
2) The SECBIT <SECBIT> bit is set to”1”.

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

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

Return:

None

6.2.3.2 FC_GetSecurityBit

Get the value of SECBIT register.

Prototype:

FunctionalState
FC_GetSecurityBit(void)

Parameters:

None

Description:

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

Return:

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

6.2.3.3 FC_GetBusyState

Get the status of the flash auto operation.

Prototype:

WorkState
FC_GetBusyState (void)

Parameters:

None

Description:

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

Return:

Status of the flash automatic operation:

BUSY: Flash memory is in automatic operation.

DONE: Automatic operation is normally terminated. The next command can be sent and executed.

6.2.3.4 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,
- **FC_BLOCK_2** for block 2,
- **FC_BLOCK_3** for block 3,

Description:

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

Return:

Block protection status.

DISABLE: Block is unprotected

ENABLE: Block is protected

6.2.3.5 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,
- **FC_BLOCK_2** for block 2,
- **FC_BLOCK_3** for block 3,

Description:

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

Return:

Result of the operation to program the protection bit.

FC_SUCCESS: Set the protection bit to “1” successfully.

FC_ERROR_PROTECTED: The protection bit is “1” already, and it doesn’t need to program it again.

FC_ERROR_OVER_TIME: Program block protection bit operation over time error.

6.2.3.6 FC_EraseBlockProtectState

Erase the protection bits.

Prototype:

FC_Result

FC_EraseBlockProtectState(uint8_t ***BlockGroup***)

Parameters:

BlockGroup: The flash block group

- **FC_BLOCK_GROUP_0** for block 0 through 3.

Description:

This API is used to erase the protection bits (clear them to”0”) so that the

corresponding blocks will not be protected.
One group of protection bits will be erased when this API is executed each time.

Return:
Result of the operation to erase the protection bits.
FC_SUCCESS: Erase the protection bits successfully.

FC_ERROR_OVER_TIME: Erase block protection bits operation over time error.

6.2.3.7 FC_WritePage

Write data to the specified page.

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

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

Description:
This API is used to write data to specified page.
The TMPM037 contains 32 words in a page. The flash can only be written page by page.
The automatic page programming is allowed only once for a page already erased. No programming can be performed twice or more time irrespective of data value whether it is “1” or “0”.
***Note**: An attempt to rewrite a page two or more times without erasing the content can cause damages to the device.

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

6.2.3.8 FC_EraseBlock

Erase the content of specified block.

Prototype:

FC_Result

FC_EraseBlock(uint32_t ***BlockAddr***)

Parameters:

BlockAddr: The block starts address.

Description:

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

be erased.

Return:

Result of the operation to erase the content of specified block.

FC_SUCCESS: the content of the specified block is erased successfully.

FC_ERROR_PROTECTED: The block is protected. The erase operation can't be executed. The block will not be erased.

FC_ERROR_OVER_TIME: Erase operation over time error.

6.2.3.9 FC_EraseChip

Erase the content of the entire chip.

Prototype:

FC_Result

FC_EraseChip(void)

Parameters:

None

Description:

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

Return:

Result of the operation to erase the content of the entire chip.

FC_SUCCESS: If all the blocks are unprotected, the entire chip is erased. If parts of blocks are protected, only unprotected blocks are erased.

FC_ERROR_PROTECTED: All blocks are protected. The erase chip operation can't be executed.

FC_ERROR_OVER_TIME: Erase Chip operation over time error.

6.2.4 Data Structure Description

None

7. GPIO

7.1 Overview

For TOSHIBA TMPM037 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/TX00_Periph_Driver/src/ tmpm037 _gpio.c, with /Libraries/TX00_Periph_Driver/inc/tmpm037 _gpio.h containing the macros, data types, structures and API definitions for use by applications.

7.2 API Functions

7.2.1 Function List

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

7.2.2 Detailed Description

Functions listed above can be divided into three parts:

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

7.2.3 Function Documentation

7.2.3.1 GPIO_ReadData

Read specified GPIO Data register.

Prototype:

uint8_t

GPIO_ReadData(GPIO_Port ***GPIO_x***)

Parameters:

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

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

Description:

This function will read specified GPIO Data register.

Return:

The value read from DATA register.

7.2.3.2 GPIO_ReadDataBit

Read specified GPIO pin.

Prototype:

uint8_t

GPIO_ReadDataBit(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***)

Parameters:

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

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

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

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

Description:

This function will read specified GPIO pin.

Return:

The value read from GPIO pin as:

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

7.2.3.3 GPIO_WriteData

Write specified value to GPIO Data register.

Prototype:

void

GPIO_WriteData(GPIO_Port ***GPIO_x***,

uint8_t ***Data***)

Parameters:

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

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

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

Description:

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

Return:

None

7.2.3.4 GPIO_WriteDataBit

Write specified value of single bit to GPIO pin.

Prototype:

void

GPIO_WriteDataBit(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***,

uint8_t ***BitValue***)

Parameters:

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

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

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

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

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

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

Description:

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

Return:

None

7.2.3.5 GPIO_Init

Initialize GPIO port function.

Prototype:

void

GPIO_Init(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***,

GPIO_InitTypeDef * ***GPIO_InitStruct***)

Parameters:

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

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

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

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

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

Description:

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

Return:

None

7.2.3.6 GPIO_SetOutput

Set specified GPIO pin as output port.

Prototype:

void

GPIO_SetOutput(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***)

Parameters:

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

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

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

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

Description:

This function will set specified GPIO pin as output port.

Return:

None

7.2.3.7 GPIO_SetInput

Set specified GPIO Pin as input port.

Prototype:

void

GPIO_SetInput(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***)

Parameters:

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

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

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

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

Description:

This function will set specified GPIO pin as input port.

Return:

None

7.2.3.8 GPIO_SetOutputEnableReg

Enable or disable specified GPIO Pin output function.

Prototype:

void

GPIO_SetOutputEnableReg(GPIO_Port ***GPIO_x***,

uint8_t ***Bit_x***,

FunctionalState ***NewState***)

Parameters:

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

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.

- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.

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

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

NewState:

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

Description:

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

Return:

None

7.2.3.9 GPIO_SetInputEnableReg

Enable or disable specified GPIO Pin input function.

Prototype:

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

Parameters:

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

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.

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

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

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

NewState:

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

Description:

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

Return:

None

7.2.3.10 GPIO_SetPullUp

Enable or disable specified GPIO Pin pull-up function.

Prototype:

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

Parameters:

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

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.

- **GPIO_PF:** GPIO port F.
- **GPIO_PG:** GPIO port G.

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

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

NewState:

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

Description:

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

Return:

None

7.2.3.11 GPIO_SetPullDown

Enable or disable specified GPIO Pin pull-down function.

Prototype:

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

Parameters:

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

- **GPIO_PA:** GPIO port A.
- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **GPIO_PE:** GPIO port E.
- **GPIO_PF:** GPIO port F.

- **GPIO_PG:** GPIO port G.

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

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

NewState:

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

Description:

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

Return:

None

7.2.3.12 GPIO_SetOpenDrain

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

Prototype:

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

Parameters:

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

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

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

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

NewState:

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

Description:

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

Return:

None

7.2.3.13 GPIO_EnableFuncReg

Enable specified GPIO function.

Prototype:

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

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **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,

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

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

Description:

This function will enable GPIO pin specified function.

Return:

None

7.2.3.14 GPIO_DisableFuncReg

Disable specified GPIO function.

Prototype:

void

GPIO_DisableFuncReg(GPIO_Port ***GPIO_x***,

uint8_t ***FuncReg_x***,

uint8_t ***Bit_x***)

Parameters:

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

- **GPIO_PB:** GPIO port B.
- **GPIO_PC:** GPIO port C.
- **GPIO_PD:** GPIO port D.
- **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,

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

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

Description:

This function will disable GPIO pin specified function.

Return:

None

7.2.4 Data Structure Description

7.2.4.1 GPIO_InitTypeDef

Data Fields:

uint8_t

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

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

uint8_t

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

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

uint8_t

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

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

uint8_t

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

- **GPIO_PULLDOWN_ENABLE:** Enable specified GPIO pin pull-down function**.**
- **GPIO_PULLDOWN_DISABLE:** Disable specified GPIO pin pull-down function**.**

- **GPIO_PULLDOWN_NONE:** Don't have pull-down function or needn't change.

7.2.4.2 GPIO_RegTypeDef

Data Fields:

uint8_t

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

uint8_t

PinCR Port x output control register:

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

uint8_t

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

uint8_t

PinOD Port x open drain control register:

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

uint8_t

PinPUP Port x pull-up control register:

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

uint8_t

PinPDN Port x pull-down control register:

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

uint8_t

PinPIE Port x input control register:

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

7.2.4.3 TSB_Port_TypeDef

Data Fields:

__IO uint32_t

DATA The "DATA" can be read and written.

__IO uint32_t

PinCR The "CR" can be read and written.

__IO uint32_t

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

uint32_t

RESERVED0 [RESER] Reserved

__IO uint32_t

PinOD The “OD” can be read and written.

__IO uint32_t

PinPUP The “PUP” can be read and written.

__IO uint32_t

PinPDN The “PDN” can be read and written.

uint32_t

RESERVED1 [RESER] Reserved

__IO uint32_t

PinPIE Port x input control register.

8. I2C

8.1 Overview

The TMPM037 contains I2C Bus Interface with 1 channel.
The I2C bus is connected to external devices via SCL and SDA, and it can communicate With multiple devices.
Data can be transferred in free data format by the I2C channels. In free data format, data is always sent by master-transmitter and received by slave-receiver.

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

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm037_i2c.c, with /Libraries/TX00_Periph_Driver/inc/tmpm037_i2c.h containing the macros, data types, structures and API definitions for use by applications.

8.2 API Functions

8.2.1 Function List

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

8.2.2 Detailed Description

Functions listed above can be divided into four parts:

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

8.2.3 Function Documentation

8.2.3.1 I2C_SetACK

Enable or disable the generation of ACK clock.

Prototype:

void

I2C_SetACK(FunctionalState ***NewState***)

Parameters:

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

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

Description:

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

Return:

None

8.2.3.2 I2C_Init

Initialize the I2C channel in I2C mode.

Prototype:

void

I2C_Init(I2C_InitTypeDef* ***InitI2CStruct***)

Parameters:

InitI2CStruct is the structure containing I2C configuration (refer to Data

Structure Description for details).

Description:

This function will initialize and configure the self-address, bit length of transfer data, clock division, the generation of ACK clock and the operation mode of I2C transfer for the I2C channel.

Return:

None

8.2.3.3 I2C_SetBitNum

Specify the number of bits per transfer.

Prototype:

void

I2C_SetBitNum(uint32_t ***I2CBitNum***)

Parameters:

I2CBitNum specifies the number of bits per transfer, max. 8.

This parameter can be one of the following values:

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

Description:

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

Return:

None

8.2.3.4 I2C_SWReset

Reset the state of the I2C channel.

Prototype:

void

I2C_SWReset(void)

Parameters:

None

Description:

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

Return:

None

8.2.3.5 I2C_ClearINTReq

Clear I2C interrupt request in I2C bus mode.

Prototype:

void

I2C_ClearINTReq(void)

Parameters:

None

Description:

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

Return:

None

8.2.3.6 I2C_GenerateStart

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

Prototype:

void

I2C_GenerateStart(void)

Parameters:

None

Description:

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

Return:

None

8.2.3.7 I2C_GenerateStop

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

Prototype:

void

I2C_GenerateStop(void)

Parameters:

None

Description:

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

Return:

None

8.2.3.8 I2C_GetState

Get the I2C channel state in I2C bus mode.

Prototype:

I2C_State

I2C_GetState(void)

Parameters:

None

Description:

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

Return:

The state value of the I2C channel in I2C bus.

8.2.3.9 I2C_SetSendData

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

Prototype:

void

I2C_SetSendData(uint32_t ***Data***)

Parameters:

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 I2C channel. It is appropriate to call the function after the transmission of the start condition, which can be done by **I2C_Generatestart()**, or the reception of an ACK (usually causes an I2C interrupt), to send further data required by receiver.

Return:

None

8.2.3.10 I2C_GetReceiveData

Get data received from the I2C channel.

Prototype:

uint32_t

I2C_GetReceiveData(void)

Parameters:

None

Description:

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

Return:

Data which has been received

8.2.3.11 I2C_SetFreeDataMode

Set I2C channel working in I2C free data mode.

Prototype:

void

I2C_SetFreeDataMode(FunctionalState ***NewState***)

Parameters:

NewState specifies the state of the I2C when system is idle mode, which can be

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

Description:

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

Return:

None

8.2.3.12 I2C_GetSlaveAddrMatchState

Get slave address match detection state.

Prototype:

FunctionalState

I2C_ GetSlaveAddrMatchState(void)

Parameters:

None

Description:

Get slave address match detection state.

Return:

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

8.2.3.13 I2C_SetPrescalerClock

Set prescaler clock of the I2C channel.

Prototype:

void

I2C_SetPrescalerClock(uint32_t ***PrescalerClock***)

Parameters:

PrescalerClock is the prescaler clock value.

This parameter can be one of the following values:

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

Description:

This function will set prescaler clock of the I2C channel,

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

Return:

None

8.2.3.14 I2C_SetINTReq

Enable or disable interrupt request of the I2C channel.

Prototype:

void

I2C _SetINTReq(FunctionalState ***NewState***)

Parameters:

NewState: Specify I2C interrupt setting

This parameter can be one of the following values:

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

Description:

This function will enable or disable I2C interrupt request.

Return:

None

8.2.3.15 I2C_GetINTStatus

Get interrupt generation state.

Prototype:

FunctionalState

I2C_GetINTStatus(void)

Parameters:

None

Description:

This function will get the state of I2C interrupt generation.

Return:

The state of interrupt generation
ENABLE: I2C interrupt has been generated
DISABLE: I2C has not interrupt

8.2.3.16 I2C_ClearINTOutput

Clear the I2C interrupt output.

Prototype:

void

I2C_ClearINTOutput(void)

Parameters:

None

Description:

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

Return:

None

8.2.4 Data Structure Description

8.2.4.1 I2C_InitTypeDef

Data Fields:

uint32_t

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

uint32_t

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

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

uint32_t

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

- **I2C_SCK_CLK_DIV_20**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 20;
- **I2C_SCK_CLK_DIV_24**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 24;
- **I2C_SCK_CLK_DIV_32**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 32;
- **I2C_SCK_CLK_DIV_48**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 48;
- **I2C_SCK_CLK_DIV_80**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 80;
- **I2C_SCK_CLK_DIV_144**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 144;
- **I2C_SCK_CLK_DIV_272**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 272;
- **I2C_SCK_CLK_DIV_528**, which means that the frequency of the serial clock for I2C transfer is quotient of fprsck divided by 528;

uint32_t

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

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

- **I2C_PRESCALER_DIV_16**, which means that the frequency of the prescaler clock is quotient of fsys divided by 16
- **I2C_PRESCALER_DIV_17**, which means that the frequency of the prescaler clock is quotient of fsys divided by 17
- **I2C_PRESCALER_DIV_18**, which means that the frequency of the prescaler clock is quotient of fsys divided by 18
- **I2C_PRESCALER_DIV_19**, which means that the frequency of the prescaler clock is quotient of fsys divided by 19
- **I2C_PRESCALER_DIV_20**, which means that the frequency of the prescaler clock is quotient of fsys divided by 20
- **I2C_PRESCALER_DIV_21**, which means that the frequency of the prescaler clock is quotient of fsys divided by 21
- **I2C_PRESCALER_DIV_22**, which means that the frequency of the prescaler clock is quotient of fsys divided by 22
- **I2C_PRESCALER_DIV_23**, which means that the frequency of the prescaler clock is quotient of fsys divided by 23
- **I2C_PRESCALER_DIV_24**, which means that the frequency of the prescaler clock is quotient of fsys divided by 24
- **I2C_PRESCALER_DIV_25**, which means that the frequency of the prescaler clock is quotient of fsys divided by 25
- **I2C_PRESCALER_DIV_26**, which means that the frequency of the prescaler clock is quotient of fsys divided by 26
- **I2C_PRESCALER_DIV_27**, which means that the frequency of the prescaler clock is quotient of fsys divided by 27
- **I2C_PRESCALER_DIV_28**, which means that the frequency of the prescaler clock is quotient of fsys divided by 28
- **I2C_PRESCALER_DIV_29**, which means that the frequency of the prescaler clock is quotient of fsys divided by 29
- **I2C_PRESCALER_DIV_30**, which means that the frequency of the prescaler clock is quotient of fsys divided by 30
- **I2C_PRESCALER_DIV_31**, which means that the frequency of the prescaler clock is quotient of fsys divided by 31
- **I2C_PRESCALER_DIV_32**, which means that the frequency of the prescaler clock is quotient of fsys divided by 32

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

FunctionalState

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

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

8.2.4.2 I2C_State

Data Fields:

uint32_t

All specifies state data in I2C mode

Bit Fields:

uint32_t

LastRxBit specifies last received bit monitor.

uint32_t

GeneralCall specifies general call detected monitor.

uint32_t

SlaveAddrMatch specifies slave address match monitor.

uint32_t

ArbitrationLost specifies arbitration last detected monitor.

uint32_t

INTReq specifies Interrupt request monitor.

uint32_t

BusState specifies bus busy flag.

uint32_t

TRx specifies transfer or Receive selection monitor.

uint32_t

MasterSlave specifies master or slave selection monitor.

9. LVD

9.1 Overview

TMPM037 has Low voltage detection circuit (LVD). The voltage detection circuit generates a reset signal or an interrupt signal by detecting a decreasing/increasing voltage.

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

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm037_lvd.c, with /Libraries/TX00_Periph_Driver/inc/tmpm037_lvd.h containing the macros, data types, structures and API definitions for use by applications.

9.2 API Functions

9.2.1 Function List

- void LVD_EnableVD1(void)
- void LVD_DisableVD1(void)
- void LVD_SetVD1Level(uint32_t ***VDLevel***)
- LVD_VDStatus LVD_GetVD1Status(void)
- void LVD_SetVD1ResetOutput(FunctionalState ***NewState***)
- void LVD_SetVD1INTOutput(FunctionalState ***NewState***)
- uint32_t LVD_GetINTCondition(void)

9.2.2 Detailed Description

Functions listed above can be divided into two parts:

1) Configure LVD are handled by LVD_EnableVD1(), LVD_DisableVD1(), LVD_SetVD1Level();LVD_SetVD1ResetOutput(), LVD_SetVD1INTOutput(),
2) Get the power supply voltage detection status and interrupt generation condition info by LVD_GetVD1Status(), LVD_GetINTCondition().

9.2.3 Function Documentation

9.2.3.1 LVD_EnableVD1

Enable the operation of voltage detection 1.

Prototype:

void

LVD_EnableVD1(void)

Parameters:

None.

Description:

This function will enable the voltage detection 1 operation.

Return:

None.

9.2.3.2 LVD_DisableVD1

Disable the operation of voltage detection 1.

Prototype:

void

LVD_DisableVD1(void)

Parameters:

None.

Description:

This function will disable the voltage detection 1 operation.

Return:

None.

9.2.3.3 LVD_SetVD1Level

Select the level for detection voltage 1.

Prototype:

void

LVD_SetVD1Level(uint32_t ***VDLevel***)

Parameters:

VDLevel is the level of voltage detection 1.

This parameter can be one of the following values:

- **LVD_VDLVL1_250:** Voltage detection level is from 2.50 ± 0.2V.
- **LVD_VDLVL1_260: Voltage detection level is from 2.60 ± 0.2V.**
- **LVD_VDLVL1_270:** Voltage detection level is from 2.70 ± 0.2V.
- **LVD_VDLVL1_280:** Voltage detection level is from 2.80 ± 0.2V.
- **LVD_VDLVL1_290:** Voltage detection level is from 2.90 ± 0.2V.

Description:

This function will set the level of voltage detection 1.

Return:

None.

9.2.3.4 LVD_GetVD1Status

Get voltage detection 1 status.

Prototype:

LVD_VDStatus

LVD_GetVD1Status(void)

Parameters:

None.

Description:

This function will get voltage detection 1 status.

Return:

LVD_VDStatus: The voltage detection 1 status, which can be one of:

LVD_VD_UPPER: Power-supply voltage is the same as detection voltage or higher.

LVD_VD_LOWER: Power-supply voltage is the same as detection voltage or lower.

9.2.3.5 LVD_SetVD1ResetOutput

Enable or disable LVD reset output of voltage detection 1.

Prototype:

void

LVD_SetVD1ResetOutput(FunctionalState ***NewState***)

Parameters:

NewState: new state of LVD reset output.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function enables or disables LVD reset output of voltage detection 1.

Return:

None.

9.2.3.6 LVD_SetVD1INTOutput

Enable or disable LVD interrupt output of voltage detection 1.

Prototype:

void

LVD_SetVD1INTOutput(FunctionalState ***NewState***)

Parameters:

NewState: new state of LVD interrupt output.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function enables or disables LVD interrupt output of voltage detection 1.

Return:

None.

9.2.3.7 LVD_GetINTCondition

Get voltage detection interrupt generation condition.

Prototype:

uint32_t

LVD_GetINTCondition(void)

Parameters:

None.

Description:

This function will get voltage detection interrupt generation condition.

Return:

The voltage detection interrupt generation condition, which can be

LVD_INTSEL_LOWER: Only lower than the setting voltage when voltage decreasing.

LVD_INTSEL_LOWER_UPPER: Both lower and upper than the setting voltage when voltage decreasing.

9.2.4 Data Structure Description

None

10. TMR16A

10.1 Overview

TOSHIBA TMPM037 has 2 channels TMR16A that contains the following functions:

- Match interrupt
- Square waveform output
- Read capture.

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm037_tmr16a.c, with /Libraries/TX00_Periph_Driver/inc/tmpm037_tmr16a.h containing the macros, data types, structures and API definitions for use by applications.

10.2 API Functions

10.2.1 Function List

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

10.2.2 Detailed Description

Functions listed above can be divided into three parts:

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

10.2.3 Function Documentation

***Note**: in all of the following APIs, unless otherwise specified, the parameter:

"TSB_T16A_TypeDef * ***T16Ax***" can be one of the following values:

TSB_T16A0, TSB_T16A1.

10.2.3.1 TMR16A_SetIdleMode

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

Prototype:

void

TMR16A _SetIdleMode(TSB_ T16A _TypeDef* ***T16Ax***,

FunctionalState ***NewState***)

Parameters:

T16Ax is the specified TMR16A channel.

NewState specifies the state of the TMR16A when system is idle mode, which can be

- **ENABLE:** enables the TMR16A channel,
- **DISABLE:** disables the TMR16A channel.

Description:

The specified TMR16A channel can still be running if ***NewState*** is **ENABLE** even if system enters idle mode. **DISABLE** can stop the running TMR16A if system enters idle mode.

Return:

None

10.2.3.2 TMR16A_SetClkInCoreHalt

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

Prototype:

void

TMR16A_SetClkInCoreHalt (TSB_T16A_TypeDef* ***T16Ax***,

uint8_t ***ClkState***)

Parameters:

T16Ax is the specified TMR16A channel.

ClkState specifies timer state in HALT mode, which can be

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

Description:

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

Return:

None

10.2.3.3 TMR16A_SetRunState

Start or stop counter of the specified T16A channel.

Prototype:

void

TMR16A_SetRunState(TSB_T16A_TypeDef* ***T16Ax***,

uint32_t ***Cmd***)

Parameters:

T16Ax is the specified TMR16A channel.

Cmd sets the state of up-counter, which can be:

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

Description:

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

Return:

None

10.2.3.4 TMR16A_SetSrcClk

Specifies a source clock.

Prototype:

void

TMR16A_SetSrcClk(TSB_T16A_TypeDef* ***T16Ax***,

uint32_t ***SrcClk***)

Parameters:

T16Ax is the specified TMR16A channel.

SrcClk specifies the state of the TMR16A source clock, which can be

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

Description:

This function can select TMR16A channel's source clock.

Return:

None

10.2.3.5 TMR16A_SetFlipFlop

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

Prototype:

void

TMR16A_SetFlipFlop(TSB_T16A_TypeDef* ***T16Ax***,

TMR16A_FFOutputTypeDef* ***FFStruct***)

Parameters:

T16Ax is the specified TMR16A channel.

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

Description:

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

Return:

None

10.2.3.6 TMR16A_ChangeCycle

Change the value of cycle for the specified channel.

Prototype:

void

TMR16A_ChangeCycle(TSB_T16A_TypeDef* ***T16Ax***,

uint32_t ***Cycle***)

Parameters:

T16Ax is the specified TMR16A channel.

Cycle specifies the value of cycle, max is 0xFFFF.

Description:

This function will specify the absolute value of cycle for the specified TMR16A.

The actual interval of cycle depends on the configuration of CG and the value of ***ClkDiv***

Return:

None

10.2.3.7 TMR16A_GetCaptureValue

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

Prototype:

uint16_t

TMR16A_GetCaptureValue(TSB_T16A_TypeDef* ***T16Ax***)

Parameters:

T16Ax is the specified TMR16A channel.

Description:

This function will return the value of capture register of the specified TMR16A channel.

Return:

The captured value.

10.2.1 Data Structure Description

10.2.1.1 TMR16A_FFOutputTypeDef

Data Fields:

uint32_t

TMR16AFlipflopCtrl selects the level of flip-flop output which can be

- **TMR16A_FLIPFLOP_INVERT:** setting output reversed by using software.
- **TMR16A_FLIPFLOP_SET:** setting output to be high level.
- **TMR16A_FLIPFLOP_CLEAR:** setting output to be low level.

uint32_t

TMR16AFlipflopReverseTrg specifies the reverse trigger of the flip-flop output, which can be set as:

- **TMR16A_DISALBE_FLIPFLOP**, which disables the flip-flop output reverse trigger.
- **TMR16A_FLIPFLOP_MATCH_CYCLE**, which means that the reversing flip-flop output will be triggered when the up-counter matches the cycle.

11. TMRB

11.1 Overview

TOSHIBA TMPM037 contains 8 channels of TMRB (TMRB0 through TMRB7). Each channel consists of a 16-bit up-counter, two 16-bit timer registers (Double-buffered), two 16-bit capture registers, two comparators, a capture input control, a timer flip-flop and its associated control circuit. Each channel can operate in the following modes:

- Interval timer mode
- Event counter mode
- Programmable pulse generation (PPG) mode
- Programmable pulse generation (PPG) external trigger mode

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

- Frequency measurement
- Pulse width measurement

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

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm037_tmrb.c, with /Libraries/TX00_Periph_Driver/inc/tmpm037_tmrb.h containing the macros, data types, structures and API definitions for use by applications.

11.2 API Functions

11.2.1 Function List

- void TMRB_Enable(TSB_TB_TypeDef * ***TBx***);
- void TMRB_Disable(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetRunState(TSB_TB_TypeDef * ***TBx***, uint32_t ***Cmd***);
- void TMRB_Init(TSB_TB_TypeDef * ***TBx,*** TMRB_InitTypeDef * ***InitStruct***);
- void TMRB_SetCaptureTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***CaptureTiming***);
- void TMRB_SetFlipFlop(TSB_TB_TypeDef * ***TBx***,
  TMRB_FFOutputTypeDef * ***FFStruct***);
- TMRB_INTFactor TMRB_GetINTFactor(TSB_TB_TypeDef * ***TBx***);

- TMRB_INTMask TMRB_GetINTMask(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetINTMask(TSB_TB_TypeDef * ***TBx***, uint32_t ***INTMask***);
- void TMRB_ChangeLeadingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***LeadingTiming***);
- void TMRB_ChangeTrailingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***TrailingTiming***);
- uint16_t TMRB_GetRegisterValue(TSB_TB_TypeDef * ***TBx***, uint8_t ***Reg***);
- uint16_t TMRB_GetUpCntValue(TSB_TB_TypeDef * ***TBx***);
- uint16_t TMRB_GetCaptureValue(TSB_TB_TypeDef * ***TBx***, uint8_t ***CapReg***);
- void TMRB_ExecuteSWCapture(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetIdleMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);
- void TMRB_SetSyncMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);
- void TMRB_SetDoubleBuf(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);
- void TMRB_SetExtStartTrg(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***TrgMode***);
- void TMRB_SetClkInCoreHalt(TSB_TB_TypeDef * ***TBx***, uint8_t ***ClkState***);
- void TMRB_SetDMAReq(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***DMAReq***);

11.2.2 Detailed Description

Functions listed above can be divided into four parts:

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

11.2.3 Function Documentation

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

TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3,

TSB_TB4, TSB_TB5, TSB_TB6, TSB_TB7.

11.2.3.1 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

11.2.3.2 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

11.2.3.3 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

11.2.3.4 TMRB_Init

Initialize the specified TMRB channel.

Prototype:

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

Parameters:

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

Description:

This function will initialize and configure the count mode, clock division, up-counter setting, trailingtiming and leadingtiming duration for the specified TMRB channel selected by ***TBx***.

Return:

None

11.2.3.5 TMRB_SetCaptureTiming

Configure the capture timing.

Prototype:

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

Parameters:

TBx is the specified TMRB channel.

TSB_TB0, TSB_TB1, TSB_TB3, 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_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_FF_RISING_FALLING**: Takes count values into capture register 0 (TBxCP0) upon rising of TBxFF0 pin input and takes count values into capture register 1 (TBxCP1) upon falling of TBxFF0 pin input.

Description:

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_FF_RISING_FALLING**, then at the time of the rising edge of port TBxFF0 pin, the value in up-counter will be captured and saved into capture register0 (TBxCP0) of the TMRB channel. And the value in up-counter will be captured and saved into capture register1 (TBxCP1) upon falling of TBxFF0 pin input.

The flip-flop output of TMRB6, TMRB7 can be used as the capture trigger of other channels.

TMRB0~2: TB6OUT

TMRB3~5: TB7OUT

Return:

None

11.2.3.6 TMRB_SetFlipFlop

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

Prototype:

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

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

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

Return:
None

11.2.3.7 TMRB_GetINTFactor

Indicate what causes the interrupt.

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

Parameters:
TBx is the specified TMRB channel.

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

Return:
TMRB Interrupt factor. Each bit has the following meaning:
MatchLeadingTiming(Bit0): a match with the leadingtiming value is detected

MatchTrailingTiming(Bit1): a match with the trailingtiming value is detected
OverFlow(Bit2): an up-counter is overflow

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

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

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

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

11.2.3.8 TMRB_GetINTMask

Indicate what causes the interrupt be masked.

Prototype:
TMRB_INTMask
TMRB_GetINTMask (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 be masked. Bit of **MatchLeadingTimingMask** indicates if the up-counter matches with leadingtiming value interrupt is masked; bit of **MatchTrailingTimingMask** Indicates if the up-counter matches with trailingtiming value interrupt is masked, and bit of **OverflowMask** indicates if overflow had occurred interrupt is masked.

Return:

TMRB Interrupt factor be masked. Each bit has the following meaning:
MatchLeadingTimingMask(Bit0): a match with the leadingtiming value interrupt is masked. **MatchTrailingTimingMask**(Bit1): a match with the trailingtiming value interrupt is masked.
OverFlowMask(Bit2): an up-counter is overflow interrupt is masked.

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

```
TMRB_INTMask factor = TMRB_GetINTMask(TSB_TB0);
if (factor.Bit.MatchLeadingTimingMask) {
    // Do A
}

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

if (factor.Bit.OverFlowMask) {
    // Do C
}
```

11.2.3.9 TMRB_SetINTMask

Mask the specified TMRB interrupt.

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

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

- **TMRB_MASK_MATCH_TRAILINGTIMING_INT**: Mask the interrupt the factor of which is that the value in up-counter and trailingtiming are match.
- **TMRB_MASK_MATCH_LEADINGTIMING_INT**: Mask the interrupt the factor of which is that the value in up-counter and leadingtiming are match.
- **TMRB_MASK_OVERFLOW_INT**: Mask the interrupt the factor of which is the occurrence of overflow.

- **TMRB_NO_INT_MASK**: Unmask the interrupt.

Description:

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

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

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

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

Return:

None

11.2.3.10 TMRB_ChangeLeadingTiming

Change the value of leadingtiming for the specified channel.

Prototype:

void

TMRB_ChangeLeadingTiming(TSB_TB_TypeDef* ***TBx***,

uint32_t ***LeadingTiming***)

Parameters:

TBx is the specified TMRB channel.

LeadingTiming specifies the value of leadingtiming, max is 0xFFFF.

Description:

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

Return:

None

***Note:**

LeadingTiming cannot exceed ***TrailingTiming***.

11.2.3.11 TMRB_ChangeTrailingTiming

Change the value of trailingtiming for the specified channel.

Prototype:

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

Parameters:

TBx is the specified TMRB channel.
TrailingTiming specifies the value of trailingtiming, max is 0xFFFF.

Description:

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

Return:

None

***Note:**

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

11.2.3.12 TMRB_GetRegisterValue

Get register value of the specified TMRB channel.

Prototype:

uint16_t
TMRB_GetRegisterValue(TSB_TB_TypeDef* ***TBx,***
uint8_t ***Reg***)

Parameters:

TBx is the specified TMRB channel.

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

- **TMRB_Reg_0:** specifying register0.
- **TMRB_Reg_1:** specifying register1.

Description:

This function will return the value in register of the specified TMRB channel.

Return:

The value of register

11.2.3.13 TMRB_GetUpCntValue

Get up-counter value of the specified TMRB channel.

Prototype:

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

Parameters:

TBx is the specified TMRB channel.

Description:

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

Return:

The value of up-counter

11.2.3.14 TMRB_GetCaptureValue

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

Prototype:

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

Parameters:

TBx is the specified TMRB channel.

TSB_TB0, TSB_TB1, TSB_TB3, TSB_TB7.

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

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

Description:

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

Return:

The captured value

11.2.3.15 TMRB_ExecuteSWCapture

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

Prototype:

void

TMRB_ExecuteSWCapture(TSB_TB_TypeDef* ***TBx***)

Parameters:

TBx is the specified TMRB channel.

TSB_TB0, TSB_TB1, TSB_TB3, TSB_TB7.

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

11.2.3.16 TMRB_SetIdleMode

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

Prototype:

void

TMRB_SetIdleMode(TSB_TB_TypeDef* ***TBx***,

FunctionalState ***NewState***)

Parameters:

TBx is the specified TMRB channel.

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

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

Description:

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

Return:

None

11.2.3.17 TMRB_SetSyncMode

Enable or disable the synchronous mode of specified TMRB channel.

Prototype:

void

TMRB_SetSyncMode(TSB_TB_TypeDef* ***TBx***,

FunctionalState ***NewState***)

Parameters:

TBx is the specified TMRB channel, which can be

TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB5, TSB_TB6, TSB_TB7.

NewState specifies the state of the synchronous mode of the TMRB, which can be

- **ENABLE:** enables the synchronous mode,
- **DISABLE:** disables the synchronous mode.

Description:

If the synchronous mode is enabled for TMRB1 through TMRB3, their start timing is synchronized with TMRB0. If the synchronous mode is enabled for TMRB5 through TMRB7, their start timing is synchronized with TMRB4.

Return:

None

***Note:**

TMRB1 through TMRB3, TMRB5 through TMRB7 must start counting by calling **TMRB_SetRunState()** before TMRB0, TMRB4 start counting, so that start timing can be synchronized.

11.2.3.18 TMRB_SetDoubleBuf

Enable or disable double buffering for the specified TMRB channel.

Prototype:

void

TMRB_SetDoubleBuf(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***)

Parameters:

TBx is the specified TMRB channel.

NewState specifies the state of double buffering of the TMRB, which can be

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

Description:

This function will enable or disable double buffering for the specified TMRB channel.

Return:

None

11.2.3.19 TMRB_SetExtStartTrg

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

Prototype:

void

TMRB_SetExtStartTrg (TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***,
uint8_t ***TrgMode***)

Parameters:

TBx is the specified TMRB channel.

NewState specifies the state external trigger, which can be

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

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

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

Description:

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

Return:

None

11.2.3.20 TMRB_SetClkInCoreHalt

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

Prototype:

```
void
TMRB_SetClkInCoreHalt (TSB_TB_TypeDef* TBx,
                       uint8_t ClkState)
```

Parameters:

TBx is the specified TMRB channel.

ClkState specifies timer state in HALT mode, which can be

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

Description:

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

Return:

None

11.2.3.21 TMRB_SetDMAReq

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

Prototype:

void

TMRB_SetExtStartTrg (TSB_TB_TypeDef* ***TBx***,

FunctionalState ***NewState***,

uint8_t ***DMAReq***)

Parameters:

TBx is the specified TMRB channel.

TSB_TB0, TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB4,

TSB_TB5, TSB_TB6, TSB_TB7.

NewState enable or disable the DMA request,which can be

- **ENABLE:** enable the DMA request,
- **DISABLE:** disable the DMA request.

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

- **TMRB_DMA_REQ_CMP_MATCH:** Select DMA request: compare match.
- **TMRB_DMA_REQ_CAPTURE_1:** Select DMA request: input capture1.
- **TMRB_DMA_REQ_CAPTURE_0:** Select DMA request: input capture0.

Description:

This function will enable or disable the selected DMA request for the specified TMRB channel.

Return:

None

***Note:**

When mask configuration by TBxIM register is valid, DMA request does not issue even if it is enabled.

11.2.4 Data Structure Description

11.2.4.1 TMRB_InitTypeDef

Data Fields:

uint32_t

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

uint32_t

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

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

uint32_t

TrailingTiming specifies the trailingtiming value to be written into TBnRG1, max. is 0xFFFF.

uint32_t

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

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

uint32_t

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

11.2.4.2 TMRB_FFOutputTypeDef

Data Fields:

uint32_t

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

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

uint32_t

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

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

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

11.2.4.3 TMRB_INTFactor

Data Fields:

uint32_t

All: TMRB interrupt factor.

Bit

uint32_t

MatchLeadingTiming: 1 a match with the leadingtiming value is detected

uint32_t

MatchTrailingTiming: 1 a match with the trailingtiming value is detected

uint32_t

OverFlow: 1 an up-counter is overflow

uint32_t

Reserverd: 29 -

11.2.4.4 TMRB_INTMask

Data Fields:

uint32_t

All: TMRB interrupt factor be masked.

Bit

uint32_t

MatchLeadingTimingMask: 1 a match with the leadingtiming value interrupt is masked.

uint32_t

MatchTrailingTimingMask: 1 a match with the trailingtiming value interrupt is masked.

uint32_t

OverFlowMask: 1 an up-counter is overflow interrupt is masked.

uint32_t

Reserverd: 29 -

12. UART

12.1 Overview

TMPM037 has five serial I/O channels. Each channel can operate in both UART mode (asynchronous communication) and I/O Interface mode(synchronous communication), which can be 7-bit length, 8-bit length and 9-bit length.
In 9-bit UART mode, a wakeup function can be used when the master controller can start up slave controllers via the serial link (multi-controller system).

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

All driver APIs are contained in /Libraries/TX00_Periph_Driver/src/tmpm037_uart.c, with /Libraries/TX00_Periph_Driver/inc/tmpm037_uart.h containing the macros, data types, structures and API definitions for use by applications.

12.2 API Functions

12.2.1 Function List

- void UART_Enable(TSB_SC_TypeDef* ***UARTx***)
- void UART_Disable(TSB_SC_TypeDef* ***UARTx***)
- WorkState UART_GetBufState(TSB_SC_TypeDef* ***UARTx***, uint8_t ***Direction***)
- void UART_SWReset(TSB_SC_TypeDef* ***UARTx***)
- void UART_Init(TSB_SC_TypeDef* ***UARTx***, UART_InitTypeDef* ***InitStruct***)
- uint32_t UART_GetRxData(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetTxData(TSB_SC_TypeDef* ***UARTx***, uint32_t ***Data***)
- void UART_DefaultConfig(TSB_SC_TypeDef* ***UARTx***)
- UART_Err UART_GetErrState(TSB_SC_TypeDef* ***UARTx***)
- void UART_SetWakeUpFunc(TSB_SC_TypeDef* ***UARTx***,
  FunctionalState ***NewState***)
- void UART_SetIdleMode(TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_FIFOConfig(TSB_SC_TypeDef * ***UARTx***, FunctionalState NewState);
- void UART_SetFIFOTransferMode(TSB_SC_TypeDef * ***UARTx,***
  uint32_t ***TransferMode***);
- void UART_TRxAutoDisable(TSB_SC_TypeDef * ***UARTx***,
  UART_TRxAutoDisable ***TRxAutoDisable***);
- void UART_RxFIFOINTCtrl(TSB_SC_TypeDef * ***UARTx***, FunctionalState ***NewState***);
- void UART_TxFIFOINTCtrl(TSB_SC_TypeDef * ***UARTx***, FunctionalState ***NewState***);
- void UART_RxFIFOByteSel(TSB_SC_TypeDef * ***UARTx***, uint32_t ***BytesUsed***);
- void UART_RxFIFOFillLevel(TSB_SC_TypeDef * ***UARTx***, uint32_t ***RxFIFOLevel***);

- void UART_RxFIFOINTSel(TSB_SC_TypeDef * ***UARTx***, uint32_t ***RxINTCondition***);
- void UART_RxFIFOClear(TSB_SC_TypeDef * ***UARTx***);
- void UART_TxFIFOFillLevel(TSB_SC_TypeDef * ***UARTx***, uint32_t ***TxFIFOLevel***);
- void UART_TxFIFOINTSel(TSB_SC_TypeDef * ***UARTx***, uint32_t ***TxINTCondition***);
- void UART_TxFIFOClear(TSB_SC_TypeDef * ***UARTx***);
- void UART_TxBufferClear(TSB_SC_TypeDef * ***UARTx***);
- uint32_t UART_GetRxFIFOFillLevelStatus(TSB_SC_TypeDef * ***UARTx***);
- uint32_t UART_GetRxFIFOOverRunStatus(TSB_SC_TypeDef * ***UARTx***);
- uint32_t UART_GetTxFIFOFillLevelStatus(TSB_SC_TypeDef * ***UARTx***);
- uint32_t UART_GetTxFIFOUnderRunStatus(TSB_SC_TypeDef * ***UARTx***);
- void UART_SetRxDMAReq (TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetTxDMAReq (TSB_SC_TypeDef* ***UARTx***, FunctionalState ***NewState***)
- void UART_SetInputClock(TSB_SC_TypeDef * ***UARTx***, uint32_t ***clock***)
- void SIO_SetInputClock(TSB_SC_TypeDef * ***SIOx***, uint32_t ***Clock***)
- void SIO_Enable(TSB_SC_TypeDef* ***SIOx***)
- void SIO_Disable(TSB_SC_TypeDef* ***SIOx***)
- void SIO_Init(TSB_SC_TypeDef* ***SIOx***,
  uint32_t ***IOClkSel***,
  UART_InitTypeDef* ***InitStruct***)
- uint8_t SIO_GetRxData(TSB_SC_TypeDef* ***SIOx***)
- void SIO_SetTxData(TSB_SC_TypeDef* ***SIOx***, uint8_t ***Data***)

12.2.2 Detailed Description

Functions listed above can be divided into four parts:

1) Initialize and configure the common functions of each UART channel are handled by UART_Enable(), UART_Disable(),UART_SetInputClock(),UART_Init() and UART_DefaultConfig(),SIO_Enable(), SIO_Disable(),SIO_SetInputClock(),SIO_Init().
2) Transfer control and error check of each UART channel are handled by UART_GetBufState(), UART_GetRxData(), UART_SetTxData() and UART_GetErrState(),SIO_GetRxData(), SIO_SetTxData().
3) UART_SetRxDMAReq , UART_SetTxDMAReq ,UART_SWReset(), UART_SetWakeUpFunc() and UART_SetIdleMode() handle other specified functions.
4) FIFO operation functions are UART_FIFOConfig(),UART_SetFIFOTransferMode(), UART_TrxAutoDisable(),UART_RxFIFOINTCtrl(),UART_TxFIFOINTCtrl(),

   UART_RxFIFOByteSel(),UART_RxFIFOFillLevel(),UART_RxFIFOINTSel().

   UART_RxFIFOClear(),UART_TxFIFOFillLevel(),UART_TxFIFOINTSel().

   UART_TxFIFOClear(),UART_TxBufferClear (),UART_GetRxFIFOFillLevelStatus(),

   UART_GetRxFIFOOverRunStatus(),UART_GetTxFIFOFillLevelStatus(),and

   UART_GetTxFIFOUnderRunStatus(),

12.2.3 Function Documentation

Note**: in all of the following APIs, parameter "TSB_SC_TypeDef* ***UARTx" can be one of the following values:

UART0, **UART1**, **UART2**, **UART3**, **UART4**.

parameter "TSB_SC_TypeDef* ***SIOx***" can be one of the following values:

SIO0, **SIO1**, **SIO2**, **SIO3**, **SIO4**.

12.2.3.1 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

12.2.3.2 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

12.2.3.3 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.

12.2.3.4 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

12.2.3.5 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, and 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, and transfer mode and flow control for the specified UART channel selected by ***UARTx***.

Return:

None

12.2.3.6 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

12.2.3.7 UART_SetTxData

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

Prototype:

void

UART_SetTxData(TSB_SC_TypeDef* ***UARTx***,

uint32_t ***Data***)

Parameters:

UARTx is the specified UART channel.

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

Description:

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

Return:

None

12.2.3.8 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

12.2.3.9 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.

12.2.3.10 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

12.2.3.11 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

12.2.3.12 UART_FIFOConfig

Enable or disable FIFO of specified UART channel.

Prototype:

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

Parameters:

UARTx is the specified UART channel.
NewState is the new state of the UART FIFO.
This parameter can be one of the following values:
ENABLE or DISABLE

Description:

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

Return:

None

12.2.3.13 UART_SetFIFOTransferMode

Transfer mode setting.

Prototype:

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

Parameters:

UARTx is the specified UART channel.

TransferMode Transfer mode.

This parameter can be one of the following values:

UART_TRANSFER_PROHIBIT, UART_TRANSFER_HALFDPX_RX, UART_TRANSFER_HALFDPX_TX or UART_TRANSFER_FULLDPX.

Description:

Transfer mode setting.

Return:

None

12.2.3.14 UART_TRxAutoDisable

Controls automatic disabling of transmission and reception.

Prototype:

void

UART_TRxAutoDisable (TSB_SC_TypeDef* ***UARTx***,

UART_TRxAutoDisable ***TRxAutoDisable***);

Parameters:

UARTx is the specified UART channel.

TRxAutoDisable Disabling transmission and reception or not

This parameter can be one of the following values:

UART_RXTXCNT_NONE or UART_RXTXCNT_AUTODISABLE .

Description:

Controls automatic disabling of transmission and reception.

Return:

None

12.2.3.15 UART_RxFIFOINTCtrl

Enable or disable receive interrupt for receive FIFO.

Prototype:

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

Parameters:
UARTx is the specified UART channel.
NewState is new state of receive interrupt for receive FIFO.
This parameter can be one of the following values:
ENABLE or DISABLE

Description:
Enable or disable receive interrupt for receive FIFO.

Return:
None

12.2.3.16 UART_TxFIFOINTCtrl

Enable or disable transmit interrupt for transmit FIFO.

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

Parameters:
UARTx is the specified UART channel.
NewState is new state of transmit interrupt for transmit FIFO.
This parameter can be one of the following values:
ENABLE or DISABLE

Description:
Enable or disable transmit interrupt for transmit FIFO.

Return:
None

12.2.3.17 UART_RxFIFOByteSel

Bytes used in receive FIFO.

Prototype:

void

UART_RxFIFOByteSel (TSB_SC_TypeDef* ***UARTx***,

uint32_t ***BytesUsed***);

Parameters:

UARTx is the specified UART channel.

BytesUsed is bytes used in receive FIFO.

This parameter can be one of the following values:

UART_RXFIFO_MAX or UART_RXFIFO_RXFLEVEL

Description:

Bytes used in receive FIFO.

Return:

None

12.2.3.18 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, UART_RXFIFO4B_FLEVLE_3_1B.

Description:

Receive FIFO fill level to generate receive interrupts.

Return:

None

12.2.3.19 UART_RxFIFOINTSel

Select RX interrupt generation condition.

Prototype:

void

UART_RxFIFOINTSel (TSB_SC_TypeDef* ***UARTx***,

uint32_t ***RxINTCondition***);

Parameters:

UARTx is the specified UART channel.

RxINTCondition is RX interrupt generation condition.

This parameter can be one of the following values:

UART_RFIS_REACH_FLEVEL or UART_RFIS_REACH_EXCEED_FLEVEL

Description:

Select RX interrupt generation condition.

Return:

None

12.2.3.20 UART_RxFIFOClear

Receive FIFO clear.

Prototype:

void

UART_RxFIFOClear (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Receive FIFO clear.

Return:

None

12.2.3.21 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, UART_TXFIFO4B_FLEVLE_3_1B.

Description:

Transmit FIFO fill level to generate transmit interrupts.

Return:

None

12.2.3.22 UART_TxFIFOINTSel

Select TX interrupt generation condition.

Prototype:

void

UART_TxFIFOINTSel (TSB_SC_TypeDef* ***UARTx***,

uint32_t ***TxINTCondition***);

Parameters:

UARTx is the specified UART channel.

TxINTCondition is TX interrupt generation condition.

This parameter can be one of the following values:

UART_TFIS_REACH_FLEVEL or UART_TFIS_REACH_NOREACH_FLEVEL.

Description:

Select TX interrupt generation condition.

Return:

None

12.2.3.23 UART_TxFIFOClear

Transmit FIFO clear.

Prototype:

void

UART_TxFIFOClear (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Transmit FIFO clear.

Return:

None

12.2.3.24 UART_TxBufferClear

Transmit buffer clear.

Prototype:

void

UART_TxBufferClear (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Transmit buffer clear.

Return:

None

12.2.3.25 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.

12.2.3.26 UART_GetRxFIFOOverRunStatus

Receive FIFO overrun.

Prototype:

uint32_t

UART_ GetRxFIFOOverRunStatus (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Receive FIFO overrun.

Return:

UART_RXFIFO_OVERRUN: Flags for RX FIFO overrun.

12.2.3.27 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.

12.2.3.28 UART_GetTxFIFOUnderRunStatus

Transmit FIFO under run

Prototype:

uint32_t

UART_ GetTxFIFOUnderRunStatus (TSB_SC_TypeDef* ***UARTx***);

Parameters:

UARTx is the specified UART channel.

Description:

Transmit FIFO under run

Return:

UART_TXFIFO_UNDERRUN: Flags for TX FIFO under-run.

12.2.3.29 UART_SetRxDMAReq

Enable or disable the specified UART channel DMA Request By receive interrupt INTRX.

Prototype:

void
UART_SetRxDMAReq (TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

Parameters:

UARTx is the specified UART channel.

NewState is the new state of the UART channel DMA Request.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable the Rx DMA Reuqest of the specified UART channel selected by ***UARTx*** DMA Request when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:

None

12.2.3.30 UART_SetTxDMAReq

Enable or disable the specified UART channel DMA Request By receive interrupt INTTX.

Prototype:

void
UART_SetTxDMAReq (TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

Parameters:

UARTx is the specified UART channel.

NewState is the new state of the UART channel DMA Request.

This parameter can be one of the following values:

ENABLE or **DISABLE**

Description:

This function will enable the Tx DMA Request of the specified UART channel selected by ***UARTx*** DMA Request when ***NewState*** is **ENABLE**, and disable the channel when ***NewState*** is **DISABLE**.

Return:

None

12.2.3.31 UART_SetInputClock

Selects input clock for prescaler.

Prototype:

void

UART_SetInputClock (TSB_SC_TypeDef * UARTx,

uint32_t clock)

Parameters:

UARTx is the specified UART channel.

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

This parameter can be one of the following values:

0: PhiT0/2

1: PhiT0

Description:

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

Return:

None

12.2.3.32 SIO_SetInputClock

Selects input clock for prescaler.

Prototype:

void

SIO_SetInputClock (TSB_SC_TypeDef * SIOx,

uint32_t Clock)

Parameters:

SIOx is the specified SIO channel.

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

This parameter can be one of the following values:

SIO_CLOCK_T0_HALF: PhiT0/2

SIO_CLOCK_T0: PhiT0

Description:

This function will select the specified SIO channel by ***SIOx*** and specified the input clock for prescaler by ***clock***

Return:

None

12.2.3.33 SIO_Enable

Enable the specified SIO channel.

Prototype:

void

SIO_Enable(TSB_SC_TypeDef* ***SIOx***)

Parameters:

SIOx is the specified SIO channel.

Description:

This function will enable the specified SIO channel selected by ***SIOx***.

Return:

None

12.2.3.34 SIO_Disable

Disable the specified SIO channel.

Prototype:

void

SIO_Disable(TSB_SC_TypeDef* ***SIOx***)

Parameters:

SIOx is the specified SIO channel.

Description:

This function will disable the specified SIO channel selected by ***SIOx***.

Return:

None

12.2.3.35 SIO_Init

Initialize and configure the specified SIO channel.

Prototype:

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

Parameters:

SIOx is the specified SIO channel.
InitStruct is the structure containing basic SIO configuration. (Refer to “Data Structure Description” for details).

Description:

This function will initialize and configure the specified SIO channel selected by ***SIOx***.

Return:

None

12.2.3.36 SIO_GetRxData

Get data received from the specified SIO channel.

Prototype:

Uint8_t

SIO_GetRxData(TSB_SC_TypeDef* ***SIOx***)

Parameters:

SIOx is the specified SIO channel.

Description:

This function will get the data received from the specified SIO channel selected by ***SIOx***.

Return:

Data which has been received

12.2.3.37 SIO_SetTxData

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

Prototype:

void
SIO_SetTxData(TSB_SC_TypeDef* ***SIOx***,
Uint8_t ***Data***)

Parameters:

SIOx is the specified SIO channel.
Data is a frame to be sent.

Description:

This function will set the data to be sent from the specified SIO channel selected by ***SIOx***.

Return:

None

12.2.4 Data Structure Description

12.2.4.1 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.

12.2.4.2 SIO_InitTypeDef

Data Fields:

uint32_t

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

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

uint32_t

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

- **SIO_TIDLE_LOW** Set the status of TXDx pin keep a low level output.

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

uint32_t

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

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

uint32_t

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

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

uint32_t

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

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

uint32_t

TransferMode Setting transfer mode, which can be set as:

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

uint32_t

TransferDir Setting transfer mode, which can be set as:

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

uint32_t

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

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

uint32_t

DoubleBuffer Double Buffer mode, which can be set as:

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

uint32_t

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

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

uint32_t

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

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

13. WDT

13.1 Overview

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

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

This driver is contained in \Libraries\TX00_Periph_Driver\src\tmpm037_wdt.c, with \Libraries/TX00_Periph_Driver\inc\tmpm037_wdt.h containing the API definitions for use by applications.

13.2 API Functions

13.2.1 Function List

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

13.2.2 Detailed Description

Functions listed above can be divided into two parts:

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

13.2.3 Function Documentation

13.2.3.1 WDT_SetDetectTime

Set detection time for WDT.

Prototype:

void

WDT_SetDetectTime(uint32_t ***DetectTime***)

Parameters:

***DetectTime*:** Set the detection time

This parameter can be one of the following values:

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

Description:

This function will set detection time for WDT.

Return:

None

13.2.3.2 WDT_SetIdleMode

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

Prototype:

void

WDT_SetIdleMode(FunctionalState ***NewState***)

Parameters:

NewState: Run or stop WDT counter.

This parameter can be one of the following values:

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

Description:

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

***Note:**

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

Return:

None

13.2.3.3 WDT_SetOverflowOutput

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

Prototype:

void

WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)

Parameters:

OverflowOutput: Select function of WDT when counter overflow.

This parameter can be one of the following values:

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

Description:

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

Return:

None

13.2.3.4 WDT_Init

Initialize and configure WDT.

Prototype:

void

WDT_Init (WDT_InitTypeDef* ***InitStruct***)

Parameters:

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

Description:

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

Return:

None

13.2.3.5 WDT_Enable

Enable the WDT function.

Prototype:

void
WDT_Enable(void)

Parameters:

None

Description:

This function will enable WDT.

Return:

None

13.2.3.6 WDT_Disable

Disable the WDT function.

Prototype:

void
WDT_Disable(void)

Parameters:

None

Description:

This function will disable WDT.

Return:

None

13.2.3.7 WDT_WriteClearCode

Write the clear code.

Prototype:

void

WDT_WriteClearCode (void)

Parameters:

None

Description:

This function will clear the WDT counter.

Return:

None

13.2.4

13.2.5 Data Structure Description

13.2.5.1 WDT_InitTypeDef

Data Fields:

uint32_t

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

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

uint32_t

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

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