

# **TOSHIBA**

## **TX03 ペリフェラルドライバ ユーザガイド (TMPM365)**

第一版  
2017 年 9 月

**東芝デバイス&ストレージ株式会社**

## **本製品取り扱い上のお願い**

- ソフトウェア使用権許諾契約書の同意無しに使用しないで下さい。

**© 2017 Toshiba Electronic Devices & Storage Corporation**

## 目次

1.	はじめに .....	1
2.	TX03 ペリフェラルドライバの構成 .....	1
3.	ADC .....	2
3.1	概要 .....	2
3.2	API 関数 .....	2
3.2.1	関数一覧 .....	2
3.2.2	関数の種類 .....	3
3.2.3	関数仕様 .....	3
3.2.4	データ構造 .....	15
4.	CG .....	17
4.1	概要 .....	17
4.2	API 関数 .....	17
4.2.1	関数一覧 .....	17
4.2.2	関数の種類 .....	18
4.2.3	関数仕様 .....	18
4.2.4	データ構造 .....	33
5.	DMAC .....	35
5.1	概要 .....	35
5.2	API 関数 .....	35
5.2.1	関数一覧 .....	35
5.2.2	関数の種類 .....	36
5.2.3	関数仕様 .....	36
5.2.4	データ構造 .....	44
6.	FC .....	47
6.1	概要 .....	47
6.2	API 関数 .....	47
6.2.1	関数一覧 .....	47
6.2.2	関数の種類 .....	47
6.2.3	関数仕様 .....	47
6.2.4	データ構造 .....	51
7.	GPIO .....	52
7.1	概要 .....	52
7.2	API 関数 .....	52
7.2.1	関数一覧 .....	52
7.2.2	関数の種類 .....	52
7.2.3	関数仕様 .....	53
7.2.4	データ構造 .....	63
8.	SBI .....	65
8.1	概要 .....	65
8.2	API 関数 .....	65
8.2.1	関数一覧 .....	65
8.2.2	関数の種類 .....	65
8.2.3	関数仕様 .....	66
8.2.4	データ構造 .....	71
9.	TMRB .....	73

9.1	概要 .....	73
9.2	API 関数 .....	73
9.2.1	関数一覧 .....	73
9.2.2	関数の種類 .....	74
9.2.3	関数仕様 .....	74
9.2.4	データ構造 .....	84
<b>10.</b>	<b>SIO/UART .....</b>	<b>86</b>
10.1	概要 .....	86
10.2	API 関数 .....	86
10.2.1	関数一覧 .....	86
10.2.2	関数の種類 .....	87
10.2.3	関数仕様 .....	87
10.2.4	データ構造 .....	101
<b>11.</b>	<b>USB .....</b>	<b>104</b>
11.1	概要 .....	104
11.2	API 関数 .....	104
11.2.1	関数一覧 .....	104
11.2.2	関数の種類 .....	105
11.2.3	関数仕様 .....	105
11.2.4	データ構造 .....	115
<b>12.</b>	<b>WDT .....</b>	<b>126</b>
12.1	概要 .....	126
12.2	API 関数 .....	126
12.2.1	関数一覧 .....	126
12.2.2	関数の種類 .....	126
12.2.3	関数仕様 .....	126
12.2.4	データ構造 .....	129

## 1. はじめに

本ソフトウェアは、東芝製TX03シリーズマイコンTMPM365用ペリフェラルドライバセットです。

TX03ペリフェラルドライバでは、ユーザーアプリケーション内で各ペリフェラルを簡単に使用するためのマクロ、データ構造、関数および使用例を用意しています。

TMPM365 ペリフェラルドライバは以下の仕様にに基づいています。

➤ スタートアップルーチンといくつかの関数を除き、C 言語で記述されています。

## 2. TX03 ペリフェラルドライバの構成

### **/Libraries**

TX03 CMSIS ファイルと TMPM365 ペリフェラルドライバが格納されています。

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

このフォルダには TMPM365 CMSIS ファイルのデバイス・ペリフェラル・アクセス・レイヤーが格納されています。

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

TMPM365 ペリフェラルドライバの全てのソースコードが格納されています。

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

TMPM365 ペリフェラルドライバのヘッダファイルが格納されています。

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

TMPM365 ペリフェラルドライバのソースファイルが格納されています。

### **/Project**

TMPM365 ペリフェラルドライバのテンプレートプロジェクトと使用例が格納されています。

### **/Project/Template**

TMPM365 ペリフェラルドライバのテンプレートプロジェクトが格納されています。

### **/Project/Examples**

TMPM365 ペリフェラルドライバの使用例が格納されています。

### **/Utilities/TMPM365-EVAL**

TMPM365 評価ボードのハードウェアリソース用の設定ファイル、およびドライバファイル (例: led, key) が格納されています。

## 3. ADC

### 3.1 概要

本デバイスは、12ビット逐次変換方式アナログ/デジタルコンバータ(ADコンバータ)を内蔵しており、12チャンネルのアナログ入力を持っています。

12ビット A/D コンバータは、以下のような特徴があります。

- (1) 通常 AD 変換、最優先 AD 変換の起動
  - ソフトウェアによる起動
  - 外部トリガ入力(ADTRG)によるハードウェア起動
  - 16ビットタイマによる起動
- (2) 通常 AD 変換機能の動作モード
  - チャンネル固定シングル変換モード
  - チャンネルスキップシングル変換モード
  - チャンネル固定リピート変換モード
  - チャンネルスキップリピート変換モード
- (3) 最優先 AD 変換機能の動作モード
  - チャンネル固定シングル変換モード
- (4) 通常 AD 変換終了、最優先 AD 変換終了時、割り込み発生機能
- (5) 通常 AD 変換機能、最優先 AD 変換機能は以下のステータスフラグを持っています。  
AD 変換結果格納フラグ、オーバーランフラグ、AD 変換終了フラグ、AD 変換ビジーフラグ
- (6) AD 監視機能  
AD 変換結果とあらかじめ設定した値とを比較し、特定の条件で割り込みを発生
- (7) AD 変換クロックを 1/fc~1/16fc まで制御可能
- (8) AD 変換終了時、2 種類の DMA リクエストをサポート
- (9) スタンバイモードをサポート
- (10) 出力スイッチングモニタ機能

ADCドライバ API は、各モジュールの設定機能を持ち、チャンネル選択、モード設定、モニタ機能設定、割り込み設定、ステータスリード、AD 変換結果の取得などの機能を提供します。

全ドライバ API は、アプリで使用する API 定義を格納する以下のファイルで構成されています。

/Libraries/TX03\_Periph\_Driver/src/tmpm365\_adc.c  
/Libraries/TX03\_Periph\_Driver/inc/tmpm365\_adc.h

### 3.2 API 関数

#### 3.2.1 関数一覧

- ◆ void ADC\_SWReset(void)
- ◆ void ADC\_SetClk(uint32\_t **Sample\_HoldTime**, uint32\_t **Prescaler\_Output**)
- ◆ void ADC\_Start(void)
- ◆ void ADC\_SetScanMode(FunctionalState **NewState**)
- ◆ void ADC\_SetRepeatMode(FunctionalState **NewState**)
- ◆ void ADC\_SetINTMode(uint8\_t **INTMode**)
- ◆ void ADC\_SetInputChannel(uint8\_t **InputChannel**)
- ◆ void ADC\_SetScanChannel(uint8\_t **StartChannel**, uint8\_t **Range**)
- ◆ void ADC\_SetVrefCut(uint8\_t **VrefCtrl**)
- ◆ void ADC\_SetIdleMode(FunctionalState **NewState**)
- ◆ void ADC\_SetVref(FunctionalState **NewState**)

- ◆ void ADC\_SetInputChannelTop(uint8\_t **TopInputChannel**)
- ◆ void ADC\_StartTopConvert(void)
- ◆ void ADC\_SetMonitor(ADC\_CMPCR<sub>x</sub> **ADCMP<sub>x</sub>**, FunctionalState **NewState**)
- ◆ void ADC\_ConfigMonitor(ADC\_CMPCR<sub>x</sub> **ADCMP<sub>x</sub>**, ADC\_MonitorTypeDef\* **Monitor**)
- ◆ void ADC\_SetHWTrg(uint8\_t **HwSource**, FunctionalState **NewState**)
- ◆ void ADC\_SetHWTrgTop(uint8\_t **HwSource**, FunctionalState **NewState**)
- ◆ ADC\_State ADC\_GetConvertState(void)
- ◆ ADC\_Result ADC\_GetConvertResult (uint8\_t **ADREG<sub>x</sub>**)
- ◆ void ADC\_SetClkSupply(FunctionalState **NewState**)
- ◆ void ADC\_SetDMAReq(uint8\_t **DMAReq**, FunctionalState **NewState**)

### 3.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

- 1) AD 変換設定:  
ADC\_SetClk(), ADC\_SetScanMode(), ADC\_SetRepeatMode(), ADC\_SetINTMode(),  
ADC\_SetInputChannel(), ADC\_SetScanChannel(), ADC\_SetVref(),  
ADC\_SetInputChannelTop(), ADC\_SetMonitor(), ADC\_ConfigMonitor(),  
ADC\_SetHWTrg(), ADC\_SetHWTrgTop()
- 2) AD 変換の許可/禁止と開始:  
ADC\_Start(), ADC\_StartTopConvert()
- 3) AD 変換ステータス/結果の読み出し:  
ADC\_GetConvertState(), ADC\_GetConvertResult()
- 4) その他:  
ADC\_SWReset(), ADC\_SetVrefCut(), ADC\_SetIdleMode(), ADC\_SetClkSupply(),  
ADC\_SetDMAReq()

### 3.2.3 関数仕様

#### 3.2.3.1 ADC\_SWReset

ADC のソフトウェアリセット

関数のプロトタイプ宣言:

```
void  
ADC_SWReset(void)
```

引数:

なし

機能:

ADC をソフトウェアリセットします。

補足:

ADxCLK<ADCLK>を除くレジスタは、すべて初期化されます。  
ソフトウェアリセットを行う場合、初期化に 3μs の時間が必要となります。

戻り値:

なし

#### 3.2.3.2 ADC\_SetClk

AD 変換サンプルホールド時間とプリスケアラ出力(SCLK)の設定

## 関数のプロトタイプ宣言:

```
void
ADC_SetClk(uint32_t Sample_HoldTime,
           uint32_t Prescaler_Output)
```

## 引数:

**Sample\_HoldTime**: 以下から ADC サンプルホールド時間を選択します。

- **ADC\_CONVERSION\_CLK\_10**: 10x <ADCLK>
- **ADC\_CONVERSION\_CLK\_20**: 20x <ADCLK>
- **ADC\_CONVERSION\_CLK\_30**: 30x <ADCLK>
- **ADC\_CONVERSION\_CLK\_40**: 40x <ADCLK>
- **ADC\_CONVERSION\_CLK\_80**: 80x <ADCLK>

**Prescaler\_Output**: 以下から ADC プリスケール出力(ADCLK)を選択します。

- **ADC\_FC\_DIVIDE\_LEVEL\_1**:  $fc$
- **ADC\_FC\_DIVIDE\_LEVEL\_2**:  $fc / 2$
- **ADC\_FC\_DIVIDE\_LEVEL\_4**:  $fc / 4$
- **ADC\_FC\_DIVIDE\_LEVEL\_8**:  $fc / 8$

## 機能:

**Sample\_HoldTime** で ADC サンプルホールド時間を設定し、**Prescaler\_Output** でプリスケール出力を設定します。

## 補足:

AD変換中は、この関数を使わないでください。またAD変換状態を確認するための **ADC\_GetConvertState()** がBUSYでない場合、この関数をコールすることができます。

サンプルホールド時間と変換時間例は下記ようになります。

<b>Prescaler_Output</b>	<b>Sample_HoldTime</b>	<b>Conversion time</b>		
		$fc=32\text{MHz}$	$fc=40\text{MHz}$	$fc=54\text{MHz}$
ADC_FC_DIVIDE_LEVEL_1 ( $fc$ )	ADC_CONVERSION_CLK_10	1.25 $\mu\text{s}$	1.00 $\mu\text{s}$	-
	ADC_CONVERSION_CLK_20	1.56 $\mu\text{s}$	1.25 $\mu\text{s}$	-
	ADC_CONVERSION_CLK_30	1.88 $\mu\text{s}$	1.50 $\mu\text{s}$	-
	ADC_CONVERSION_CLK_40	2.19 $\mu\text{s}$	1.75 $\mu\text{s}$	-
	ADC_CONVERSION_CLK_80	3.44 $\mu\text{s}$	2.75 $\mu\text{s}$	-
ADC_FC_DIVIDE_LEVEL_2 ( $fc / 2$ )	ADC_CONVERSION_CLK_10	2.50 $\mu\text{s}$	2.00 $\mu\text{s}$	1.48 $\mu\text{s}$
	ADC_CONVERSION_CLK_20	3.13 $\mu\text{s}$	2.50 $\mu\text{s}$	1.85 $\mu\text{s}$
	ADC_CONVERSION_CLK_30	3.75 $\mu\text{s}$	3.00 $\mu\text{s}$	2.22 $\mu\text{s}$
	ADC_CONVERSION_CLK_40	4.38 $\mu\text{s}$	3.50 $\mu\text{s}$	2.59 $\mu\text{s}$
	ADC_CONVERSION_CLK_80	6.88 $\mu\text{s}$	5.50 $\mu\text{s}$	4.07 $\mu\text{s}$
ADC_FC_DIVIDE_LEVEL_4 ( $fc / 4$ )	ADC_CONVERSION_CLK_10	5.00 $\mu\text{s}$	4.00 $\mu\text{s}$	2.96 $\mu\text{s}$
	ADC_CONVERSION_CLK_20	6.25 $\mu\text{s}$	5.00 $\mu\text{s}$	3.70 $\mu\text{s}$
	ADC_CONVERSION_CLK_30	7.50 $\mu\text{s}$	6.00 $\mu\text{s}$	4.44 $\mu\text{s}$
	ADC_CONVERSION_CLK_40	8.75 $\mu\text{s}$	7.00 $\mu\text{s}$	5.19 $\mu\text{s}$
	ADC_CONVERSION_CLK_80	-	-	8.15 $\mu\text{s}$
ADC_FC_DIVIDE_LEVEL_8 ( $fc / 8$ )	ADC_CONVERSION_CLK_10	10.0 $\mu\text{s}$	8.00 $\mu\text{s}$	5.93 $\mu\text{s}$
	ADC_CONVERSION_CLK_20	-	10.0 $\mu\text{s}$	7.41 $\mu\text{s}$
	ADC_CONVERSION_CLK_30	-	-	8.89 $\mu\text{s}$
	ADC_CONVERSION_CLK_40	-	-	-
	ADC_CONVERSION_CLK_80	-	-	-

上記一覧で "-" で示される部分の設定は禁止されています。ADCLK は1 $\mu\text{s}$  から10 $\mu\text{s}$  間の値で設定してください。



戻り値:  
なし

### 3.2.3.3 ADC\_Start

AD 変換の開始

関数のプロトタイプ宣言:  
void  
ADC\_Start(void)

引数:  
なし

機能:  
AD 変換を開始します。

補足:  
この関数をコールする前に、以下のいずれかのモードを選択してください:  
    チャンネル固定シングル変換モード  
    チャンネルスキャンシングル変換モード  
    チャンネル固定リピート変換モード  
    チャンネルスキャンリピート変換モード  
詳細は、ADC\_SetScanMode(), ADC\_SetRepeatMode(),  
ADC\_SetInputChannel(), ADC\_SetScanChannel() を参照してください。

AD 変換をスタートさせる場合、ADC\_SetVref (ENABLE)をコールして Vref を有効にしてください。なお、Vref 有効後、3  $\mu$ s の安定時間が必要です。その後、ADC\_Start()をコールしてください。

戻り値:  
なし

### 3.2.3.4 ADC\_SetScanMode

スキャンモードの設定

関数のプロトタイプ宣言:  
void  
ADC\_SetScanMode(FunctionalState **NewState**)

引数:  
**NewState**: 以下から、スキャンモードを設定します。  
➤ **ENABLE**: チャンネルスキャン  
➤ **DISABLE**: チャンネル固定

機能:  
AD 変換スキャンモードを設定します。

戻り値:

なし

### 3.2.3.5 ADC\_SetRepeatMode

リピートモードの設定

関数のプロトタイプ宣言:

```
void  
ADC_SetRepeatMode(FunctionalState NewState)
```

引数:

**NewState**: 以下から、リピートモードを設定します。

- **ENABLE**: リピート変換
- **DISABLE**: シングル変換

機能:

リピートモードを設定します。

戻り値:

なし

### 3.2.3.6 ADC\_SetINTMode

チャンネル固定リピート変換モード時の割り込みタイミングの設定

関数のプロトタイプ宣言:

```
void  
ADC_SetINTMode(uint8_t INTMode)
```

引数:

**INTMode**: 以下から、割り込みタイミングを選択します。

- **ADC\_INT\_SINGLE**: 1 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_2**: 2 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_3**: 3 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_4**: 4 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_5**: 5 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_6**: 6 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_7**: 7 回毎、割り込み発生
- **ADC\_INT\_CONVERSION\_8**: 8 回毎、割り込み発生

機能:

チャンネル固定リピート変換モード時の割り込みタイミングを設定します。

補足:

この関数は、チャンネル固定リピート変換モード時のみ有効です。

以下は、チャンネル固定リピート変換モードの例です:

1. **ADC\_SetScanMode(DISABLE)**.
2. **ADC\_SetRepeatMode(ENABLE)**.

戻り値:

なし

## 3.2.3.7 ADC\_SetInputChannel

アナログ入力チャネルの選択

関数のプロトタイプ宣言:

```
void  
ADC_SetInputChannel(uint8_t InputChannel)
```

引数:

**InputChannel**: 以下から、いずれか 1 つのアナログ入力チャネルを使用します。

ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,  
ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07,  
ADC\_AN\_08, ADC\_AN\_09, ADC\_AN\_10, ADC\_AN\_11

機能:

アナログ入力チャネルを選択します。

補足:

通常変換入力の場合 1 チャネルのみ選択できます。

戻り値:

なし

## 3.2.3.8 ADC\_SetScanChannel

スキャンチャネルの設定

関数のプロトタイプ宣言:

```
void  
ADC_SetScanChannel (uint8_t StartChannel, uint8_t Range)
```

引数:

**StartChannel**: 以下から、チャネルスキャンの先頭チャネルを設定します。

ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,  
ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07,  
ADC\_AN\_08, ADC\_AN\_09, ADC\_AN\_10, ADC\_AN\_11

**Range**: チャネルスキャンの範囲を 1～12 のいずれか選択できます。

機能:

**StartChannel** にてチャネルスキャンの先頭チャネルの設定を、**Range** にてチャネルスキャンの範囲を設定します。

補足:

設定可能なチャネルスキャンの範囲を下表に示します。

<b>StartChannel</b>	<b>Range</b>
ADC_AN_00	1 ~ 12
ADC_AN_01	1 ~ 11
ADC_AN_02	1 ~ 10
ADC_AN_03	1 ~ 9
ADC_AN_04	1 ~ 8
ADC_AN_05	1 ~ 7
ADC_AN_06	1 ~ 6

ADC_AN_07	1 ~ 5
ADC_AN_08	1 ~ 4
ADC_AN_09	1 ~ 3
ADC_AN_10	1 ~ 2
ADC_AN_11	1

上記以外の場合、**ADC\_Start()**をコールしてもAD変換は行われません。

戻り値:

なし

### 3.2.3.9 ADC\_SetVrefCut

AVREFH-AVREFL間のリファレンス電流制御

関数のプロトタイプ宣言:

void

ADC\_SetVrefCut(uint8\_t **VrefCtrl**)

引数:

**VrefCtrl**: AVREFH-AVREFL間のリファレンス電流を制御します。

- **ADC\_APPLY\_VREF\_IN\_CONVERSION**: 変換中のみ通電
- **ADC\_APPLY\_VREF\_AT\_ANY\_TIME**: リセット時以外常時通電

機能:

AVREFH-AVREFL間のリファレンス電流を制御します。

戻り値:

なし

### 3.2.3.10 ADC\_SetIdleMode

IDLEモード時のADC動作制御

関数のプロトタイプ宣言:

void

ADC\_SetIdleMode(FunctionalState **NewState**)

引数:

**NewState**: 以下から、IDLEモード時のADC動作を選択します。

- **ENABLE**: 動作
- **DISABLE**: 停止

機能:

IDLEモード時のADC動作を制御します。

IDLEモードに移行する前にこの関数をコールする必要があります。

戻り値:

なし

## 3.2.3.11 ADC\_SetVref

Vref 回路の on/off 制御

**関数のプロトタイプ宣言:**

void  
ADC\_SetVref(FunctionalState **NewState**)

**引数:**

**NewState:** 以下から、Vref 回路の状態を選択します。

- **ENABLE:** ON
- **DISABLE:** OFF

**機能:**

Vref 回路の on/off を制御します。

**補足:**

低消費電力モードに移行する前に、ADC\_SetVref(DISABLE) をコールしてください。

**戻り値:**

なし

## 3.2.3.12 ADC\_SetInputChannelTop

最優先 AD 変換入力チャネルの設定

**関数のプロトタイプ宣言:**

void  
ADC\_SetInputChannelTop(uint8\_t **TopInputChannel**)

**引数:**

**TopInputChannel:** 以下から、最優先 AD 変換入力チャネルを選択します。

ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,  
ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07,  
ADC\_AN\_08, ADC\_AN\_09, ADC\_AN\_10, ADC\_AN\_11

**機能:**

最優先 AD 変換入力チャネルを設定します。

**補足:**

最優先 AD 変換入力を 1 チャネルのみ選択できます。

**戻り値:**

なし

## 3.2.3.13 ADC\_StartTopConvert

最優先 AD 変換の開始

**関数のプロトタイプ宣言:**

void  
ADC\_StartTopConvert(void

引数:

なし

機能:

最優先 AD 変換を開始します。

補足:

この関数をコールする前 **ADC\_SetInputChannelTop()**をコールしてください。

戻り値:

なし

### 3.2.3.14 ADC\_SetMonitor

AD 監視機能の許可/禁止

関数のプロトタイプ宣言:

```
void  
ADC_SetMonitor(ADC_CMPCRx ADCMPx,  
                FunctionalState NewState)
```

引数:

**ADCMP<sub>x</sub>**: 以下から、監視機能設定レジスタを選択します。

- **ADC\_CMPCR\_0**: ADCMPCR0
- **ADC\_CMPCR\_1**: ADCMPCR1

**NewState**: 以下から、監視機能を設定します。

- **ENABLE**: 許可(条件成立で AD 監視割り込みを発生します)
- **DISABLE**: 禁止(大小判定カウント数はクリア)

機能:

本デバイスは、2つの AD 監視機能を持ち、それぞれ設定レジスタで制御します。  
ADCMP<sub>x</sub> 設定で AD 監視レジスタを選択し、NewState で許可/禁止を設定します。

戻り値:

なし

### 3.2.3.15 ADC\_ConfigMonitor

AD 監視機能の設定

関数のプロトタイプ宣言:

```
void  
ADC_ConfigMonitor(ADC_CMPCRx ADCMPx,  
                  ADC_MonitorTypeDef * Monitor)
```

引数:

**ADCMP<sub>x</sub>**: 以下から、AD 変換レジスタを選択します。

- **ADC\_CMPCR\_0**: ADCMPCR0
- **ADC\_CMPCR\_1**: ADCMPCR1

**Monitor:** AD 監視機能に関する構造体で、大小判定カウント数、判定カウント条件、判定条件、比較対象のアナログ入力チャネルが含まれます。詳細は"データ構成"の ADC\_MonitorTypeDef を参照してください。

**機能:**

本デバイスは、2つの AD 監視機能を持ち、それぞれ設定レジスタで制御します。  
**ADCMPx** 設定で AD 監視レジスタを選択し、**Monitor** で監視機能を設定します。

**補足:** この関数をコールする前に ADC 監視機能を禁止してください。

**戻り値:**

なし

### 3.2.3.16 ADC\_SetHWTrg

通常 AD 変換を開始するためのハードウェア起動要因の選択

**関数のプロトタイプ宣言:**

```
void  
ADC_SetHWTrg(uint8_t HwSource,  
              FunctionalState NewState)
```

**引数:**

**HwSource:** 以下から、通常 AD 変換のハードウェア起動要因を選択します。

- **ADC\_EXT\_TRG:** ADTRG 端子で起動することが可能です。
- **ADC\_MATCH\_TB5RG0:** 16 ビットタイマ/イベントカウンタのコンペアレジスタ 0 一致割り込みで起動することが可能です。(TB5RG0)

**NewState:** 以下から、ハードウェア起因による通常 AD 変換開始の許可/禁止を選択します。

- **ENABLE:** 許可
- **DISABLE:** 禁止

**機能:**

**HwSource** の設定により通常 AD 変換のハードウェア起動要因を設定し、  
**NewState**により通常 AD 変換のハードウェア起動の許可/禁止を選択します。  
この関数は TB5 の設定にも関連しています。

**補足:**

最優先 AD 変換のハードウェア起動要因に使用する場合、外部トリガを通常 AD 変換のハードウェア要因起動に使用することはできません。

**戻り値:**

なし

### 3.2.3.17 ADC\_SetHWTrgTop

最優先 AD 変換を開始するためのハードウェア起動要因の選択

**関数のプロトタイプ宣言:**

```
void
```

ADC\_SetHWTrgTop(uint8\_t **HwSource**,  
FunctionalState **NewState**)

引数:

**HwSource**: 以下から、最優先 AD 変換のハードウェア起動要因を選択します。

- **ADC\_EXT\_TRG**: ADTRG 端子で起動することが可能です。
- **ADC\_MATCH\_TB4RG0**: 16 ビットタイマ/イベントカウンタのコンペアレジスタ 0 一致割り込みで起動することが可能です。(TB4RG0)

**NewState**: 以下から、ハードウェア起因による通常 AD 変換開始の許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

**HwSource** の設定により最優先 AD 変換のハードウェア起動要因を設定し、**NewState** により最優先 AD 変換のハードウェア起動の許可/禁止を選択します。  
この関数は TB4 の設定にも関連しています。

補足:

最優先 AD 変換のハードウェア起動要因に使用する場合、外部トリガを通常 AD 変換のハードウェア要因起動に使用することはできません。

戻り値:

なし

### 3.2.3.18 ADC\_GetConvertState

AD 変換終了フラグの取得(通常と最優先)

関数のプロトタイプ宣言:

WorkState  
ADC\_GetConvertState(void)

引数:

なし

機能:

AD 変換終了フラグ (通常と最優先の両方)を取得します。この関数は、AD 変換が終了したかどうかを確認するために使います。

戻り値:

AD 変換状態:

**NormalComplete** (Bit 1) : 通常 AD 変換終了

**TopComplete** (Bit 3) : 最優先 AD 変換終了

### 3.2.3.19 ADC\_GetConvertResult

AD 変換結果の取得

関数のプロトタイプ宣言:



ADC\_Result  
ADC\_GetConvertResult(uint8\_t **ADREGx**)

## 引数:

**ADREGx**: 以下から、ADC 変換結果レジスタを選択します。

**ADC\_REG\_00**, **ADC\_REG\_01**, **ADC\_REG\_02**, **ADC\_REG\_03**,  
**ADC\_REG\_04**, **ADC\_REG\_05**, **ADC\_REG\_06**, **ADC\_REG\_07**,  
**ADC\_REG\_08**, **ADC\_REG\_09**, **ADC\_REG\_10**, **ADC\_REG\_11**,  
**ADC\_REG\_SP**

## 機能:

AD 変換結果格納フラグ、オーバーランフラグ、変換結果を取得します。

## 補足:

変換結果が格納されると AD 変換格納フラグ **ADREGx** が **DONE** になります。本関数によって変換結果が読み出されると、AD 変換結果格納フラグ **ADREGx** がクリアされます。

変換結果格納レジスタ(ADREGx)の値が読み出される前に変換結果が上書きされた場合、AD 変換結果格納フラグ **ADREGx** に **ADC\_OVERRUN** がセットされます。本関数によってオーバーランフラグが読み出されるとオーバーランフラグがクリアされます。

アナログチャネル入力と AD 変換結果レジスタの関係を下表に示します。

チャネル固定シングル変換モード	
チャネル	格納レジスタ
ADC_AN_00	ADC_REG_00
ADC_AN_01	ADC_REG_01
ADC_AN_02	ADC_REG_02
ADC_AN_03	ADC_REG_03
ADC_AN_04	ADC_REG_04
ADC_AN_05	ADC_REG_05
ADC_AN_06	ADC_REG_06
ADC_AN_07	ADC_REG_07
ADC_AN_08	ADC_REG_08
ADC_AN_09	ADC_REG_09
ADC_AN_10	ADC_REG_10
ADC_AN_11	ADC_REG_11

チャネル固定リピート変換モード	
割り込み発生タイミング	格納レジスタ
Interrupt by each time AD/C	ADC_REG_00
Interrupt by each time 2 AD/C	ADC_REG_00 to ADC_REG_01
Interrupt by each time 3 AD/C	ADC_REG_00 to ADC_REG_02
Interrupt by each time 4 AD/C	ADC_REG_00 to ADC_REG_03
Interrupt by each time 5 AD/C	ADC_REG_00 to ADC_REG_04
Interrupt by each time 6 AD/C	ADC_REG_00 to ADC_REG_05
Interrupt by each time 7 AD/C	ADC_REG_00 to ADC_REG_06
Interrupt by each time 8 AD/C	ADC_REG_00 to ADC_REG_07

チャネルスキャンシングル変換モード / リピート変換モード		
スタートチャネル	スキャンチャネル幅	格納レジスタ
ADC_AN_00	12 channels	ADC_REG_00 to ADC_REG_11

ADC_AN_01	11 channels	ADC_REG_01 to ADC_REG_11
ADC_AN_02	10 channels	ADC_REG_02 to ADC_REG_11
ADC_AN_03	9 channels	ADC_REG_03 to ADC_REG_11
ADC_AN_04	8 channels	ADC_REG_04 to ADC_REG_11
ADC_AN_05	7 channels	ADC_REG_05 to ADC_REG_11
ADC_AN_06	6 channels	ADC_REG_06 to ADC_REG_11
ADC_AN_07	5 channels	ADC_REG_07 to ADC_REG_11
ADC_AN_08	4 channels	ADC_REG_08 to ADC_REG_11
ADC_AN_09	3 channels	ADC_REG_09 to ADC_REG_11
ADC_AN_10	2 channels	ADC_REG_10 to ADC_REG_11
ADC_AN_11	1 channels	ADC_REG_11 to ADC_REG_11

The AD 変換モードの詳細は、関連 API を参照ください。  
最優先 AD 変換結果は、ADC\_REG\_SP に格納されます。

戻り値:

AD 変換結果:

**ADResult** (Bit 0 ~ Bit 11) : AD 変換結果が格納されます

**Stored** (Bit 12) : AD 変換結果格納フラグ

**OverRun** (Bit 13) : オーバーランフラグ

**OutputSwitching** (Bit 14) : AIN 兼用ポートの出力スイッチングフラグ

### 3.2.3.20 ADC\_SetClkSupply

ADC クロック選択

関数のプロトタイプ宣言:

void

ADC\_SetClkSupply(FunctionalState **NewState**)

引数:

**NewState**: 以下から、ADC クロックを選択します。

➤ **ENABLE**: 動作

➤ **DISABLE**: 停止

機能:

ADC クロックを選択します。

戻り値:

なし

### 3.2.3.21 ADC\_SetDMAReq

通常 AD 変換、最優先 AD 変換の各 DMA 起動要因の設定

関数のプロトタイプ宣言:

void

ADC\_SetDMAReq(uint8\_t **DMAReq**,  
FunctionalState **NewState**)

引数:

**DMAReq**: 以下から AD 変換の種類を選択します。

- **ADC\_DMA\_REQ\_NORMAL**: 通常 AD 変換
- **ADC\_DMA\_REQ\_TOP**: 最優先 AD 変換

**NewState**: 以下から DMA 起動の許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

**機能**:

通常 AD 変換、最優先 AD 変換の各 DMA 起動要因を選択します。

**戻り値**:

なし

## 3.2.4 データ構造

### 3.2.4.1 ADC\_MonitorTypeDef

**メンバ**:

uint8\_t

**CmpChannel** 以下から、比較対象のアナログ入力チャネルを選択します:

**ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,**  
**ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07,**  
**ADC\_AN\_08, ADC\_AN\_09, ADC\_AN\_10, ADC\_AN\_11,**  
**ADC\_AN\_12, ADC\_AN\_13, ADC\_AN\_14.**

uint32\_t

**CmpCnt**: 大小判定カウント数を選択します。(1 ~ 16)

ADC\_CmpCondition

**Condition**: 以下から、判定条件を選択します。

- **ADC\_LARGER\_THAN\_CMP\_REG**: 比較レジスタ(ADCMPn (n=0/1))より AD 変換結果が大
- **ADC\_SMALLER\_THAN\_CMP\_REG**: 比較レジスタ(ADCMPn (n=0/1))より AD 変換結果が小

ADC\_CmpCntMode

**CntMode**: 以下から、判定カウント条件を選択します。

- **ADC\_SEQUENCE\_CMP\_MODE**: 連続方式
- **ADC\_CUMULATION\_CMP\_MODE**: 蓄積方式

uint32\_t

**CmpValue**: AD 変換結果比較値を設定します。(0 ~ 4095)

### 3.2.4.2 ADC\_State

**メンバ**:

uint32\_t

**All**: すべての AD 変換状態

**ビットフィールド**:

uint32\_t

**Reserved0** (Bit 0) : 未使用  
uint32\_t  
**NormalComplete** (Bit 1) : 通常 AD 変換終了フラグ  
uint32\_t  
**Reserved1** (Bit 2) : 未使用  
uint32\_t  
**TopComplete** (Bit 3) : 最優先 AD 変換終了フラグ  
uint32\_t  
**Reserved2** (Bit 4 ~ Bit 31) : 未使用

### 3.2.4.3 ADC\_Result

メンバ:

uint32\_t

**All**: すべての AD 変換結果

ビットフィールド:

uint32\_t

**ADResult** (Bit 0 ~ Bit 11) : AD 変換結果の値

uint32\_t

**Stored** (Bit 12) : AD 結果終了フラグ

uint32\_t

**OverRun** (Bit 13) : オーバーランフラグ

uint32\_t

**OutputSwitching** (Bit 14) : AIN 兼用ポートの出カスイッチングフラグ

uint32\_t

**Reserved** (Bit 15 to Bit 31) : 未使用

## 4. CG

### 4.1 概要

本 CG API は以下の機能を提供します。

- 高速発振器、PLL(逡倍回路)の設定
- クロックギア、プリスケールクロック、PLL、発振器の設定
- ウォームアップタイマの設定と結果の読み出し
- 低消費電力モードの設定
- 動作モードの変更 (ノーマルモード、低消費電力モード)
- スタンバイモードに関する割り込みの設定

本ドライバは、以下のファイルで構成されています。

/Libraries/TX03\_Periph\_Driver/src/tmpm365\_cg.c

/Libraries/TX03\_Periph\_Driver/inc/tmpm365\_cg.h

CG のクロックとして、以下のシンボルを使用しています。詳しくは MCU データシートの「クロックシステムブロック図」を参照してください。

**fosc** : 内部発振回路で生成されるクロック、X1、X2 端子より入力されるクロック

**fPLL** : PLL により逡倍されたクロック

**fc** : CGPLLSEL<PLL0SEL>で選択されたクロック(高速クロック)

**fgear** : CGSYSCR<GEAR[2:0]>で選択されたクロック

**fsys** : fgear と同等のクロック

**fperiph** : CGSYSCR<FPSEL>で選択されたクロック

**ΦT0** : CGSYSCR<PRCK[2:0]>で選択されたクロック (プリスケールクロック)

### 4.2 API 関数

#### 4.2.1 関数一覧

- ◆ void CG\_SetFgearLevel(CG\_DivideLevel **DivideFgearFromFc**)
- ◆ CG\_DivideLevel CG\_GetFgearLevel(void)
- ◆ void CG\_SetPhiT0Src(CG\_PhiT0Src **PhiT0Src**)
- ◆ CG\_PhiT0Src CG\_GetPhiT0Src(void)
- ◆ Result CG\_SetPhiT0Level(CG\_DivideLevel **DividePhiT0FromFc**)
- ◆ CG\_DivideLevel CG\_GetPhiT0Level(void)
- ◆ void CG\_SetSCOUTSrc(CG\_SCOUTSrc **Source**)
- ◆ CG\_SCOUTSrc CG\_GetSCOUTSrc(void)
- ◆ void CG\_SetWarmUpTime(CG\_WarmUpSrc **Source**, uint16\_t **Time**)
- ◆ void CG\_StartWarmUp(void)
- ◆ WorkState CG\_GetWarmUpState(void)
- ◆ Result CG\_SetFPLLValue(CG\_FpllValue **NewValue**)
- ◆ CG\_FpllValue CG\_GetFPLLValue(void)
- ◆ Result CG\_SetPLL(FunctionalState **NewState**)

- ◆ FunctionalState CG\_GetPLLState(void)
- ◆ Result CG\_SetFosc(CG\_FoscSrc Source, FunctionalState **NewState**)
- ◆ void CG\_SetFoscSrc(CG\_FoscSrc **Source**)
- ◆ CG\_FoscSrc CG\_GetFoscSrc(void)
- ◆ FunctionalState CG\_GetFoscState(CG\_FoscSrc **Source**)
- ◆ void CG\_SetSTBYMode(CG\_STBYMode **Mode**)
- ◆ CG\_STBYMode CG\_GetSTBYMode(void)
- ◆ void CG\_SetPinStateInStop1Mode(FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetPinStateInStop1Mode(void)
- ◆ Result CG\_SetFcSrc(CG\_FcSrc **Source**)
- ◆ CG\_FcSrc CG\_GetFcSrc(void)
- ◆ void CG\_SetUSBSrcClk(CG\_USBSrc **Source**)
- ◆ CG\_USBSrc CG\_GetUSBSrcClk(void)
- ◆ void CG\_SetUSBClkState(FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetUSBClkState(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)

## 4.2.2 関数の種類

上記関数は以下の 3 種類に分けられます。

- 1) クロックの選択:  
CG\_SetFgearLevel(), CG\_GetFgearLevel(), CG\_SetPhiT0Src(), CG\_GetPhiT0Src(),  
CG\_SetPhiT0Level(), CG\_GetPhiT0Level(), CG\_SetSCOUTSrc(),  
CG\_GetSCOUTSrc(), CG\_SetWarmUpTime(), CG\_StartWarmUp(),  
CG\_GetWarmUpState(), CG\_SetFPLLValue(), CG\_GetFPLLValue(), CG\_SetPLL(),  
CG\_GetPLLState(), CG\_SetFosc(), CG\_SetFoscSrc(), CG\_GetFoscSrc(),  
CG\_GetFoscState(), CG\_SetFcSrc(), CG\_GetFcSrc(), CG\_SetUSBSrcClk(),  
CG\_GetUSBSrcClk(), CG\_SetUSBClkState(), CG\_GetUSBClkState(),  
CG\_SetProtectCtrl()
- 2) スタンバイモードの設定:  
CG\_SetSTBYMode(), CG\_GetSTBYMode(), CG\_SetPinStateInStop1Mode(),  
CG\_GetPinStateInStop1Mode()
- 3) 割り込みの設定:  
CG\_SetSTBYReleaseINTSrc(), CG\_GetSTBYReleaseINTState(), CG\_ClearINTReq(),  
CG\_GetNMIFlag(), CG\_GetResetFlag()

## 4.2.3 関数仕様

### 4.2.3.1 CG\_SetFgearLevel

fgear,fc 間の分周レベル設定

関数のプロトタイプ宣言:

```
void  
CG_SetFgearLevel(CG_DivideLevel DivideFgearFromFc)
```

引数:

**DivideFgearFromFc:** 以下から、fgear,fc 間の分周レベルを選択します。

➤ **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:**  $fgear = fc/16$

**機能:**

$fgear, fc$  間の分周レベルを設定します。

**戻り値:**

なし

#### 4.2.3.2 CG\_GetFgearLevel

$fgear, fc$  間の分周レベルの取得

**関数のプロトタイプ宣言:**

CG\_DivideLevel

CG\_GetFgearLevel(void)

**引数:**

なし。

**機能:**

$fgear, fc$  間の分周レベルを取得します。レジスタから読み出した値が“Reserved” の場合、**CG\_DIVIDE\_UNKNOWN** を返します。

**戻り値:**

$fgear, fc$  間の分周レベルで、下記のいずれかの値になります。

**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:** 無効

#### 4.2.3.3 CG\_SetPhiT0Src

PhiT0(fperiph)ソースの設定

**関数のプロトタイプ宣言:**

void

CG\_SetPhiT0Src(CG\_PhiT0Src **PhiT0Src**)

**引数:**

**PhiT0Src:** 以下から PhiT0 ソースを選択します。

➤ **CG\_PHIT0\_SRC\_FGEAR:**  $fgear$  が PhiT0 ソース

➤ **CG\_PHIT0\_SRC\_FC:**  $fc$  が PhiT0 ソース

**機能:**

PhiT0 (ΦT0) ソースを選択します。

**戻り値:**

なし

#### 4.2.3.4 CG\_GetPhiT0Src

PhiT0 (ΦT0) ソースの取得

**関数のプロトタイプ宣言:**

CG\_PhiT0Src

CG\_GetPhiT0Src(void)

**引数:**

なし。

**機能:**

PhiT0 (ΦT0) ソースを取得します。

**戻り値:**

**CG\_PHIT0\_SRC\_FGEAR** : fgear が PhiT0 ソース

**CG\_PHIT0\_SRC\_FC**: fc が PhiT0 ソース

#### 4.2.3.5 CG\_SetPhiT0Level

PhiT0 (ΦT0) と fc 間の分周レベルの設定

**関数のプロトタイプ宣言:**

Result

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

**引数:**

**DividePhiT0FromFc**: PhiT0 (ΦT0) と fc 間の分周レベルを下記の値から設定します。

- **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

**機能:**

プリスケラークロックの分周レベルを設定します。

**戻り値:**

**SUCCESS**: 設定成功

**ERROR**: エラー

#### 4.2.3.6 CG\_GetPhiT0Level

ΦT0 ,fc 間の分周レベルの取得



**関数のプロトタイプ宣言:**

CG\_DivideLevel

CG\_GetPhiT0Level(void)

**引数:**

なし。

**機能:**

PhiT0( $\Phi T0$ ),  $fc$  間の分周レベルを取得します。レジスタから読み出した値が“Reserved”の場合、**CG\_DIVIDE\_UNKNOWN** を返します。

**戻り値:**

PhiT0( $\Phi T0$ ),  $fc$  間の分周レベル:

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

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

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

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

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

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

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

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

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

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

**CG\_DIVIDE\_UNKNOWN**: 無効データ

#### 4.2.3.7 CG\_SetSCOUTSrc

SCOUT 出力ソースクロック設定

**関数のプロトタイプ宣言:**

void

CG\_SetSCOUTSrc(CG\_SCOUTSrc **Source**)

**引数:**

**Source**: 以下から、SCOUT 出力のソースクロックを選択します。

- **CG\_SCOUT\_SRC\_HALF\_FSYS**:  $fsys/2$  に設定
- **CG\_SCOUT\_SRC\_FSYS**:  $fsys$  に設定
- **CG\_SCOUT\_SRC\_PHIT0**:  $\Phi T0$  に設定

**機能:**

SCOUT 出力のソースクロックを設定します。

**戻り値:**

なし

#### 4.2.3.8 CG\_GetSCOUTSrc

SCOUT 出力ソースクロック設定の取得

**関数のプロトタイプ宣言:**

SCOUTSrc

CG\_GetSCOUTSrc(void)

引数:

なし

機能:

SCOUT 出力ソースクロック設定を取得します。

戻り値:

SCOUT 出力のソースクロック:

- **CG\_SCOUT\_SRC\_HALF\_FSYS**: fsys/2 に設定
- **CG\_SCOUT\_SRC\_FSYS**: fsys に設定
- **CG\_SCOUT\_SRC\_PHIT0**:  $\phi T0$  に設定
- **CG\_SCOUT\_SRC\_UNKNOWN**: 無効データ

#### 4.2.3.9 CG\_SetWarmUpTime

ウォーミングアップ時間の設定

関数のプロトタイプ宣言:

void

CG\_SetWarmUpTime(CG\_WarmUpSrc **Source**,  
uint16\_t **Time**)

引数:

**Source**: 以下から、ウォーミングアップカウンタのソースクロックを選択します。

- **CG\_WARM\_UP\_SRC\_OSC\_INT**: 内部高速発振器を選択
- **CG\_WARM\_UP\_SRC\_OSC\_EXT**: 外部高速発振器を選択

**Time**: ウォーミングアップタイマーのカウント数を選択します。最大値は 0xFFFF です。

機能:

ウォーミングアップ時間とウォーミングアップカウンタを設定します。計算式は下記になります。

ウォーミングアップサイクル数 = (ウォーミングアップ時間) / (ウォームアップクロック周期)

高速発振子 8MHz 使用時、ウォーミングアップ時間 5ms を設定する場合のウォーミングアップサイクル数は以下になります:

(ウォーミングアップ時間) / (ウォームアップクロック周期) = 5ms / (1/8MHz) = 4000cycle = 0x9C40

従って、**Time** = 0x9C40 となります。

戻り値:

なし

#### 4.2.3.10 CG\_StartWarmUp

ウォーミングアップ開始

関数のプロトタイプ宣言:

void

CG\_StartWarmUp(void)

引数:

なし。

機能:

ウォーミングアップを開始します。

戻り値:

なし

#### 4.2.3.11 CG\_GetWarmUpState

ウォーミングアップ動作状態 (動作中、完了)の確認

関数のプロトタイプ宣言:

WorkState

CG\_GetWarmUpState(void)

引数:

なし。

機能:

ウォーミングアップ動作状態を確認します。

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

戻り値:

ウォーミングアップ動作状態:

**DONE**:ウォーミングアップ動作終了

**BUSY**:ウォーミングアップ動作中

#### 4.2.3.12 CG\_SetFPLLValue

PLL (fsys 用)の逡倍数を設定。

関数のプロトタイプ宣言:

Result

CG\_SetFPLLValue(CG\_FpllValue **NewValue**)

引数:

**NewValue**:

➤ **CG\_FPLL\_MULTIPLY\_8**: 8 逡倍

機能:

PLL (fsys 用)の逡倍数を設定します。

戻り値:

**SUCCESS:** 成功

**ERROR:** 失敗

## 4.2.3.13 CG\_GetFPLLValue

PLL 通倍値の取得

**関数のプロトタイプ宣言:**

CG\_FpllValue

CG\_GetFPLLValue(void)

**引数:**

なし

**機能:**

PLL 通倍値を取得します。

レジスタ値が“Reserved”の場合、本 API の戻り値は **CG\_FPLL\_MULTIPLY\_UNKNOWN** です。

**戻り値:**

PLL 通倍値:

**CG\_FPLL\_MULTIPLY\_8:** 8 通倍値

**CG\_FPLL\_MULTIPLY\_UNKNOWN:** 無効値

## 4.2.3.14 CG\_SetPLL

PLL 回路の設定

**関数のプロトタイプ宣言:**

Result

CG\_SetPLL(FunctionalState **NewState**)

**引数:**

**NewState:**

- **ENABLE:** PLL 回路を使用する
- **DISABLE:** PLL 回路を使用しない

**機能:**

PLL 回路の有効/無効を設定します。

**戻り値:**

**SUCCESS:** 成功

**ERROR:** 失敗

## 4.2.3.15 CG\_GetPLLState

PLL 回路の状態の取得

**関数のプロトタイプ宣言:**

FunctionalState

CG\_GetPLLState(void)

引数:

なし。

機能:

PLL 回路の状態を取得します。

戻り値:

PLL 回路の状態

**ENABLE:** PLL 有効

**DISABLE:** PLL 無効

#### 4.2.3.16 CG\_SetFosc

高速発振器(fosc)の有効/無効設定

関数のプロトタイプ宣言:

Result

CG\_SetFosc(CG\_FoscSrc **Source**,  
FunctionalState **NewState**)

引数:

**Source:** 以下から、fosc のソースクロックを選択します。

- **CG\_FOSC\_OSC\_EXT:** 外部高速発信
- **CG\_FOSC\_OSC\_INT:** 内部高速発信

**NewState:** 以下から、高速発振器の有効/無効を設定します。

- **ENABLE:** 有効
- **DISABLE:** 無効

機能:

高速発信器の有効/無効を設定します。

戻り値:

**SUCCESS:** 成功

**ERROR:** 失敗

#### 4.2.3.17 CG\_SetFoscSrc

高速発振器(fosc)のソース設定

関数のプロトタイプ宣言:

void

CG\_SetFoscSrc(CG\_FoscSrc Source)

引数:

**Source:** fosc のソースを選択します。

- **CG\_FOSC\_OSC\_EXT:** 外部高速発信子
- **CG\_FOSC\_CLKIN\_EXT:** 外部クロック入力
- **CG\_FOSC\_OSC\_INT:** 内部高速発信器

**機能:**

高速発振器(fosc)のソースを設定します。

**戻り値:**

なし

#### 4.2.3.18 CG\_GetFoscSrc

高速発振器のソース取得

**関数のプロトタイプ宣言:**

CG\_FoscSrc

CG\_GetFoscSrc(void)

**引数:**

なし。

**機能:**

高速発振器のソースを取得します。

**戻り値:**

高速発振器のソース

**CG\_FOSC\_OSC\_EXT:** 外部高速発信子

**CG\_FOSC\_CLKIN\_EXT:** 外部クロック入力

**CG\_FOSC\_OSC\_INT:** 内部高速発信器

#### 4.2.3.19 CG\_GetFoscState

高速発信器の状態

**関数のプロトタイプ宣言:**

FunctionalState

CG\_GetFoscState(CG\_FoscSrc Source)

**引数:**

**Source:** 以下から、fosc のソースを指定します。

➤ **CG\_FOSC\_OSC\_EXT:** 外部高速発信

➤ **CG\_FOSC\_OSC\_INT:** 内部高速発信

**機能:**

高速発信器の状態を取得します。

**戻り値:**

fosc の状態

**ENABLE:** fosc が有効

**DISABLE:** fosc が無効

#### 4.2.3.20 CG\_SetSTBYMode

スタンバイモードの選択

**関数のプロトタイプ宣言:**

void  
CG\_SetSTBYMode(CG\_STBYMode **Mode**)

**引数:**

**Mode:** 以下から、スタンバイモードを選択します。

- **CG\_STBY\_MODE\_STOP1:** STOP1 モード (内部発振器も含めてすべての内部回路が停止)
- **CG\_STBY\_MODE\_IDLE:** IDLE モード (CPU が停止)

**機能:**

スタンバイモードを選択します。

**戻り値:**

なし

#### 4.2.3.21 CG\_GetSTBYMode

スタンバイモードの取得

**関数のプロトタイプ宣言:**

CG\_STBYMode  
CG\_GetSTBYMode(void)

**引数:**

なし。

**機能:**

スタンバイモードの設定状態を取得します。

“Reserved”の場合、“**CG\_STBY\_MODE\_UNKNOWN**”を返却します。

**戻り値:**

**CG\_STBY\_MODE\_STOP1:** STOP1 モード

**CG\_STBY\_MODE\_IDLE:** IDLE モード

**CG\_STBY\_MODE\_UNKNOWN:** 無効なモード

#### 4.2.3.22 CG\_SetPinStateInStop1Mode

STOP1 モード中の端子状態の設定

**関数のプロトタイプ宣言:**

void  
CG\_SetPinStateInStop1Mode(FunctionalState **NewState**)

**引数:**

**NewState:**

- **DISABLE:** STOP1 モード中端子をドライブしません
- **ENABLE:** STOP1 モード中端子をドライブします

STOP1 モード中の端子状態制御については、MCU データシートの“低消費電力モード”を参照してください。

**機能:**

STOP1 モード時の端子状態を設定します。

**戻り値:**

なし

#### 4.2.3.23 CG\_GetPinStateInStop1Mode

STOP1 モード中の端子状態の取得。

**関数のプロトタイプ宣言:**

FunctionalState

CG\_GetPinStateInStop1Mode(void)

**引数:**

なし。

**機能:**

STOP1 モード中の端子状態を取得します。

**戻り値:**

**DISABLE:** STOP1 モード中端子をドライブしません

**ENABLE:** STOP1 モード中端子をドライブします

#### 4.2.3.24 CG\_SetFcSrc

fc のソース選択

**関数のプロトタイプ宣言:**

Result

CG\_SetFcSrc(CG\_FcSrc **Source**)

**引数:**

**Source:** fc のソースを選択します。

➤ **CG\_FC\_SRC\_FOSC** : fosc を使用

➤ **CG\_FC\_SRC\_QUARTER\_FPLL** : fpll/4 を使用

**機能:**

fc のソースクロックを選択します。

**戻り値:**

**SUCCESS:** 成功

**ERROR:** 失敗

#### 4.2.3.25 CG\_GetFcSrc

fc ソースの取得

**関数のプロトタイプ宣言:**

CG\_FcSrc

CG\_GetFosc (void)



引数:

なし

機能:

fc ソースを取得します。

戻り値:

fc のソース:

**CG\_FC\_SRC\_FOSC**: fosc

**CG\_FC\_SRC\_QUARTER\_FPLL**: fpll/4

#### 4.2.3.26 CG\_SetUSBSrcClk

USB ソースクロック選択

関数のプロトタイプ宣言:

void

CG\_SetUSBSrcClk(CG\_USBSrc **Source**)

引数:

**Source**: 以下から USB クロックソースを選択します。

- **CG\_USB\_SRC\_CLK\_PLL**: PLL クロック(fpll)
- **CG\_USB\_SRC\_CLK\_EXT**: 外部入力クロック(EHCLKIN)

機能:

USB ソースクロックを選択します。

戻り値:

なし

#### 4.2.3.27 CG\_GetUSBSrcClk

USB ソースクロックの設定状態取得

関数のプロトタイプ宣言:

CG\_USBSrc

CG\_GetUSBSrcClk(void)

引数:

なし

機能:

USB ソースクロックの設定状態を取得します。

戻り値:

USB ソースクロックの設定状態:

**CG\_USB\_SRC\_CLK\_PLL**: PLL クロック(fpll)

**CG\_USB\_SRC\_CLK\_EXT**: 外部入力クロック(EHCLKIN)

## 4.2.3.28 CG\_SetUSBClkState

USB ソースクロックの設定

関数のプロトタイプ宣言:

void

CG\_SetUSBClkState(FunctionalState **NewState**)

引数:

**NewState**

➤ **DISABLE**: 停止(OFF)

➤ **ENABLE**: 設定(ON)

機能:

USB ソースクロックを設定します。

戻り値:

なし

## 4.2.3.29 CG\_GetUSBClkState

USB ソースクロック設定状態の取得

関数のプロトタイプ宣言:

FunctionalState

CG\_GetUSBClkState(void)

引数:

なし。

機能:

USB ソースクロック設定状態を取得します。

戻り値:

USB ソースクロックの設定状態:

**DISABLE**: 停止(OFF)

**ENABLE**: 設定(ON)

## 4.2.3.30 CG\_SetProtectCtrl

CG レジスタの書き込み制御

関数のプロトタイプ宣言:

void

CG\_SetProtectCtrl(FunctionalState **NewState**)

引数:

**NewState**

➤ **DISABLE**: 書き込み禁止

➤ **ENABLE**: 書き込み許可

機能:

CGレジスタの書き込み許可/禁止を設定します。

戻り値:  
なし

## 4.2.3.31 CG\_SetSTBYReleaseINTSrc

スタンバイモードの解除割り込みソースの設定

関数のプロトタイプ宣言:

```
void  
CG_SetSTBYReleaseINTSrc (CG_INTSrc INTSource,  
                          CG_INTActiveState ActiveState,  
                          FunctionalState NewState)
```

引数:

**INTSource**: 以下から、スタンバイモードの解除割り込みソースを選択します。

- 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
- CG\_INT\_SRC\_6 : INT6
- CG\_INT\_SRC\_7 : INT7
- CG\_INT\_SRC\_8 : INT8
- CG\_INT\_SRC\_9 : INT9
- CG\_INT\_SRC\_USB\_PON : USB Poewe ON(V-Bus Connect)検出割り込み
- CG\_INT\_SRC\_USB\_WKUP : USB Wake-up 割り込み

**ActiveState**: 以下から、解除トリガのアクティブ状態を選択します。

- CG\_INT\_ACTIVE\_STATE\_L: "Low"レベル
- CG\_INT\_ACTIVE\_STATE\_H: "High"レベル
- CG\_INT\_ACTIVE\_STATE\_FALLING: ↓エッジ
- CG\_INT\_ACTIVE\_STATE\_RISING: ↑エッジ
- CG\_INT\_ACTIVE\_STATE\_BOTH\_EDGES: 両エッジ

**NewState**: 以下から、解除トリガの有効/無効を選択します。

- ENABLE: 許可
- DISABLE: 禁止

機能:

スタンバイモードの解除割り込みソースを設定します。

戻り値:  
なし

## 4.2.3.32 CG\_GetSTBYReleaseINTState

スタンバイモードの解除割り込みソースのアクティブ状態の取得

関数のプロトタイプ宣言:

CG\_INT\_ActiveState  
CG\_GetSTBYReleaseINTSrc(CG\_INTSrc **INTSource**)

引数:

**INTSource**: 以下から、解除割り込みソースを選択します。

CG\_INT\_SRC\_0, CG\_INT\_SRC\_1, CG\_INT\_SRC\_2, CG\_INT\_SRC\_3,  
CG\_INT\_SRC\_4, CG\_INT\_SRC\_5, CG\_INT\_SRC\_6, CG\_INT\_SRC\_7,  
CG\_INT\_SRC\_8, CG\_INT\_SRC\_9,  
CG\_INT\_SRC\_USB\_PON, CG\_INT\_SRC\_USB\_WKUP.

機能:

スタンバイモードの解除割り込みソースのアクティブ状態を取得します。

戻り値:

解除割り込みソースのアクティブ状態

CG\_INT\_ACTIVE\_STATE\_L: "Low"レベル

CG\_INT\_ACTIVE\_STATE\_H: "High"レベル

CG\_INT\_ACTIVE\_STATE\_FALLING: ↓エッジ

CG\_INT\_ACTIVE\_STATE\_RISING: ↑エッジ

CG\_INT\_ACTIVE\_STATE\_BOTH\_EDGES: 両エッジ

CG\_INT\_ACTIVE\_STATE\_INVALID: 無効な値

#### 4.2.3.33 CG\_ClearINTReq

スタンバイ解除割り込み要求のクリア

関数のプロトタイプ宣言:

void  
CG\_ClearINTReq(CG\_INTSrc **INTSource**)

引数:

**INTSource**: 以下から、解除割り込みソースを選択します。

CG\_INT\_SRC\_0, CG\_INT\_SRC\_1, CG\_INT\_SRC\_2, CG\_INT\_SRC\_3,  
CG\_INT\_SRC\_4, CG\_INT\_SRC\_5, CG\_INT\_SRC\_6, CG\_INT\_SRC\_7,  
CG\_INT\_SRC\_8, CG\_INT\_SRC\_9,  
CG\_INT\_SRC\_USB\_PON, CG\_INT\_SRC\_USB\_WKUP.

機能:

スタンバイ解除割り込み要求をクリアします。

戻り値:

なし

#### 4.2.3.34 CG\_GetNMIFlag

NMI 起動要因フラグの取得

関数のプロトタイプ宣言:

CG\_NMI\_Factor  
CG\_GetNMIFlag (void)

引数:

なし

**機能:**

NMI 起動要因フラグを取得します。

**戻り値:**

NMI 起動要因:

**WDT** (Bit 0) :WDT による NMI 発生

**NMIPin**(Bit 1):NMI 端子 による NMI 発生

## 4.2.3.35 CG\_GetResetFlag

リセットフラグの取得とクリア

**関数のプロトタイプ宣言:**

CG\_ResetFlag

CG\_GetResetFlag(void)

**引数:**

なし

**機能:**

リセットフラグの取得とクリアを行います。

**戻り値:**

リセットフラグ:

**ResetPin** (Bit 0) リセット端子によるリセット

**Reserved**(Bit1) 未使用

**WDTReset** (Bit 2) WDT リセット

**Reserved**(Bit3) 未使用

**DebugReset** (Bit 4) SYSRESETREQ リセット

## 4.2.4 データ構造

### 4.2.4.1 CG\_NMIFactor

**メンバ:**

uint32\_t

**All** すべての NMI 要因

**ビットフィールド:**

uint32\_t

**WDT**(Bit 0) WDT による NMI 発生

uint32\_t

**NMIPin**(Bit 1) NMI 端子による NMI 発生

### 4.2.4.2 CG\_ResetFlag

**メンバ:**

uint32\_t

**All** すべてのリセット要因

ビットフィールド:

uint32\_t

**ResetPin**(Bit 0)      RESET 端子によるリセット

uint32\_t

**Reserved**(Bit 1)      未使用

uint32\_t

**WDTReset**(Bit 2)      WDT によるリセット

uint32\_t

**Reserved**(Bit 3)      未使用

uint32\_t

**DebugReset**(Bit 4) <SYSResetREQ>によるリセット

## 5. DMAC

### 5.1 概要

本デバイスは、DMA 要求選択レジスタにより制御される 1 ユニットの DMA コントローラ (UNITA) を内蔵しています。ユニット A は、4 つの転送タイプのどれかで動作します。4 つの転送タイプは、メモリ-メモリ、メモリ-周辺回路、周辺回路-メモリ、周辺回路-周辺回路です。ユニット A は 2 チャンネルの DMAC を内蔵し、DMA チャンネル 0 は DMA チャンネル 1 より優先度が高くなります。

DMA ドライバ API は DMAC 設定機能を持ち、引数には、ソースアドレス、ソースアドレスのインクリメント状態、転送ソースのビット幅、転送ソースのバースト幅、宛先アドレス、宛先アドレスのインクリメント状態、転送先ビット幅、転送先バーストサイズ、転送サイズ、転送方向、データ転送ペリフェラル、転送割り込みステータスなどがあります。

全ドライバ API は、アプリ使用の API 定義を格納する以下のファイルで構成されています。

\\Libraries\\TX03\_Periph\_Driver\\src\\tmpm365\_dmac.c

\\Libraries\\TX03\_Periph\_Driver\\inc\\tmpm365\_dmac.h

### 5.2 API 関数

#### 5.2.1 関数一覧

- ◆ void DMAC\_Enable(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ void DMAC\_Disable(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ DMAL\_INTRReq DMAL\_GetINTRReq(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ DMAL\_TxINTRReq DMAL\_GetTxINTRReq(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**);
- ◆ void DMAL\_ClearTxINTRReq(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**, DMAL\_INTSrc **INTSource**);
- ◆ DMAL\_TxINTRReq DMAL\_GetRawTxINTRReq(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**);
- ◆ WorkState DMAL\_GetChannelTxState(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**);
- ◆ void DMAL\_SetSWBurstReq(DMALA\_ReqNum **BurstReq**);
- ◆ DMAL\_BurstReqState DMAL\_GetSWBurstReqState(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ void DMAL\_SetLinkedList(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**, uint32\_t **LinkedAddr**);
- ◆ WorkState DMAL\_GetFIFOState(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**);
- ◆ void DMAL\_SetDMAHalt(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**, FunctionalState **NewState**);
- ◆ void DMAL\_SetLockedTx(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**, FunctionalState **NewState**);
- ◆ void DMAL\_SetTxINTConfig(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**, DMAL\_INTSrc **INTSource**, FunctionalState **NewState**);
- ◆ void DMAL\_SetDMAChannel(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**, FunctionalState **NewState**);

- ◆ void DMAC\_Init(TSB\_DMAM\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**,  
DMAC\_InitTypeDef \* **InitStruct**);

## 5.2.2 関数の種類

関数は、主に以下の 5 種類に分かれています。

- 1) DMAC 基本設定:  
DMAC\_Enable(), DMAC\_Disable(), DMAC\_SetDMAMChannel(), DMAC\_Init()
- 2) DMA 転送割り込みステータス、FIFO または DMA チャンネル状態:  
DMAC\_GetINTReq(), DMAC\_GetTxINTReq(), DMAC\_GetRawTxINTReq(),  
DMAC\_GetChannelTxState(), DMAC\_GetFIFOState()
- 3) DMA 割り込み設定、DMA 割り込み要求のクリア:  
DMAC\_ClearTxINTReq(), DMAC\_SetTxINTConfig()
- 4) DMA ソフトウェア要求の設定、および取得:  
DMAC\_SetSWBurstReq(), DMAC\_GetSWBurstReqState(), DMAC\_SetLinkedList(),  
DMAC\_GetSWSingleReqState()
- 5) その他の設定:  
DMAC\_SetDMAHalt(), DMAC\_SetLockedTx()

## 5.2.3 関数仕様

補足: 下記の全 API において、パラメータ“TSB\_DMAM\_TypeDef \* **DMACx**” は以下を選択してください。

**DMAC\_UNIT\_A**

### 5.2.3.1 DMAC\_Enable

DMA 回路動作の許可

関数のプロトタイプ宣言:

void  
DMAC\_Enable(TSB\_DMAM\_TypeDef \* **DMACx**);

引数:

**DMACx**: ユニットを選択します。

機能:

DMA 回路動作を許可します。

補足:

DMAC を使用する際、まず本関数をコールして DMA 回路を動作させてください。  
DMA 回路用レジスタは、DMA 回路が動作していないと書き込み/読み出しができません。

戻り値:

なし

### 5.2.3.2 DMAC\_Disable

DMA 回路動作の禁止

関数のプロトタイプ宣言:

void



DMAC\_Disable(TSB\_DMACH\_TypeDef \* **DMACx**);

引数:

**DMACx**: ユニットを選択します。

機能:

DMA 回路動作を禁止します。

戻り値:

なし

### 5.2.3.3 DMAC\_GetINTReq

DMA チャンネル割り込みステータスの取得

関数のプロトタイプ宣言:

DMAC\_INTReq

DMAC\_GetINTReq(TSB\_DMACH\_TypeDef \* **DMACx**);

引数:

**DMACx**: ユニットを選択します。

機能:

DMA チャンネル割り込み要求状態を取得します。

戻り値:

割り込み要求状態を返します。構造体"DMAC\_INTReq"の詳細はデータ構造を参照してください。

### 5.2.3.4 DMAC\_GetTxINTReq

DMA チャンネル転送割り込み要求状態の取得

関数のプロトタイプ宣言:

DMAC\_TxINTReq

DMAC\_GetTxINTReq(TSB\_DMACH\_TypeDef \* **DMACx**,  
DMACH\_Channel **Chx**);

引数:

**DMACx**: ユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0**: チャンネル 0
- **DMAC\_CHANNEL\_1**: チャンネル 1

機能:

DMA チャンネル転送割り込み要求状態を取得します。

戻り値:

以下のいずれかの DMA チャンネル転送割り込み要求状態を返します。

**DMAC\_TX\_NO\_REQ**: 転送割り込み要求なし

**DMAC\_TX\_END\_REQ**: 転送終了割り込み要求あり

**DMAC\_TX\_ERR\_REQ:** 転送エラー割り込み要求あり  
**DMAC\_TX\_REQS:** 2 つ以上の割り込み要求あり

## 5.2.3.5 DMAC\_ClearTxINTReq

転送割り込み要求のクリア

**関数のプロトタイプ宣言:**

```
void  
DMAC_ClearTxINTReq(TSB_DMAM_TypeDef * DMACx,  
                   DMAC_Channel Chx,  
                   DMAC_INTSrc INTSource);
```

**引数:**

**DMACx:** ユニットを選択します。

**Chx:** 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0:** チャンネル 0
- **DMAC\_CHANNEL\_1:** チャンネル 1

**INTSource:** 以下からリリース割り込みソースを選択します。

- **DMAC\_INT\_TX\_END:** DMA 転送終了割り込み
- **DMAC\_INT\_TX\_ERR:** DMA 転送エラー割り込み

**機能:**

転送割り込み要求をクリアします。

**戻り値:**

なし

## 5.2.3.6 DMAC\_GetRawTxINTReq

DMA チャンネルの許可前転送終了割り込み発生状態の取得

**関数のプロトタイプ宣言:**

```
DMAC_TxINTReq  
DMAC_GetRawTxINTReq(TSB_DMAM_TypeDef * DMACx,  
                    DMAC_Channel Chx);
```

**引数:**

**DMACx:** ユニットを選択します。

**Chx:** 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0:** チャンネル 0
- **DMAC\_CHANNEL\_1:** チャンネル 1

**機能:**

DMA チャンネルの許可前転送終了割り込み発生状態を取得します。

**戻り値:**

以下のいずれかの DMA チャンネルの許可前転送終了割り込み発生状態を返します。

**DMAC\_TX\_NO\_REQ:** 転送前の転送終了割り込み発生なし

DMAC\_TX\_END\_REQ: 転送終了割り込みあり  
DMAC\_TX\_ERR\_REQ: 転送エラー割り込みあり  
DMAC\_TX\_REQS : 2 つ以上の割り込み要求あり

## 5.2.3.7 DMAC\_GetChannelTxState

DMA チャンネル転送状態の取得

関数のプロトタイプ宣言:

WorkState

```
DMAC_GetChannelTxState(TSB_DMAL_TypeDef * DMACx,  
                        DMA_Channel Chx);
```

引数:

**DMACx**: ユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0**: チャンネル 0
- **DMAC\_CHANNEL\_1**: チャンネル 1

機能:

本関数は、**Chx** が **DMAC\_CHANNEL\_0** の時、DMA チャンネル 0 転送状態を取得します。**Chx** が **DMAC\_CHANNEL\_1** の時、DMA チャンネル 1 転送状態を取得します。戻り値が **BUSY** の時は、DMA チャンネルは有効で、データ送信中であることを示します。戻り値が **DONE** の時は、DMA チャンネルは無効で、データ送信は終了していることを示します。

戻り値:

以下どちらかの DMA 転送状態を返します。

**BUSY**、または **DONE**

## 5.2.3.8 DMACA\_SetSWBurstReq

ソフトウェアによるユニット A の DMA バースト転送要求の設定

関数のプロトタイプ宣言:

void

```
DMACA_SetSWBurstReq(DMACA_ReqNum BurstReq);
```

引数:

**BurstReq**: 以下のいずれかのバースト要求番号を選択します。

- **DMACA\_SIO0\_UART0\_RX**: SIO0/UART0 受信
- **DMACA\_SIO0\_UART0\_TX**: SIO0/UART0 送信
- **DMACA\_SIO1\_UART1\_RX**: SIO1/UART1 受信
- **DMACA\_SIO1\_UART1\_TX**: SIO1/UART1 送信
- **DMACA\_TMRB8\_CMP\_MATCH**: TMRB8 コンペア一致
- **DMACA\_TMRB9\_CMP\_MATCH**: TMRB9 コンペア一致
- **DMACA\_TMRB0\_CAPTURE0**: TMRB0 キャプチャ 0 割り込み
- **DMACA\_TMRB4\_CAPTURE0**: TMRB4 キャプチャ 0 割り込み
- **DMACA\_TMRB4\_CAPTURE1**: TMRB4 キャプチャ 1 割り込み
- **DMACA\_TMRB5\_CAPTURE0**: TMRB5 キャプチャ 0 割り込み
- **DMACA\_TMRB5\_CAPTURE1**: TMRB5 キャプチャ 1 割り込み

- **DMACA\_NORMAL\_ADC**: 通常 A/D 変換終了
- **DMACA\_I2C0\_SIO0\_RX**: I2C0 受信
- **DMACA\_I2C0\_SIO0\_TX**: I2C0 送信
- **DMACA\_I2C1\_SIO1\_RX**: I2C1 受信
- **DMACA\_I2C1\_SIO1\_TX**: I2C1 送信

**機能:**

ソフトウェアによる DMA ユニット A のバースト転送要求を設定します。  
ソフトウェアでの DMA 要求とハードウェアからの同時実行は禁止です。

**戻り値:**

なし

### 5.2.3.9 DMAC\_GetSWBurstReqState

ソフトウェアによる DMA バースト要求状態の取得

**関数のプロトタイプ宣言:**

```
DMAC_BurstReqState  
DMAC_GetSWBurstReqState(TSB_DMCA_TypeDef * DMACx);
```

**引数:**

**DMACx**: 以下からユニットを選択します。

**機能:**

ソフトウェアによる DMA バースト要求状態を取得します。

**戻り値:**

DMA バースト要求状態を返します。構造体"DMAC\_BurstReqState"の詳細はデータ構造を参照してください。

### 5.2.3.10 DMAC\_SetLinkedList

DMA チャンネル・コレクションアイテムレジスタの設定

**関数のプロトタイプ宣言:**

```
void  
DMAC_SetLinkedList(TSB_DMCA_TypeDef * DMACx,  
                   DMAC_Channel Chx,  
                   uint32_t LinkedAddr);
```

**引数:**

**DMACx**: 以下からユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0**: チャンネル 0
- **DMAC\_CHANNEL\_1**: チャンネル 1

**LinkedAddr**: 次の転送開始アドレスを指定します。0xFFFFFFFF0 まで指定可能です。

**機能:**

DMA チャンネル・コレクションレジスタを設定します。スキッター・ギャザー機能が不要な場合は、**LinkedAddr** を 0 に設定し本関数を呼び出します。

**補足:**

スキッター・ギャザー機能を用いる場合、転送ソース、転送先データアドレスは、コレクション(LinkedList)を最初に作成する必要があります。

各設定は LLI (コレクション LinkedList) と呼ばれます。各 LLI はデータブロック転送を制御します。また、DMA が通常設定であることを示し、連続データの転送を制御します。

DMA 転送終了ごとに、DMA 動作を継続するために次の LLI 設定がロードされます。(デイジーチェーン)

コレクションと共に設定されるアイテムは、以下の4ワードで設定されます。

- 1) DMACCxSrcAddr
- 2) DMACCxDestAddr
- 3) DMACCxLLI
- 4) DMACCxControl

**戻り値:**

なし

## 5.2.3.11 DMAC\_GetFIFOState

FIFO 状態の取得

**関数のプロトタイプ宣言:**

WorkState

DMAC\_GetFIFOState(TSB\_DMAL\_TypeDef \* **DMACx**,  
DMAC\_Channel **Chx**);

**引数:**

**DMACx**: 以下からユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0**: チャンネル 0
- **DMAC\_CHANNEL\_1**: チャンネル 1

**機能:**

FIFO 状態を取得します。

戻り値が **BUSY** の場合は FIFO にデータが存在することを示し、**DONE** の場合は FIFO にデータがないことを示します。

**戻り値:**

FIFO 状態:

**BUSY**、または **DONE**

## 5.2.3.12 DMAC\_SetDMAHalt

DMA 要求の設定

**関数のプロトタイプ宣言:**

void

DMAC\_SetDMAHalt(TSB\_DMAL\_TypeDef \* **DMACx**,  
DMAC\_Channel **Chx**,

FunctionalState **NewState**);

引数:

**DMACx**: 以下からユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0**: チャンネル 0
- **DMAC\_CHANNEL\_1**: チャンネル 1

**NewState**: 以下から、DMA 要求受付制御を選択します。

- **ENABLE**: DMA 要求 受付
- **DISABLE**: DMA 要求 無視

機能:

DMA 要求受付制御を設定します。

戻り値:

なし

### 5.2.3.13 DMAC\_SetLockedTx

ロック転送の設定

関数のプロトタイプ宣言:

```
void  
DMAC_SetLockedTx(TSB_DMACH_TypeDef * DMACx,  
                  DMACH_Channel Chx,  
                  FunctionalState NewState);
```

引数:

**DMACx**: 以下からユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0**: チャンネル 0
- **DMAC\_CHANNEL\_1**: チャンネル 1

**NewState**: 以下から、ロック転送設定を選択します。

- **ENABLE**: ロック転送 許可
- **DISABLE**: ロック転送 禁止

機能:

ロック転送を設定します。

戻り値:

なし

### 5.2.3.14 DMAC\_SetTxINTConfig

転送割り込みの設定

関数のプロトタイプ宣言:

```
void  
DMAC_SetTxINTConfig(TSB_DMAC_TypeDef * DMACx,  
                    DMAC_Channel Chx,  
                    DMAC_INTSrc INTSource,  
                    FunctionalState NewState);
```

**引数:**

**DMACx:** 以下からユニットを選択します。

**Chx:** 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0:** チャンネル 0
- **DMAC\_CHANNEL\_1:** チャンネル 1

**INTSource:** 以下から、割り込みソースを選択します。

- **DMAC\_INT\_TX\_END:** 転送終了割り込み
- **DMAC\_INT\_TX\_ERR:** エラー割り込み

**NewState:** 以下から、割り込み状態を選択します。

- **ENABLE:** 許可
- **DISABLE:** 禁止

**機能:**

転送割り込みを設定します。

**戻り値:**

なし

## 5.2.3.15 DMAC\_SetDMAChannel

DMA チャンネルの許可/禁止設定

**関数のプロトタイプ宣言:**

```
void  
DMAC_SetDMAChannel(TSB_DMAC_TypeDef * DMACx,  
                   DMAC_Channel Chx,  
                   FunctionalState NewState);
```

**引数:**

**DMACx:** 以下からユニットを選択します。

**Chx:** 以下から DMA チャンネルを選択します。

- **DMAC\_CHANNEL\_0:** チャンネル 0
- **DMAC\_CHANNEL\_1:** チャンネル 1

**NewState:** 以下から、DMA チャンネルの許可/禁止を選択します。

- **ENABLE:** 許可
- **DISABLE:** 禁止

**機能:**

DMA チャンネルの許可/禁止を設定します。

DMA チャンネルの初期設定を行った後に本関数をコールし、DMA チャンネルを有効にしてください。本関数を使用し、DMA チャンネルを無効にすると、FIFO 中のデータが失われます。

FIFO 中のデータ喪失を防ぐため、**DMAC\_SetDMAHalt()** をコールし、DMA 要求を無視した後、**DMAC\_GetFIFOState()** をコールし、FIFO のステータスを取得してください。その後、本関数をコールし、DMA チャンネルを無効にしてください。

戻り値:  
なし

## 5.2.3.16 DMAC\_Init

DMA チャンネルの初期設定

関数のプロトタイプ宣言:

```
void  
DMAC_Init(TSB_DMACH_TypeDef * DMACx,  
           DMACH_Channel Chx,  
           DMACH_InitTypeDef * InitStruct);
```

引数:

**DMACx**: 以下からユニットを選択します。

**Chx**: 以下から DMA チャンネルを選択します。

- **DMACH\_CHANNEL\_0**: チャンネル 0
- **DMACH\_CHANNEL\_1**: チャンネル 1

**InitStruct**: 基本的な DMA 設定を含む構造体で、転送元アドレス、転送元アドレスインクリメントステート、転送元ビット幅、転送元バーストサイズ、転送先アドレス、転送先アドレスインクリメントステート、転送先ビット幅、転送先バーストサイズ、転送サイズ、転送方向、転送ペリフェラル、転送割り込み状態が含まれます。(詳細は“データ構造”を参照してください)

機能:

DMA チャンネルの初期設定を行います。

補足:

**DMAC\_SetDMAChannel()**をコールする前に、本関数を用いて初期設定を行ってください。

戻り値:  
なし

## 5.2.4 データ構造

### 5.2.4.1 DMACH\_InitTypeDef

メンバ:

uint32\_t

**TxDirection**: 以下から、転送方向を選択します。

- **DMACH\_MEMORY\_TO\_MEMORY**: メモリ->メモリ
- **DMACH\_MEMORY\_TO\_PERIPH**: メモリ->周辺回路
- **DMACH\_PERIPH\_TO\_MEMORY**: 周辺回路->メモリ



➤ **DMAC\_PERIPH\_TO\_PERIPH**: 周辺回路->周辺回路

uint32\_t

**SrcAddr**: 転送元アドレスを設定します。

uint32\_t

**DstAddr**: 転送先アドレスを設定します。

FunctionalState

**SrcIncrementState**: 以下から、転送元アドレスのインクリメント設定を選択します。  
**ENABLE**、または **DISABLE**。

FunctionalState

**DstIncrementState**: 以下から、転送先アドレスのインクリメント設定を選択します。  
**ENABLE**、または **DISABLE**。

DMAC\_BitWidth

**SrcBitWidth**: 以下から、転送元データの幅を選択します。

- **DMAC\_BYTE**: バイト
- **DMAC\_HALF\_WORD**: ハーフワード
- **DMAC\_WORD**: ワード

DMAC\_BurstSize

**SrcBurstSize**: 以下から、転送元のバーストサイズを選択します。

- **DMAC\_1\_BEAT**: 1 ビート
- **DMAC\_4\_BEATS**: 4 ビート
- **DMAC\_8\_BEATS**: 8 ビート
- **DMAC\_16\_BEATS**: 16 ビート
- **DMAC\_32\_BEATS**: 32 ビート
- **DMAC\_64\_BEATS**: 64 ビート
- **DMAC\_128\_BEATS**: 128 ビート
- **DMAC\_256\_BEATS**: 256 ビート

DMAC\_BurstSize

**DstBurstSize**: 以下から、転送先のバーストサイズを選択します。

- **DMAC\_1\_BEAT**: 1 ビート
- **DMAC\_4\_BEATS**: 4 ビート
- **DMAC\_8\_BEATS**: 8 ビート
- **DMAC\_16\_BEATS**: 16 ビート
- **DMAC\_32\_BEATS**: 32 ビート
- **DMAC\_64\_BEATS**: 64 ビート
- **DMAC\_128\_BEATS**: 128 ビート
- **DMAC\_256\_BEATS**: 256 ビート

uint32\_t

**TxSize**: 最大転送数で、最大値は 0x0FFF です。

DMACA\_ReqNum

**A\_TxDstPeriph**: 以下のいずれかのバースト要求番号を選択します。

- **DMACA\_SIO0\_UART0\_RX**: SIO0/UART0 受信
- **DMACA\_SIO0\_UART0\_TX**: SIO0/UART0 送信

- **DMACA\_SIO1\_UART1\_RX**: SIO1/UART1 受信
- **DMACA\_SIO1\_UART1\_TX**: SIO1/UART1 送信
- **DMACA\_TMRB8\_CMP\_MATCH**: TMRB8 コンペアー致
- **DMACA\_TMRB9\_CMP\_MATCH**: TMRB9 コンペアー致
- **DMACA\_TMRB0\_CAPTURE0**: TMRB0 キャプチャ 0 割り込み
- **DMACA\_TMRB4\_CAPTURE0**: TMRB4 キャプチャ 0 割り込み
- **DMACA\_TMRB4\_CAPTURE1**: TMRB4 キャプチャ 1 割り込み
- **DMACA\_TMRB5\_CAPTURE0**: TMRB5 キャプチャ 0 割り込み
- **DMACA\_TMRB5\_CAPTURE1**: TMRB5 キャプチャ 1 割り込み
- **DMACA\_NORMAL\_ADC**: 通常 A/D 変換終了
- **DMACA\_I2C0\_SIO0\_RX**: I2C0 受信
- **DMACA\_I2C0\_SIO0\_TX**: I2C0 送信
- **DMACA\_I2C1\_SIO1\_RX**: I2C1 受信
- **DMACA\_I2C1\_SIO1\_TX**: I2C1 送信

DMACA\_ReqNum

**A\_TxSrcPeriph**: 以下のいずれかのバースト要求番号を選択します。

- **DMACA\_SIO0\_UART0\_RX**: SIO0/UART0 受信
- **DMACA\_SIO0\_UART0\_TX**: SIO0/UART0 送信
- **DMACA\_SIO1\_UART1\_RX**: SIO1/UART1 受信
- **DMACA\_SIO1\_UART1\_TX**: SIO1/UART1 送信
- **DMACA\_TMRB8\_CMP\_MATCH**: TMRB8 コンペアー致
- **DMACA\_TMRB9\_CMP\_MATCH**: TMRB9 コンペアー致
- **DMACA\_TMRB0\_CAPTURE0**: TMRB0 キャプチャ 0 割り込み
- **DMACA\_TMRB4\_CAPTURE0**: TMRB4 キャプチャ 0 割り込み
- **DMACA\_TMRB4\_CAPTURE1**: TMRB4 キャプチャ 1 割り込み
- **DMACA\_TMRB5\_CAPTURE0**: TMRB5 キャプチャ 0 割り込み
- **DMACA\_TMRB5\_CAPTURE1**: TMRB5 キャプチャ 1 割り込み
- **DMACA\_NORMAL\_ADC**: 通常 A/D 変換終了
- **DMACA\_I2C0\_SIO0\_RX**: I2C0 受信
- **DMACA\_I2C0\_SIO0\_TX**: I2C0 送信
- **DMACA\_I2C1\_SIO1\_RX**: I2C1 受信
- **DMACA\_I2C1\_SIO1\_TX**: I2C1 送信

FunctionalState

**TxINT**: 以下から、転送割り込み状態を選択します。

- **EANBLE**: 転送割り込み許可
- **DISABLE**: 転送割り込み無効

## 5.2.4.2 DMAC\_INTReq

メンバ:

uint32\_t

**All**: DMAC 全チャネルの割り込み発生状態です

ビットフィールド

uint32\_t

**CH0\_INTReq** : 1 DMAC チャネル 0 の割り込み発生状態です。

uint32\_t

**CH1\_INTReq** : 1 DMAC チャネル 1 の割り込み発生状態です。

## 6. FC

### 6.1 概要

本デバイスは、フラッシュメモリを内蔵しています。フラッシュメモリのサイズは、256Kbyte です。

オンボードプログラミングにおいて、CPU はソフトウェアを実行し、flash メモリへのデータ書き込み / 削除を行います。データ書き込み / 削除は JEDEC 標準型コマンドに従って行います。また、Flash メモリをモニターするレジスタを提供し、各ブロックのプロテクション状態の表示、セキュリティ機能の設定を行います。

ブロック構成は、デバイスのデータシートを参照してください。

全ドライバ API は、アプリで使用する API 定義、マクロ、データタイプ、構造を格納する以下のファイルで構成されています。

```
\Libraries\TX03_Periph_Driver\src\tmpm365_fc.c  
\Libraries\TX03_Periph_Driver\inc\tmpm365_fc.h
```

### 6.2 API 関数

#### 6.2.1 関数一覧

- ◆ 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 関数の種類

関数は、主に以下の 4 種類に分かれています。

- 1) セキュリティ設定(Flash ROM データの読み出し、デバッグ):  
FC\_SetSecurityBit(), FC\_GetSecurityBit()
- 2) 自動動作状態およびプロテクト状態の取得:  
FC\_GetBusyState(), FC\_GetBlockProtectState()
- 3) プロテクトの設定:  
FC\_ProgramBlockProtectState(), FC\_EraseBlockProtectState()
- 4) 自動実行コマンド(書き込み、チップ消去、ブロック消去):  
FC\_WritePage(), FC\_EraseBlock(), FC\_EraseChip()

#### 6.2.3 関数仕様

##### 6.2.3.1 FC\_SetSecurityBit

セキュリティビットの設定

**関数のプロトタイプ宣言:**

void  
FC\_SetSecurityBit (FunctionalState **NewState**)

**引数:**

**NewState:** セキュリティビットを設定します。

- **DISABLE:** セキュリティ機能設定不可
- **ENABLE:** セキュリティビット設定可能

**機能:**

1) 書き込み/消去プロテクト用のすべてのプロテクトビット (PSRA<BLKn>)を”1”にします。

2) FCSECBIT<SECBIT>を”1”にします。

上記の 2 つの条件が成立すると、セキュリティ機能が有効になります。セキュリティ機能が有効な状態の制限内容は次の通りです。

- ROM 領域のデータの読み出し。
- JTAG/SW、トレースの通信

したがって、この API を使用する場合は、注意して実行してください。

FCSECBIT<SECBIT>はパワーオンリセットおよび低消費電力モードの STOP2 解除で初期化されます。

**戻り値:**

なし

## 6.2.3.2 FC\_GetSecurityBit

セキュリティビットの設定状態の取得

**関数のプロトタイプ宣言:**

FunctionalState  
FC\_GetSecurityBit(void)

**引数:**

なし

**機能:**

セキュリティビットの設定状態を取得します。

**戻り値:**

**DISABLE:** セキュリティ機能設定不可

**ENABLE:** セキュリティビット設定可能

## 6.2.3.3 FC\_GetBusyState

自動動作状態の取得

**関数のプロトタイプ宣言:**

WorkState  
FC\_GetBusyState(void)

引数:

なし。

機能:

自動動作状態を取得します。

戻り値:

**BUSY**: 自動動作中

**DONE**: 自動動作終了

## 6.2.3.4 FC\_GetBlockProtectState

ブロックのプロテクト状態の取得

関数のプロトタイプ宣言:

FunctionalState

FC\_GetBlockProtectState(uint8\_t **BlockNum**)

引数:

**BlockNum**: ブロック番号を選択します。

➤ FC\_BLOCK\_0 ~ FC\_BLOCK\_5

機能:

各ブロックのプロテクト状態を示します。プロテクト状態の時には、書き込み、消去ができません。

戻り値:

ブロックプロテクトの状態:

**DISABLE**: プロテクト状態ではない。

**ENABLE**: プロテクト状態

## 6.2.3.5 FC\_ProgramBlockProtectState

ブロックのプロテクト設定

関数のプロトタイプ宣言:

FC\_Result

FC\_ProgramProtectState(uint8\_t **BlockNum**)

引数:

**BlockNum**: ブロック番号を選択します。

➤ FC\_BLOCK\_0 ~ FC\_BLOCK\_5

機能:

ブロックプロテクトを設定します。プロテクト状態の時には、書き込み、消去ができません。

戻り値:

**FC\_SUCCESS**: プロテクト設定の成功

**FC\_ERROR\_PROTECTED**: プロテクト設定の失敗(すでにプロテクト済の場合は再度プロテクト設定を行いません)

**FC\_ERROR\_OVER\_TIME**: プロテクト設定の失敗(自動動作のタイムアウト)

## 6.2.3.6 FC\_EraseBlockProtectState

プロテクトの解除

関数のプロトタイプ宣言:

FC\_Result

FC\_EraseBlockProtectState(uint8\_t **BlockGroup**)

引数:

**BlockGroup**: ブロックグループを指定してください。

➤ **FC\_BLOCK\_GROUP\_1**: ブロック 4, 5

➤ **FC\_BLOCK\_GROUP\_0**: ブロック 0~3

機能:

プロテクトビットを"0"にすることでプロテクトを解除します。

戻り値:

**FC\_SUCCESS**: プロテクト解除の成功

**FC\_ERROR\_OVER\_TIME**: プロテクト解除の失敗(自動動作のタイムアウト)

## 6.2.3.7 FC\_WritePage

ページ単位の書き込み

関数のプロトタイプ宣言:

FC\_Result

FC\_WritePage(uint32\_t **PageAddr**, uint32\_t \* **Data**)

引数:

**PageAddr**: ページの開始アドレスを指定します。

**Data**: 書き込むデータバッファへのポインタを指定します。サイズは 128Byte です。

機能:

ページ書き込みを行います。

自動ページ書き込みは、既に消去された 1 ページにつき一回のみ実施されます。データ値が"1" または "0" のいずれかであっても、2 回以上書き込みを実施しないでください。

**補足**: あらかじめデータを消去せずに書き込みを行うと、デバイスに損傷を与える恐れがあります。

戻り値:

**FC\_SUCCESS**: 書き込み成功

**FC\_ERROR\_PROTECTED**: 書き込み失敗(ブロックにプロテクトが設定されている)

**FC\_ERROR\_OVER\_TIME**: 書き込みの失敗(自動動作のタイムアウト)

## 6.2.3.8 FC\_EraseBlock

ブロック単位の消去

関数のプロトタイプ宣言:

FC\_Result

FC\_EraseBlock(uint32\_t **BlockAddr**)

引数:

**BlockAddr**: ブロック開始アドレスを指定してください。

機能:

ブロック単位の消去を行います。プロテクトされていないブロックに対してのみ消去を行います。

戻り値:

**FC\_SUCCESS**: 消去成功

**FC\_ERROR\_PROTECTED**: 消去失敗(ブロックにプロテクトが設定されている)

**FC\_ERROR\_OVER\_TIME**: 消去の失敗(自動動作のタイムアウト)

## 6.2.3.9 FC\_EraseChip

チップ消去

関数のプロトタイプ宣言:

FC\_Result

FC\_EraseChip(void)

引数:

なし。

機能:

チップ消去を行います。ブロックの一部にプロテクトが設定されている場合、そのブロックのデータは消去されません。

戻り値:

**FC\_SUCCESS**: チップ消去成功。ただしブロックの一部にプロテクトが設定されている場合、そのブロックのデータは消去されません。

**FC\_ERROR\_PROTECTED**: 消去失敗(すべてのブロックにプロテクトが設定されている)

**FC\_ERROR\_OVER\_TIME**: 消去の失敗(自動動作のタイムアウト)

## 6.2.4 データ構造

なし

## 7. GPIO

### 7.1 概要

本デバイスの汎用 I/O ポートは、入出力はビット単位で指定でき、入出力ポート機能の他に、内蔵する周辺機能に対する入出力端子としても使用されます。

GPIO ドライバ API は各ポートの設定機能を持ち、入出力、プルアップ、プルダウン、オープンドレイン、CMOS などを設定します。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。

/Libraries/TX03\_Periph\_Driver/src/tmpm365\_gpio.c

/Libraries/TX03\_Periph\_Driver/inc/tmpm365\_gpio.h

### 7.2 API 関数

#### 7.2.1 関数一覧

- ◆ 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 関数の種類

関数は、主に以下の 3 種類に分かれています。

- 1) 入出力ポートへの書き込み/読み出し:  
GPIO\_ReadData(), GPIO\_ReadDataBit(), GPIO\_WriteData(), GPIO\_WriteDataBit()
- 2) 入出力ポートの初期化と設定:  
GPIO\_SetOutput(), GPIO\_SetInput(), GPIO\_SetOutputEnableReg(),  
GPIO\_SetInputEnableReg(), GPIO\_SetPullUp(), GPIO\_SetPullDown(),  
GPIO\_SetOpenDrain(), GPIO\_Init()
- 3) その他:  
GPIO\_EnableFuncReg(), GPIO\_DisableFuncReg()



## 7.2.3 関数仕様

### 7.2.3.1 GPIO\_ReadData

DATA データレジスタの読み込み

関数のプロトタイプ宣言:

```
uint8_t  
GPIO_ReadData(GPIO_Port GPIO_x);
```

引数:

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA**: GPIO port A
- **GPIO\_PB**: GPIO port B
- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

機能:

DATA レジスタを読み込みます。

戻り値:

DATA レジスタの値

### 7.2.3.2 GPIO\_ReadDataBit

ビット単位での DATA レジスタの読み込み

関数のプロトタイプ宣言:

```
uint8_t  
GPIO_ReadDataBit(GPIO_Port GPIO_x,  
uint8_t Bit_x);
```

引数:

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA**: GPIO port A
- **GPIO\_PB**: GPIO port B
- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

**Bit\_x**: GPIO 端子を選択します。

- **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

**機能:**

ビット単位で DATA データレジスタを読み込みます。

**戻り値:**

GPIO 端子の値:

- **GPIO\_BIT\_VALUE\_0:** 0
- **GPIO\_BIT\_VALUE\_1:** 1

### 7.2.3.3 GPIO\_WriteData

DATA レジスタへの書き込み

**関数のプロトタイプ宣言:**

```
void  
GPIO_WriteData(GPIO_Port GPIO_x,  
                uint8_t Data);
```

**引数:**

**GPIO\_x:** GPIO ポートを選択します。

- **GPIO\_PA:** GPIO port A
- **GPIO\_PB:** GPIO port B
- **GPIO\_PC:** GPIO port C
- **GPIO\_PD:** GPIO port D
- **GPIO\_PE:** GPIO port E
- **GPIO\_PF:** GPIO port F
- **GPIO\_PG:** GPIO port G
- **GPIO\_PH:** GPIO port H
- **GPIO\_PI:** GPIO port I
- **GPIO\_PJ:** GPIO port J
- **GPIO\_PK:** GPIO port K

**Data:** DATA レジスタに書き込む値を設定します。

**機能:**

DATA レジスタへ指定された値を書き込みます。

**戻り値:**

なし

### 7.2.3.4 GPIO\_WriteDataBit

ビット単位での DATA レジスタの書き込み

**関数のプロトタイプ宣言:**

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

**引数:**

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA**: GPIO port A
- **GPIO\_PB**: GPIO port B
- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

**Bit\_x**: GPIO 端子を選択します。

- **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

**BitValue**: GPIO 端子値を選択します。

- **GPIO\_BIT\_VALUE\_0**: 0
- **GPIO\_BIT\_VALUE\_1**: 1

**機能:**

ビット単位で DATA データレジスタを書き込みます。

**戻り値:**

なし

## 7.2.3.5 GPIO\_Init

GPIO ポートの初期設定

**関数のプロトタイプ宣言:**

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

**引数:**

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA**: GPIO port A
- **GPIO\_PB**: GPIO port B
- **GPIO\_PC**: GPIO port C

- **GPIO\_PD:** GPIO port D
- **GPIO\_PE:** GPIO port E
- **GPIO\_PF:** GPIO port F
- **GPIO\_PG:** GPIO port G
- **GPIO\_PH:** GPIO port H
- **GPIO\_PI:** GPIO port I
- **GPIO\_PJ:** GPIO port J
- **GPIO\_PK:** GPIO port K

**Bit\_x:** GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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:** すべての GPIO 端子

**GPIO\_InitStruct:** GPIO 基本設定の構造体です。(詳細は"データ構造"を参照)

**機能:**

GPIO ポートを IO モード、プルアップ、プルダウン、オープンドレインポート、CMOS ポートなどの設定をおこないます。本 API は **GPIO\_SetOutput()**, **GPIO\_SetInput()**, **GPIO\_SetPullUP()**, **GPIO\_SetOpenDrain()**を実行します。

**戻り値:**

なし

## 7.2.3.6 GPIO\_SetOutput

出力ポートの設定

**関数のプロトタイプ宣言:**

```
void  
GPIO_SetOutput(GPIO_Port GPIO_x,  
                uint8_t Bit_x);
```

**引数:**

**GPIO\_x:** GPIO ポートを選択します。

- **GPIO\_PA:** GPIO port A
- **GPIO\_PB:** GPIO port B
- **GPIO\_PC:** GPIO port C
- **GPIO\_PD:** GPIO port D
- **GPIO\_PE:** GPIO port E
- **GPIO\_PF:** GPIO port F
- **GPIO\_PG:** GPIO port G
- **GPIO\_PH:** GPIO port H
- **GPIO\_PI:** GPIO port I
- **GPIO\_PJ:** GPIO port J
- **GPIO\_PK:** GPIO port K

**Bit\_x:** GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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:** すべての GPIO 端子

**機能:**

出力ポートに設定します。

**戻り値:**

なし

## 7.2.3.7 GPIO\_SetInput

入力ポートの設定

**関数のプロトタイプ宣言:**

```
void  
GPIO_SetInput(GPIO_Port GPIO_x,  
               uint8_t Bit_x);
```

**引数:**

**GPIO\_x:** GPIO ポートを選択します。

- **GPIO\_PA:** GPIO port A
- **GPIO\_PB:** GPIO port B
- **GPIO\_PC:** GPIO port C
- **GPIO\_PD:** GPIO port D
- **GPIO\_PE:** GPIO port E
- **GPIO\_PF:** GPIO port F
- **GPIO\_PG:** GPIO port G
- **GPIO\_PH:** GPIO port H
- **GPIO\_PI:** GPIO port I
- **GPIO\_PJ:** GPIO port J
- **GPIO\_PK:** GPIO port K

**Bit\_x:** GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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:** すべての GPIO 端子

**機能:**

入力ポートに設定します。

**戻り値:**

なし

## 7.2.3.8 GPIO\_SetOutputEnableReg

出力ポートの許可/禁止設定

関数のプロトタイプ宣言:

```
void  
GPIO_SetOutputEnableReg(GPIO_Port GPIO_x,  
                        uint8_t Bit_x,  
                        FunctionalState NewState);
```

引数:

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA**: GPIO port A
- **GPIO\_PB**: GPIO port B
- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

**Bit\_x**: GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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**: すべての GPIO 端子

**NewState**:

- **ENABLE**: 出力許可
- **DISABLE**: 出力禁止

機能:

GPIO 端子出力の許可/禁止を設定します。

**NewState** が **ENABLE** の時、出力許可。

**NewState** が **DISABLE** の時、出力禁止。

戻り値:

なし

## 7.2.3.9 GPIO\_SetInputEnableReg

入力ポートの許可/禁止設定

## 関数のプロトタイプ宣言:

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

## 引数:

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA**: GPIO port A
- **GPIO\_PB**: GPIO port B
- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

**Bit\_x**: GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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**: すべての GPIO 端子

## **NewState**:

- **ENABLE**: 入力許可
- **DISABLE**: 入力禁止

## 機能:

GPIO 端子入力の許可/禁止を設定します。**NewState** が **ENABLE** の時、入力を許可します。**NewState** が **DISABLE** の時、入力を禁止します。

## 戻り値:

なし

### 7.2.3.10 GPIO\_SetPullUp

プルアップポートの設定

## 関数のプロトタイプ宣言:

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

## 引数:

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PA:** GPIO port A
- **GPIO\_PB:** GPIO port B
- **GPIO\_PC:** GPIO port C
- **GPIO\_PD:** GPIO port D
- **GPIO\_PE:** GPIO port E
- **GPIO\_PF:** GPIO port F
- **GPIO\_PG:** GPIO port G
- **GPIO\_PH:** GPIO port H
- **GPIO\_PI:** GPIO port I
- **GPIO\_PJ:** GPIO port J
- **GPIO\_PK:** GPIO port K

**Bit\_x:** GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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:** すべての GPIO 端子

**NewState:**

- **ENABLE:** プルアップ許可
- **DISABLE:** プルアップ禁止

**機能:**

GPIO 端子プルアップの許可/禁止を設定します。

**NewState** が **ENABLE** の時、プルアップを許可し、**NewState** が **DISABLE** の時、プルアップを禁止します。

**戻り値:**

なし

## 7.2.3.11 GPIO\_SetPullDown

プルダウンポートの設定

**関数のプロトタイプ宣言:**

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

**引数:**

**GPIO\_x:** GPIO ポートを選択します。

- **GPIO\_PI :** GPIO port I.

**Bit\_x:** GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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:** すべての GPIO 端子

**NewState:**

- **ENABLE:** プルダウン許可
- **DISABLE:** プルダウン禁止

**機能:**

GPIO 端子プルダウンの許可/禁止を設定します。**NewState** が **ENABLE** の時、プルダウンを許可します。**NewState** が **DISABLE** の時、プルダウンを禁止します。

**戻り値:**

なし

## 7.2.3.12 GPIO\_SetOpenDrain

CMOS/オープンドレインポートの設定

**関数のプロトタイプ宣言:**

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

**引数:**

**GPIO\_x:** GPIO ポートを選択します。

- **GPIO\_PA:** GPIO port A
- **GPIO\_PB:** GPIO port B
- **GPIO\_PC:** GPIO port C
- **GPIO\_PD:** GPIO port D
- **GPIO\_PE:** GPIO port E
- **GPIO\_PF:** GPIO port F
- **GPIO\_PG:** GPIO port G
- **GPIO\_PH:** GPIO port H
- **GPIO\_PI:** GPIO port I

**Bit\_x:** GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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:** すべての GPIO 端子

**NewState:**

- **ENABLE:** オープンドレイン許可
- **DISABLE:** CMOS 許可

**機能:**

GPIO 端子 CMOS/オープンドレインの許可/禁止を設定します。**NewState** が **ENABLE** の時、オープンドレインを許可します。**NewState** が **DISABLE** の時、CMOS を許可します。

**戻り値:**

なし

## 7.2.3.13 GPIO\_EnableFuncReg

機能ポートの有効設定

**関数のプロトタイプ宣言:**

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

**引数:**

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

**FuncReg\_x**: GPIO 機能レジスタの番号を選択します。

- **GPIO\_FUNC\_REG\_1**: GPIO 機能レジスタ 1
- **GPIO\_FUNC\_REG\_2**: GPIO 機能レジスタ 2
- **GPIO\_FUNC\_REG\_3**: GPIO 機能レジスタ 3
- **GPIO\_FUNC\_REG\_4**: GPIO 機能レジスタ 4

**Bit\_x**: GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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

**機能:**

GPIO 端子の機能を有効に設定します。

**戻り値:**

なし

## 7.2.3.14 GPIO\_DisableFuncReg

機能ポートの無効設定

関数のプロトタイプ宣言:

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

引数:

**GPIO\_x**: GPIO ポートを選択します。

- **GPIO\_PC**: GPIO port C
- **GPIO\_PD**: GPIO port D
- **GPIO\_PE**: GPIO port E
- **GPIO\_PF**: GPIO port F
- **GPIO\_PG**: GPIO port G
- **GPIO\_PH**: GPIO port H
- **GPIO\_PI**: GPIO port I
- **GPIO\_PJ**: GPIO port J
- **GPIO\_PK**: GPIO port K

**FuncReg\_x**: GPIO 機能レジスタの番号を選択します。

- **GPIO\_FUNC\_REG\_1**: GPIO 機能レジスタ 1
- **GPIO\_FUNC\_REG\_2**: GPIO 機能レジスタ 2
- **GPIO\_FUNC\_REG\_3**: GPIO 機能レジスタ 3
- **GPIO\_FUNC\_REG\_4**: GPIO 機能レジスタ 4

**Bit\_x**: GPIO 端子を選択します。有効ビットの組み合わせが可能です。

- **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

機能:

GPIO 端子の機能を無効に設定します。

戻り値:

なし

## 7.2.4 データ構造

### 7.2.4.1 GPIO\_InitTypeDef

メンバ:

uint8\_t

**IOMode**   ポートの入出力設定

- **GPIO\_INPUT**: 入力ポートに設定
- **GPIO\_OUTPUT**: 出力ポートに設定
- **GPIO\_IO\_MODE\_NONE**: 入出力モードを変更しない

uint8\_t

**PullUp** プルアップポートの許可/禁止設定

- **GPIO\_PULLUP\_ENABLE:** プルアップ許可
- **GPIO\_PULLUP\_DISABLE:** プルアップ禁止
- **GPIO\_PULLUP\_NONE:** プルアップ機能が無い、または設定変更しない

uint8\_t

**OpenDrain** オープンドレインポート/CMOSポートの設定

- **GPIO\_OPEN\_DRAIN\_ENABLE:** オープンドレインポートに設定
- **GPIO\_OPEN\_DRAIN\_DISABLE:** CMOSポートに設定
- **GPIO\_OPEN\_DRAIN\_NONE:** オープンドレイン機能がない、または設定変更しない

uint8\_t

**PullDown** プルダウンポートの許可/禁止設定

- **GPIO\_PULLDOWN\_ENABLE:** プルダウン許可
- **GPIO\_PULLDOWN\_DISABLE:** プルダウン禁止
- **GPIO\_PULLDOWN\_NONE:** プルダウン機能がない、または設定変更しない

## 8. SBI

### 8.1 概要

本デバイスはシリアルバスインターフェースチャンネルを有し、各チャンネルはマルチマスタが可能。I2C バスで動作可能です。

I2C バスモードでは、SCL および SDA を通して外部デバイスと接続されます。

SBI チャンネルによりデータをフリーデータフォーマットで転送できます。フリーデータフォーマットでは、マスタモード時は送信、スレーブモード時は受信になります。

SBI ドライバ API 関数は、SBI チャンネルの自己アドレス、クロック分周、ACK クロック生成等の設定、I2C の開始・終了条件のデータ転送、データ受信・送信の制御、状態復帰、SBI チャンネルモードの表示などの機能の設定を行う関数セットです。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。

/Libraries/TX03\_Periph\_Driver/src/tmpm365\_sbi.c

/Libraries/TX03\_Periph\_Driver/inc/tmpm365\_sbi.h

### 8.2 API 関数

#### 8.2.1 関数一覧

- ◆ void SBI\_Enable(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_Disable(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_SetI2CACK(TSB\_SBI\_TypeDef\* **SBIx**, FunctionalState **NewState**);
- ◆ void SBI\_InitI2C(TSB\_SBI\_TypeDef\* **SBIx**, SBI\_InitI2CTypeDef\* **InitI2CStruct**);
- ◆ void SBI\_SetI2CBitNum(TSB\_SBI\_TypeDef\* **SBIx**, uint32\_t **I2CBitNum**);
- ◆ void SBI\_SWReset(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_ClearI2CINTReq(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_GenerateI2Cstart(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_GenerateI2Cstop(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ SBI\_I2CState SBI\_GetI2CState(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_SetIdleMode(TSB\_SBI\_TypeDef\* **SBIx**, FunctionalState **NewState**);
- ◆ void SBI\_SetSendData(TSB\_SBI\_TypeDef\* **SBIx**, uint32\_t **Data**);
- ◆ uint32\_t SBI\_GetReceiveData(TSB\_SBI\_TypeDef\* **SBIx**);
- ◆ void SBI\_SetI2CFreeDataMode(TSB\_SBI\_TypeDef\* **SBIx**, FunctionalState **NewState**);

#### 8.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

- 1) 共通機能の設定:  
SBI\_Enable(), SBI\_Disable(), SBI\_SetI2CACK(), SBI\_SetI2CBitNum(), SBI\_InitI2C()
- 2) 転送制御:  
SBI\_ClearI2CINTReq(), SBI\_GenerateI2Cstart(), SBI\_GenerateI2Cstop(),  
SBI\_IsI2ClastRxBitSet(), SBI\_GetReceiveData()
- 3) ステータス確認:  
SBI\_GetI2CState()

4) その他:

SBI\_SWReset(), SBI\_SetIdleMode(), SBI\_EnableI2CfreeDataMode()

## 8.2.3 関数仕様

補足: 下記の全 API において、パラメータ“TSB\_SBI\_TypeDef\* **SBIx**”は以下のいずれかを選択してください。

TSB\_SBI0, TSB\_SBI1

### 8.2.3.1 SBI\_Enable

SBI 動作の許可

関数のプロトタイプ宣言:

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

引数:

**SBIx**: SBI チャンネルを指定します。

機能:

SBI 動作を有効にします。

戻り値:

なし

### 8.2.3.2 SBI\_Disable

SBI 動作の禁止

関数のプロトタイプ宣言:

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

引数:

**SBIx**: SBI チャンネルを指定します。

機能:

SBI 動作を無効にします。

戻り値:

なし

### 8.2.3.3 SBI\_SetI2CACK

I2C バスモードにおける ACK 選択

関数のプロトタイプ宣言:

void  
SBI\_SetI2CACK(TSB\_SBI\_TypeDef\* **SBIx**,  
FunctionalState **NewState**)

引数:

**SBIx**: SBI チャンネルを指定します。

**NewState**: ACK の発生有無を選択します。

- **ENABLE**: 発生する。
- **DISABLE**: 発生しない。

**機能**:

I2C 通信のアクノリッジメントクロック(ACK)のためのクロックを発生する/発生しないを選択します。**NewState**を **ENABLE** にすると ACK クロックを発生し、**DISABLE** にすると ACK クロックを発生しません。

**戻り値**:

なし

## 8.2.3.4 SBI\_InitI2C

I2C バスモードにおける通信の初期化

**関数のプロトタイプ宣言**:

```
void  
SBI_InitI2C(TSB_SBI_TypeDef* SBIx,  
             SBI_InitI2CTypeDef* InitI2CStruct)
```

**引数**:

**SBIx**: SBI チャンネルを指定します。

**InitI2CStruct**: SBI に関する構造体です。(詳細は"データ構造"を参照)

**機能**:

I2C バスアドレス、転送ビット数、出力クロックの周波数選択、ACK クロック生成、I2C 転送モードの初期化を行います。

**戻り値**:

なし

## 8.2.3.5 SBI\_SetI2CBitNum

I2C バスモードにおける転送ビット数の選択

**関数のプロトタイプ宣言**:

```
void  
SBI_SetI2CBitNum(TSB_SBI_TypeDef* SBIx,  
                  uint32_t I2CBitNum)
```

**引数**:

**SBIx**: SBI チャンネルを指定します。

**I2CBitNum**: 転送ビット数(1~8)を選択します。

- **SBI\_I2C\_DATA\_LEN\_8**: データ長 8
- **SBI\_I2C\_DATA\_LEN\_1**: データ長 1
- **SBI\_I2C\_DATA\_LEN\_2**: データ長 2
- **SBI\_I2C\_DATA\_LEN\_3**: データ長 3
- **SBI\_I2C\_DATA\_LEN\_4**: データ長 4
- **SBI\_I2C\_DATA\_LEN\_5**: データ長 5

- **SBI\_I2C\_DATA\_LEN\_6:** データ長 6
- **SBI\_I2C\_DATA\_LEN\_7:** データ長 7

**機能:**

転送ビット数を選択します。

**戻り値:**

なし

## 8.2.3.6 SBI\_SWReset

ソフトウェアリセットの発生

**関数のプロトタイプ宣言:**

```
void  
SBI_SWReset(TSB_SBI_TypeDef* SBIx)
```

**引数:**

**SBIx:** SBI チャンネルを指定します。

**機能:**

シリアルバスインターフェース回路を初期化するリセット信号を発生します。リセット後、すべての制御レジスタやステータスフラグはリセット後の値に初期化されます。

**戻り値:**

なし

## 8.2.3.7 SBI\_ClearI2CINTReq

I2C バスモードにおける INTSBIx 割り込み要求解除

**関数のプロトタイプ宣言:**

```
void  
SBI_ClearI2CINTReq(TSB_SBI_TypeDef* SBIx)
```

**引数:**

**SBIx:** SBI チャンネルを指定します。

**機能:**

SBI 割り込み要求を解除します。

**戻り値:**

なし

## 8.2.3.8 SBI\_Generatel2CStart

I2C バスモードにおけるスタート状態の発生

**関数のプロトタイプ宣言:**

```
void  
SBI_Generatel2CStart(TSB_SBI_TypeDef* SBIx)
```



引数:

**SBIx**: SBI チャンネルを指定します。

機能:

I2C バスモードをマスタにし、I2C バスにスタートコンディションを出力します。

戻り値:

なし

## 8.2.3.9 SBI\_Generatel2CStop

I2C バスモードにおけるストップ状態の発生

関数のプロトタイプ宣言:

void

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

引数:

**SBIx**: SBI チャンネルを指定します。

機能:

I2C バスモードをマスタにし、I2C バスにストップコンディションを出力します。

戻り値:

なし

## 8.2.3.10 SBI\_GetI2CState

I2C バスモードにおける SBI チャンネルの状態の読み込み

関数のプロトタイプ宣言:

SBI\_I2CState

SBI\_GetI2CState(TSB\_SBI\_TypeDef\* **SBIx**)

引数:

**SBIx**: SBI チャンネルを指定します。

機能:

I2C バスモード中の SBI チャンネルの状態を読み込みます。SBI 割り込みの ISR で本関数をコールし、SBI チャンネルの状態によってプロセスを変更します。

戻り値:

I2C モードでの SBI チャンネルの状態

## 8.2.3.11 SBI\_SetIdleMode

IDLE モード時の動作の許可/禁止

関数のプロトタイプ宣言:

void

SBI\_SetIdleMode(TSB\_SBI\_TypeDef\* **SBIx**,  
FunctionalState **NewState**)

引数:

**SBIx**: SBI チャンネルを指定します。

**NewState**: システムが idle モードの時の動作を指定します。

➤ **ENABLE**: 許可。

➤ **DISABLE**: 禁止。

機能:

**NewState** が **ENABLE** の場合 IDLE モードに遷移しても SBI チャンネルは動作します。

**DISABLE** を選択すると IDLE モード時に禁止されます。

戻り値:

なし

## 8.2.3.12 SBI\_SetSendData

データ送信

関数のプロトタイプ宣言:

void

SBI\_SetSendData(TSB\_SBI\_TypeDef\* **SBIx**,  
uint32\_t **Data**)

引数:

**SBIx**: SBI チャンネルを指定します。

**Data**: 送信データ。(最大値は 0xFF です)

機能:

設定データを送信します。**SBI\_GenerateI2Cstart()**の実行によりスタートコンディションを出力後、または ACK (通常は SBI 割り込みにより発生)受信後、データを送信します。

戻り値:

なし

## 8.2.3.13 SBI\_GetReceiveData

データ受信

関数のプロトタイプ宣言:

uint32\_t

SBI\_GetReceiveData(TSB\_SBI\_TypeDef\* **SBIx**)

引数:

**SBIx**: SBI チャンネルを指定します。

機能:

データを受信します。**SBI\_GenerateI2Cstart()**の実行によりスタートコンディションを出力後、または ACK (通常は SBI 割り込みにより発生)受信後、データを受信します。

戻り値:  
受信データ

## 8.2.3.14 SBI\_SetI2CFreeDataMode

Set SBI channel working in I2C free data mode.  
アドレス認識モードの指定

関数のプロトタイプ宣言:

```
void  
SBI_SetI2CFreeDataMode(TSB_SBI_TypeDef* SBIx,  
                        FunctionalState NewState)
```

引数:

**SBIx**: SBI チャンネルを指定します。

**NewState**: アドレス認識モードを指定します。

- **ENABLE**: スレーブアドレスを認識しない。(フリーデータフォーマット)
- **DISABLE**: スレーブアドレスを認識する。

機能:

I2C モードにおけるデータフォーマットをフリーデータフォーマットにします。フリーデータフォーマットの場合、スレーブデバイスがデータ受信中にマスターデバイスは常にデータ送信を行います。転送データをノーマル I2C フォーマットにする場合は **SBI\_InitI2C()** をコールしてください。

戻り値:  
なし

## 8.2.4 データ構造

### 8.2.4.1 SBI\_InitI2CTypeDef

メンバ:

uint32\_t  
**I2CSelfAddr**: I2C モードにおけるスレーブアドレスを指定します。(0x01~0xFE)

uint32\_t  
**I2CDataLen**: I2C モードにおける SBI チャンネルの転送ビット数を指定します。

- **SBI\_I2C\_DATA\_LEN\_8**: データ長 8
- **SBI\_I2C\_DATA\_LEN\_1**: データ長 1
- **SBI\_I2C\_DATA\_LEN\_2**: データ長 2
- **SBI\_I2C\_DATA\_LEN\_3**: データ長 3
- **SBI\_I2C\_DATA\_LEN\_4**: データ長 4
- **SBI\_I2C\_DATA\_LEN\_5**: データ長 5
- **SBI\_I2C\_DATA\_LEN\_6**: データ長 6
- **SBI\_I2C\_DATA\_LEN\_7**: データ長 7

uint32\_t  
**I2CClkDiv**: I2C 転送のソースクロックを選択します。

- **SBI\_I2C\_CLK\_DIV\_104**: fsys/104
- **SBI\_I2C\_CLK\_DIV\_136**: fsys/136

- **SBI\_I2C\_CLK\_DIV\_200:** fsys/200
- **SBI\_I2C\_CLK\_DIV\_328:** fsys/328
- **SBI\_I2C\_CLK\_DIV\_584:** fsys/584
- **SBI\_I2C\_CLK\_DIV\_1096:** fsys/1096
- **SBI\_I2C\_CLK\_DIV\_2120:** fsys/2120

FunctionalState

**I2CACKState:** ACK の有効/無効を選択します。

- **ENABLE:** 有効。
- **DISABLE:** 無効。

## 8.2.4.2 SBI\_I2CState

メンバ:

uint32\_t

**All:** I2C モードの全ての状態

ビットフィールド:

uint32\_t

**LastRxBit:** 最終受信ビットモニタ

uint32\_t

**GeneralCall:** ゼネラルコール検出モニタ

uint32\_t

**SlaveAddrMatch:** スレーブアドレス一致モニタ

uint32\_t

**ArbitrationLost:** アービトレーションロスト検出モニタ

uint32\_t

**INTReq:** 割り込み要求状態モニタ

uint32\_t

**BusState:** バス状態モニタ

uint32\_t

**TRx:** 送信/受信選択状態モニタ

uint32\_t

**MasterSlave:** マスタ/スレーブ選択状態モニタ

## 9. TMRB

### 9.1 概要

本デバイスは、10 チャンネルの多機能 16 ビットタイマ/ イベントカウンタ (TMRB0 ~ TMRB9)を内蔵しています。各チャンネルは下記モードで動作します。

- 16 ビットインタバルタイマモード
- 16 ビットイベントカウンタモード
- 16 ビットプログラマブル矩形波出力 (PPG) モード
- タイマ同期モード(各 4 チャンネルの出力設定可能)

また、キャプチャ機能を利用することで、次のような用途に使用することができます。

- 周波数測定
- パルス幅測定
- 時間差測定

本ドライバは、クロック分割、サイクル、デューティ期間、キャプチャタイミング、フリップフロップの設定など各チャンネルの設定を行う関数セットです。また、アップカウンタ、フリップフロップ出力の制御など動作状態の制御、割り込み要因、キャプチャレジスタ値の取得など、ステータスの表示も行います。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。

/Libraries/TX03\_Periph\_Driver/src/tmpm365\_tmr.c  
/Libraries/TX03\_Periph\_Driver/inc/tmpm365\_tmr.h

### 9.2 API 関数

#### 9.2.1 関数一覧

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

- ◆ void TMRB\_SetDoubleBuf(TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**);
- ◆ void TMRB\_SetExtStartTrg(TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**,  
uint8\_t **TrgMode**);
- ◆ void TMRB\_SetClkInCoreHalt(TSB\_TB\_TypeDef \* **TBx**, uint8\_t **ClkState**);
- ◆ void TMRB\_SetExtInput(TSB\_TB\_TypeDef \* **TBx**);
- ◆ void TMRB\_SetDMAReq(TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**,  
uint8\_t **DMAReq**);
- ◆ void TMRB\_SetFT0Sel (TSB\_TB\_TypeDef \* **TBx**, FunctionalState **NewState**);

## 9.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

- 1) 各タイマの設定:  
TMRB\_Enable(), TMRB\_Disable(), TMRB\_Init(), TMRB\_SetRunState(),  
TMRB\_ChangeLeadingTiming(), TMRB\_ChangeTrailingTiming()
- 2) キャプチャ機能の設定:  
TMRB\_SetCaptureTiming(), TMRB\_ExecuteSWCapture()
- 3) ステータスの確認:  
TMRB\_GetINTFactor(), TMRB\_GetUpCntValue(), TMRB\_GetCaptureValue()
- 4) その他:  
TMRB\_SetFlipFlop(), TMRB\_SetINTMask(), TMRB\_SetIdleMode(),  
TMRB\_SetSyncMode(), TMRB\_SetDoubleBuf(), TMRB\_SetExtStartTrg(),  
TMRB\_SetClkInCoreHalt (), TMRB\_SetExtInput(), TMRB\_SetDMAReq() ,  
TMRB\_SetFT0Sel()

## 9.2.3 関数仕様

**補足:** 引数に記述されている “TSB\_TB\_TypeDef\* **TBx**” は特に記載の無い限り以下から選択してください。

**TSB\_TB0, TSB\_TB1, TSB\_TB2, TSB\_TB3, TSB\_TB4, TSB\_TB5, TSB\_TB6,  
TSB\_TB7, TSB\_TB8, TSB\_TB9**

### 9.2.3.1 TMRB\_Enable

TMRB 機能の許可

**関数のプロトタイプ宣言:**

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

**引数:**

**TBx:** TMRB チャンネルを指定します。

**機能:**

TMRB 機能を有効にします。

**戻り値:**

なし

### 9.2.3.2 TMRB\_Disable

TMRB 機能の禁止

**関数のプロトタイプ宣言:**

```
void  
TMRB_Disable(TSB_TB_TypeDef* TBx)
```

引数:

**TBx**: TMRB チャンネルを指定します。

機能:

TMRB 機能を無効にします。

戻り値:

なし

### 9.2.3.3 TMRB\_SetRunState

カウンタ動作の設定

関数のプロトタイプ宣言:

```
void  
TMRB_SetRunState(TSB_TB_TypeDef* TBx,  
uint32_t Cmd)
```

引数:

**TBx**: TMRB チャンネルを指定します。

**Cmd**: カウンタ動作を選択します。

- **TMRB\_RUN**: カウント
- **TMRB\_STOP**: 停止&クリア

機能:

**Cmd** が **TMRB\_RUN** の場合、アップカウンタがカウントを開始します。

**Cmd** が **TMRB\_STOP** の場合、アップカウンタはカウントを停止し、同時にカウンタをクリアします。

戻り値:

なし

### 9.2.3.4 TMRB\_Init

TMRB チャンネルの初期化

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**InitStruct**: TMRB に関する構造体です。(詳細は"データ構造"を参照)

機能:

カウンティングモード、クロック分周、アップカウンタ設定、サイクル、デューティー期間の初期設定を行います。

戻り値:

なし

## 9.2.3.5 TMRB\_SetCaptureTiming

キャプチャタイミングの設定

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**CaptureTiming**: キャプチャタイミングを選択します。

- **TMRB\_DISABLE\_CAPTURE**: キャプチャ機能を無効にします。
- **TMRB\_CAPTURE\_IN\_RISING**: TBxIN0↑ TBxIN1↑
- **TMRB\_CAPTURE\_IN\_RISING\_FALLING**: TBxIN0↑ TBxIN0↓
- **TMRB\_CAPTURE\_OUTPUT\_EDGE**: TBxFF0↑ TBxFF0↓

機能:

**CaptureTiming** が **TMRB\_CAPTURE\_IN\_RISING** の場合、TBxIN0 端子入力の立ち上がりでキャプチャレジスタ 0 (TBxCP0) にカウント値を取り込み、TBxIN1 端子入力の立ち上がりでキャプチャレジスタ 1 (TBxCP1) にカウント値を取り込みます。

**CaptureTiming** が **TMRB\_CAPTURE\_IN\_RISING\_FALLING** の場合、TBxIN0 端子入力の立ち上がりでキャプチャレジスタ 0 (TBxCP0) にカウント値を取り込み、TBxIN0 端子入力の立ち下がりでキャプチャレジスタ 1 (TBxCP1) にカウント値を取り込みます。

**CaptureTiming** が **TMRB\_CAPTURE\_OUTPUT\_EDGE** の場合、TBxFF0 の立ち上がりでキャプチャレジスタ 0 (TBxCP0) にカウント値を取り込み、TBxFF0 端子入力の立ち下がりでキャプチャレジスタ 1 (TBxCP1) にカウント値を取り込みます。

TMRB7, TMRB8, TMRB9 のフリップフロップ出力を他のチャンネルのキャプチャトリガとして使用できます。

TMRB0~1: TB7OUT

TMRB2~3: TB8OUT

TMRB4~6: TB9OUT

戻り値:

なし

## 9.2.3.6 TMRB\_SetFlipFlop

フリップフロップ機能の設定

関数のプロトタイプ宣言:

```
void
```



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

**引数:**

**TBx:** TMRB チャンネルを指定します。

**FFStruct:** TMRB のフリップフロップ機能に関する構造体です。(詳細は"データ構造"を参照)

**機能:**

フリップフロップ出力変更のタイミングを設定します。また出力レベルも設定できます。

**戻り値:**

なし

## 9.2.3.7 TMRB\_GetINTFactor

割り込み要因の取得。

**関数のプロトタイプ宣言:**

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

**引数:**

**TBx:** TMRB チャンネルを指定します。

**機能:**

割り込み要因を取得します。

**戻り値:**

TMRB の割り込み要因:

**MatchLeadingTiming** (Bit0): 一致フラグ(TBxRG0)

**MatchTrailingTiming** (Bit1): 一致フラグ(TBxRG1)

**OverFlow** (Bit2): オーバーフローフラグ

**補足:**

異なる割り込み要因を処理する場合は、以下のように記述してください。

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

## 9.2.3.8 TMRB\_SetINTMask

割り込みマスク要因の設定

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**INTMask**: マスクする割り込みを選択します。

- **TMRB\_MASK\_MATCH\_TRAILING\_INT**: 一致フラグ(TBxRG0)
- **TMRB\_MASK\_MATCH\_LEADING\_INT**: 一致フラグ(TBxRG1)
- **TMRB\_MASK\_OVERFLOW\_INT**: オーバーフロー割り込み。
- **TMRB\_NO\_INT\_MASK**: マスクしない。

機能:

**TMRB\_MASK\_MATCH\_TRAILING\_INT** 選択時、アップカウンタ値と TBxRG1 が一致した場合、割り込みは発生しません。

**TMRB\_MASK\_MATCH\_LEADING\_INT** 選択時、アップカウンタ値と TBxRG0 が一致した場合、割り込みは発生しません。

**TMRB\_MASK\_OVERFLOW\_INT** 選択時、オーバーフロー発生時の割り込みは発生しません。

**TMRB\_NO\_INT\_MASK** 選択時、割り込みマスクはすべてクリアされます。

戻り値:

なし

## 9.2.3.9 TMRB\_ChangeLeadingTiming

デューティの設定

関数のプロトタイプ宣言:

```
void  
TMRB_ChangeLeadingTiming(TSB_TB_TypeDef* TBx,  
                          uint32_t LeadingTiming)
```

引数:

**TBx**: TMRB チャンネルを指定します。

**LeadingTiming**: デューティ値を設定します。最大値は 0xFFFF です。

機能:

デューティを設定します。実際のデューティのインターバルは、CGの校正と **CkDiv**(詳細は"データ構造"を参照) の値によります。

戻り値:

なし。

補足:

**LeadingTiming** は **TrailingTiming** を超えることはできません。

## 9.2.3.10 TMRB\_ChangeTrailingTiming

周期の設定

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**TrailingTiming**: 周期を設定します。最大は 0xFFFF です。

機能:

周期を設定します。実際の周期は、CG の校正と **ClkDiv**(詳細は"データ構造"を参照) の値によります。

戻り値:

なし。

補足:

**TrailingTiming** は **LeadingTiming** より小さくすることはできません。

## 9.2.3.11 TMRB\_GetUpCntValue

アップカウンタ値の読み込み

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

機能:

アップカウンタ値の読み込みを行います。

戻り値:

アップカウンタ値

## 9.2.3.12 TMRB\_GetCaptureValue

キャプチャレジスタの読み込み

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**CapReg:** キャプチャレジスタを選択します。

- **TMRB\_CAPTURE\_0:** キャプチャレジスタ 0
- **TMRB\_CAPTURE\_1:** キャプチャレジスタ 1

**機能:**

**CapReg** が **TMRB\_CAPTURE\_0** の場合、キャプチャレジスタ 0 の値を読み込み、**CapReg** が **TMRB\_CAPTURE\_1** の場合、キャプチャレジスタ 1 の値を読み込みます。

**戻り値:**

キャプチャされた値

### 9.2.3.13 TMRB\_ExecuteSWCapture

ソフトウェアキャプチャの実行

**関数のプロトタイプ宣言:**

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

**引数:**

**TBx:** TMRB チャンネルを指定します。

**機能:**

キャプチャレジスタ 0 (TBxCP0)にカウント値を取り込みます。

**戻り値:**

なし

### 9.2.3.14 TMRB\_SetIdleMode

IDLE 時の動作設定

**関数のプロトタイプ宣言:**

void  
TMRB\_SetIdleMode(TSB\_TB\_TypeDef\* **TBx**,  
FunctionalState **NewState**)

**引数:**

**TBx:** TMRB チャンネルを指定します。

**NewState:** IDLE 時の動作を指定します。

- **ENABLE:** 動作
- **DISABLE:** 停止

**機能:**

**NewState** が **ENABLE** の場合、IDLE 時でも TMRB チャンネルは動作します。**DISABLE** の場合、IDLE 時は動作を停止します。

**戻り値:**

なし

## 9.2.3.15 TMRB\_SetSyncMode

同期モードの切り替え

関数のプロトタイプ宣言:

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

引数:

**TBx**: 以下から TMRB チャンネルを以下から選択します。

**TSB\_TB0, TSB\_TB1, TSB\_TB2, TSB\_TB3, TSB\_TB4, TSB\_TB5,**  
**TSB\_TB6, TSB\_TB7.**

**NewState**: 同期モードを切り替えます。

- **ENABLE**: 同期動作
- **DISABLE**: 個別動作(チャンネル毎)

機能:

TMRB0～TMRB3 を同期モードに設定すると、TMRB0 のスタートに同期して動作がスタートし、TMRB4～TMRB7 を同期モードに設定すると、TMRB4 のスタートに同期して動作がスタートします。

戻り値:

なし

補足:

同期モードを使用するために、TMRB0、TMRB4 のカウントを開始する前に、**TMRB\_SetRunState()** によって TMRB0～TMRB3、TMRB4～TMRB7 をスタートしてください。

## 9.2.3.16 TMRB\_SetDoubleBuf

ダブルバッファ動作の制御

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**NewState**: ダブルバッファの有効/無効を選択します。

- **ENABLE**: 許可。
- **DISABLE**: 禁止。

機能:

ダブルバッファ動作の許可/禁止を設定します。

戻り値:

なし

## 9.2.3.17 TMRB\_SetExtStartTrg

外部トリガの設定

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**NewState**: カウントスタート方法を選択します。

- **ENABLE**: 外部トリガ
- **DISABLE**: ソフトスタート

**TrgMode**: 外部トリガのアクティブエッジを選択します。

- **TMRB\_TRG\_EDGE\_RISING**: 立ち上がりエッジ
- **TMRB\_TRG\_EDGE\_FALLING**: 立ち下りエッジ

機能:

外部トリガによる変換開始の有無とアクティブエッジの設定を行います。

補足:

**NewState** が **ENABLE** の場合のみ **TrgMode** を選択できます。

戻り値:

なし

## 9.2.3.18 TMRB\_SetClkInCoreHalt

デバッグ HALT 中のクロック動作

関数のプロトタイプ宣言:

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

引数:

**TBx**: TMRB チャンネルを指定します。

**ClkState**: デバッグ HALT 中のクロック動作を選択します。

- **TMRB\_RUNNING\_IN\_CORE\_HALT**: 動作
- **TMRB\_STOP\_IN\_CORE\_HALT**: 停止

機能:

デバッグツール使用時に HALT モードに遷移した場合、TMRB クロック動作/停止の設定を行いません。

戻り値:

なし

## 9.2.3.19 TMRB\_SetExtInput

外部入力の設定。

関数のプロトタイプ宣言:

```
void  
TMRB_SetExtInput (TSB_TB_TypeDef* TBx,  
                  uint8_t ExtInput)
```

引数:

**TBx**: TMRB チャンネルを選択します。

**ExtInput**: 以下のいずれかの外部入力を選択してください。

- **TMRB\_EXT\_INPUT\_TBxIN**: TBxIN0/1
- **TMRB\_EXT\_INPUT\_PHCxIN**: PHCxIN0/1

機能:

外部入力として TBxIN0/1 または PHCxIN0/1 を設定します。

戻り値:

なし

## 9.2.3.20 TMRB\_SetDMAReq

DMA 要求の制御

関数のプロトタイプ宣言:

```
void  
TMRB_SetExtStartTrg (TSB_TB_TypeDef* TBx,  
                     FunctionalState NewState,  
                     uint8_t DMAReq)
```

引数:

**TBx**: TMRB チャンネルを選択します。

**NewState**: 以下から DMA 要求の許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

**DMAReq**: 以下から DMA 要求の種類を選択します。

- **TMRB\_DMA\_REQ\_CMP\_MATCH**: コンペア一致
- **TMRB\_DMA\_REQ\_CAPTURE\_1**: インพุットキャプチャ 1
- **TMRB\_DMA\_REQ\_CAPTURE\_0**: インพุットキャプチャ 0

機能:

DMA 要求の制御を行います。

戻り値:

なし

補足:

TBxIM レジスタで割り込みをマスク設定している場合、DMA 要求を許可しても要求は発生しません。

## 9.2.3.21 TMRB\_SetFT0Sel

ソースクロック  $\phi$  T0 の選択

関数のプロトタイプ宣言:

```
void  
TMRB_SetFT0Sel (TSB_TB_TypeDef* TBx,  
                 FunctionalState NewState)
```

引数:

**TBx**: TMRB チャンネルを選択します。

**NewState**: 以下からソースクロック  $\phi$  T0 を選択します。

- **ENABLE**:  $\phi$  T0 を選択
- **DISABLE**:  $\phi$  T0 以外を選択

機能:

ソースクロック  $\phi$  T0 を選択します。

戻り値:

なし

## 9.2.4 データ構造

### 9.2.4.1 TMRB\_InitTypeDef

メンバ:

uint32\_t

**Mode**: タイマモードを選択します。

- **TMRB\_INTERVAL\_TIMER**: インタバルタイマ
- **TMRB\_EVENT\_CNT**: イベントカウンタモード

uint32\_t

**ClkDiv**: インタバルタイマのソースクロックの分周を選択します。

- **TMRB\_CLK\_DIV\_2**: fperiph / 2
- **TMRB\_CLK\_DIV\_8**: fperiph / 8
- **TMRB\_CLK\_DIV\_32**: fperiph / 32
- **TMRB\_CLK\_DIV\_64**: fperiph / 64
- **TMRB\_CLK\_DIV\_128**: fperiph / 128
- **TMRB\_CLK\_DIV\_256**: fperiph / 256
- **TMRB\_CLK\_DIV\_512**: fperiph / 512

uint32\_t

**TrailingTiming**: TBnRG1 へ書き込む周期 (最大 0xFFFF)

uint32\_t

**UpCntCtrl**: アップカウンタの動作を選択します。

- **TMRB\_FREE\_RUN**: 周期が一致した後も、0xFFFF になるまでアップカウンタは停止しません。その後、カウンタがクリアされ、0 からカウントを開始します。
- **TMRB\_AUTO\_CLEAR**: **TrailingTiming** と一致したときに、0 クリアされ、再スタートします。



uint32\_t

**LeadingTiming:** TBnRG0 に書き込むデューティ (最大 0xFFFF)。 **TrailingTiming** 以上の値を設定できません。

## 9.2.4.2 TMRB\_FFOutputTypeDef

メンバ:

uint32\_t

**FlipflopCtrl:** フリップフロップのレベルを選択します。

- **TMRB\_FLIPFLOP\_INVERT:** TBxFF0 の値を反転(ソフト反転)します。
- **TMRB\_FLIPFLOP\_SET:** TBxFF0 を"1"にセットします。
- **TMRB\_FLIPFLOP\_CLEAR:** TBxFF0 を"0"にクリアします。

uint32\_t

**FlipflopReverseTrg:** 以下から、フリップフロップの反転トリガを選択します。

- **TMRB\_DISALBE\_FLIPFLOP:** 反転トリガを無効にします。
- **TMRB\_FLIPFLOP\_TAKE\_CATPURE\_0:** アップカウンタの値がキャプチャレジスタ 0 に取り込まれた時にタイマフリップフロップを反転します。
- **TMRB\_FLIPFLOP\_TAKE\_CATPURE\_1:** アップカウンタの値がキャプチャレジスタ 1 に取り込まれた時にタイマフリップフロップを反転します。
- **TMRB\_FLIPFLOP\_MATCH\_TRAILING:** アップカウンタと周期との一致時にタイマフリップフロップを反転します。
- **TMRB\_FLIPFLOP\_MATCH\_LEADING:** アップカウンタとデューティとの一致時にタイマフリップフロップを反転します。

## 9.2.4.3 TMRB\_INTFactor

メンバ:

uint32\_t

**All:** TMRB 割り込み要因

**Bit**

uint32\_t

**MatchLeadingTiming:** 1 デューティとの一致検出

uint32\_t

**MatchTrailingTiming:** 1 周期との一致検出

uint32\_t

**OverFlow:** 1                      オーバーフロー

uint32\_t

**Reserverd:** 29                      未使用

## 10. SIO/UART

### 10.1 概要

本デバイスのシリアル I/O チャンネルは、I/O インタフェースモード(同期通信モード)と 7, 8, 9 ビット長の UART モード(非同期通信)を実装しています。9 ビット UART モードでは、シリアルリンク(マルチコントローラ・システム) でマスタコントローラがスレーブコントローラを起動するときにウェイクアップ機能が使用されます。

本ドライバは、ボーレート、ビット長、パリティチェック、ストップビット、フローコントロールなどの各チャンネルの設定に関する関数、およびデータ送受信、エラーチェックなどの動作に関する機能を備えています。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。

/Libraries/TX03\_Periph\_Driver/src/tmpm365\_uart.c

/Libraries/TX03\_Periph\_Driver/inc/tmpm365\_uart.h

### 10.2 API 関数

#### 10.2.1 関数一覧

- ◆ 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\_SetRxDMAReq (TSB\_SC\_TypeDef\* **UARTx**, FunctionalState **NewState**)
- ◆ void UART\_SetTxDMAReq (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\_TRxDisable  
**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**)
- ◆ uint32\_t UART\_GetRxFIFOFillLevelStatus(TSB\_SC\_TypeDef \* **UARTx**)

- ◆ uint32\_t UART\_GetRxFIFOOverRunStatus(TSB\_SC\_TypeDef \* **UARTx**)
- ◆ uint32\_t UART\_GetTxFIFOFillLevelStatus(TSB\_SC\_TypeDef \* **UARTx**)
- ◆ uint32\_t UART\_GetTxFIFOUnderRunStatus(TSB\_SC\_TypeDef \* **UARTx**)
- ◆ void SIO\_Enable(TSB\_SC\_TypeDef \* **SIOx**)
- ◆ void SIO\_Disable(TSB\_SC\_TypeDef \* **SIOx**)
- ◆ uint8\_t SIO\_GetRxData(TSB\_SC\_TypeDef \* **SIOx**)
- ◆ void SIO\_SetTxData(TSB\_SC\_TypeDef \* **SIOx**, uint8\_t **Data**)
- ◆ void SIO\_Init(TSB\_SC\_TypeDef \* **SIOx**, uint32\_t **IOClkSel**, SIO\_InitTypeDef \* **InitStruct**)

## 10.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

1. 初期化と設定:  
UART\_Enable(), UART\_Disable(), UART\_Init(), UART\_DefaultConfig(),  
SIO\_Enable(), SIO\_Disable(), SIO\_Init()
2. 送受信設定とエラー確認:  
UART\_GetBufState(), UART\_GetRxData(), UART\_SetTxData(),  
UART\_GetErrState(), SIO\_GetRxData(), SIO\_SetTxData()
3. その他:  
UART\_SetRxDMAReq(), UART\_SetTxDMAReq(), UART\_SWReset(),  
UART\_SetWakeUpFunc(), UART\_SetIdleMode()
4. FIFO モードの設定:  
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\_GetRxFIFOFillLevelStatus(), UART\_GetRxFIFOOverRunStatus(),  
UART\_GetTxFIFOFillLevelStatus(), UART\_GetTxFIFOUnderRunStatus()

## 10.2.3 関数仕様

**補足:** 引数に記述している“TSB\_SC\_TypeDef\* **UARTx**”は以下から選択してください。

**UART0, UART1**

引数に記述している“TSB\_SC\_TypeDef\* **SIOx**”は以下から選択してください。

**SIO0, SIO1**

### 10.2.3.1 UART\_Enable

UART 機能の許可

**関数のプロトタイプ宣言:**

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**機能:**

UART 機能を有効にします。

**戻り値:**

なし

## 10.2.3.2 UART\_Disable

UART 機能の禁止

関数のプロトタイプ宣言:

```
void  
UART_Disable(TSB_SC_TypeDef* UARTx)
```

引数:

**UARTx**: UART チャンネルを指定します。

機能:

UART 機能を無効にします。

戻り値:

なし

## 10.2.3.3 UART\_GetBufState

送受信バッファ状態の読み込み

関数のプロトタイプ宣言:

```
WorkState  
UART_GetBufState(TSB_SC_TypeDef* UARTx,  
                  uint8_t Direction)
```

引数:

**UARTx**: UART チャンネルを指定します。

**Direction**: 送信/受信を選択します。

- **UART\_RX**: 受信
- **UART\_TX**: 送信

機能:

**Direction** が **UART\_RX** の場合、以下の受信バッファの状態を返します。

**DONE**: 受信データはバッファに保存済み

**BUSY**: データ受信中

**Direction** が **UART\_TX** の場合、以下の送信バッファの状態を返します。

**DONE**: バッファ中のデータは送信済み

**BUSY**: データ送信中

戻り値:

**DONE**: バッファリード/ライト可能状態

**BUSY**: 送受信中

## 10.2.3.4 UART\_SWReset

ソフトウェアリセットの実行

関数のプロトタイプ宣言:

```
void  
UART_SWReset(TSB_SC_TypeDef* UARTx)
```

**引数:**

**UARTx:** UART チャンネルを指定します。

**機能:**

ソフトウェアリセットを実行します。

**戻り値:**

なし

## 10.2.3.5 UART\_Init

UART チャンネルの初期化

**関数のプロトタイプ宣言:**

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**InitStruct:** UART に関する構造体です。(詳細は“データ構造”を参照)

**機能:**

ボーレート、ビット単位の転送、ストップビット、パリティ、転送モード、フローコントロールなどの初期設定を行います。

**戻り値:**

なし

## 10.2.3.6 UART\_GetRxData

受信データの読み込み

**関数のプロトタイプ宣言:**

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**機能:**

受信データを読み込みます。**UART\_GetBufState(UARTx, UART\_RX)**にて **DONE** を読み出した後、もしくは UART (シリアルチャネル) 割り込み関数の中で実行してください。

**戻り値:**

受信データです。データ範囲は 0x00~0x1FF です。

## 10.2.3.7 UART\_SetTxData

送信データの設定

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

**Data**: 送信データ(7 ビット、8 ビット、9 ビット)

機能:

送信データを設定します。**UART\_GetBufState(UARTx, UART\_TX)**にて **DONE** を読み出した後、もしくは UART (シリアルチャンネル) 割り込み関数の中で実行してください。

戻り値:

なし

## 10.2.3.8 UART\_DefaultConfig

デフォルト構成での初期化

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

以下の構成で初期化します:

ボーレート: 115200 bps

データ長: 8 ビット

ストップビット: 1 ビット

パリティ: なし

フローコントロール: なし

送受信有効。ボーレートジェネレータはソースクロックとして使用。

戻り値:

なし

## 10.2.3.9 UART\_GetErrState

転送エラーフラグの読み出し

関数のプロトタイプ宣言:

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

**引数:**

**UARTx**: UART チャンネルを指定します。

**機能:**

転送エラーフラグを読み出します。

**戻り値:**

**UART\_NO\_ERR**: エラーなし

**UART\_OVERRUN**: オーバーランエラー

**UART\_PARITY\_ERR**: パリティエラー

**UART\_FRAMING\_ERR**: フレーミングエラー

**UART\_ERRS**: 上記の 2 つ以上のエラーが発生している

## 10.2.3.10 UART\_SetWakeUpFunc

9 ビットモード時のウェイクアップ機能の設定

**関数のプロトタイプ宣言:**

void

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

**引数:**

**UARTx**: UART チャンネルを指定します。

**NewState**: ウェイクアップ機能の有効/無効を選択します。

➤ **ENABLE**: 有効

➤ **DISABLE**: 無効

**機能:**

9 ビットモード時のウェイクアップ機能を設定します。

**NewState** が **ENABLE** の場合、ウェイクアップ機能を有効に、

**NewState** が **DISABLE** の場合、ウェイクアップ機能を無効に設定します。

ウェイクアップ機能は、9 ビットモード時のみ機能します。

**戻り値:**

なし

## 10.2.3.11 UART\_SetIdleMode

IDLE 時の動作

**関数のプロトタイプ宣言:**

void

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

**引数:**

**UARTx**: UART チャンネルを指定します。

**NewState**: IDLE 時の動作を選択します。

➤ **ENABLE**: 動作

➤ **DISABLE:** 停止

**機能:**

**NewState** が **ENABLE** の場合、IDLE 時でも UART チャンネルは動作します。**DISABLE** の場合、IDLE 時は動作を停止します。

**戻り値:**

なし

## 10.2.3.12 UART\_SetRxDMAReq

受信割り込みによる DMA 要求の設定

**関数のプロトタイプ宣言:**

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**NewState:** 以下から受信割り込みによる DMA 要求の許可/禁止を選択します。

➤ **ENABLE:** 許可

➤ **DISABLE:** 禁止

**機能:**

受信割り込みによる DMA 要求 (受信割り込み INTRX 発生により DMA リクエストを発行) を設定します。

**戻り値:**

なし

## 10.2.3.13 UART\_SetTxDMAReq

送信割り込みによる DMA 要求の設定

**関数のプロトタイプ宣言:**

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**NewState:** 以下から送信割り込みによる DMA 要求の許可/禁止を選択します。

➤ **ENABLE:** 許可

➤ **DISABLE:** 禁止

**機能:**

送信割り込みによる DMA 要求 (送信割り込み INTTX 発生により DMA リクエストを発行) を設定します。



戻り値:  
なし

## 10.2.3.14 UART\_FIFOConfig

FIFO の許可

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

**NewState**: FIFO の許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

FIFO の許可/禁止を選択します。

**NewState** が **ENABLE** の場合、FIFO を許可します。**DISABLE** の場合、FIFO を禁止します。

戻り値:  
なし

## 10.2.3.15 UART\_SetFIFOTransferMode

転送モードの選択

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

**TransferMode**: 転送モードを選択します。

- **UART\_TRANSFER\_PROHIBIT**: 転送禁止
- **UART\_TRANSFER\_HALFDPX\_RX**: 半二重(受信)
- **UART\_TRANSFER\_HALFDPX\_TX**: 半二重(送信)
- **UART\_TRANSFER\_FULLDPX**: 全二重

機能:

転送モードを選択します。

戻り値:  
なし

## 10.2.3.16 UART\_TRxAutoDisable

送信/受信の自動禁止

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

**TRxAutoDisable**: 送信/受信の自動禁止機能を制御します。

- **UART\_RTXCNT\_NONE**: なし
- **UART\_RTXCNT\_AUTODISABLE**: 自動禁止

機能:

送信/受信の自動禁止機能を制御します。

戻り値:

なし

## 10.2.3.17 UART\_RxFIFOINTCtrl

受信 FIFO 使用時の受信割り込み許可

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

**NewState**: 受信 FIFO 使用時の受信割り込みの許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

受信 FIFO 有効にされている時の受信割り込みの許可/禁止を切り替えます。

戻り値:

なし

## 10.2.3.18 UART\_TxFIFOINTCtrl

送信 FIFO 使用時の送信割り込み許可

関数のプロトタイプ宣言:

```
void  
UART_TxFIFOINTCtrl (TSB_SC_TypeDef * UARTx,  
                    FunctionalState NewState)
```

引数:

**UARTx:** UART チャンネルを指定します。

**NewState:** 送信 FIFO 使用時の送信割り込みの許可/禁止を選択します。

- **ENABLE:** 許可
- **DISABLE:** 禁止

**機能:**

送信 FIFO 有効にされている時の送信割り込みの許可/禁止を切り替えます。

**戻り値:**

なし

## 10.2.3.19 UART\_RxFIFOByteSel

受信 FIFO 使用バイト数

**関数のプロトタイプ宣言:**

void

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**BytesUsed:** 受信 FIFO 使用バイト数を設定します。

- **UART\_RXFIFO\_MAX:** 最大
- **UART\_RXFIFO\_RXFLEVEL:** 受信 FIFO の FILL レベルに同じ

**機能:**

受信 FIFO 使用バイト数を設定します。

**戻り値:**

なし

## 10.2.3.20 UART\_RxFIFOFillLevel

受信割り込みが発生する受信 FIFO の fill レベルの設定

**関数のプロトタイプ宣言:**

void

UART\_RxFIFOFillLevel (TSB\_SC\_TypeDef \* **UARTx**,  
uint32\_t **RxFIFOLevel**)

**引数:**

**UARTx:** UART チャンネルを指定します。

**RxFIFOLevel:** 受信 FIFO の fill レベルを選択します。

<b>RxFIFOLevel</b>	半二重	全二重
<b>UART_RXFIFO4B_FLEVLE_4_2B</b>	4 バイト	2 バイト
<b>UART_RXFIFO4B_FLEVLE_1_1B</b>	1 バイト	1 バイト
<b>UART_RXFIFO4B_FLEVLE_2_2B</b>	2 バイト	2 バイト
<b>UART_RXFIFO4B_FLEVLE_3_1B</b>	3 バイト	1 バイト

**機能:**

受信割り込みが発生する受信 FIFO の fill レベルを選択します。

戻り値:  
なし

## 10.2.3.21 UART\_RxFIFOINTSel

受信割り込み発生条件の選択

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

**RxINTCondition**: 受信 割り込み発生条件を選択します。

- **UART\_RFIS\_REACH\_FLEVEL**: FIFO fill レベル==割り込み発生 fill レベル
- **UART\_RFIS\_REACH\_EXCEED\_FLEVEL**: FIFO fill レベル≤割り込み発生 fill レベル

機能:

受信割り込み発生条件を選択します。

戻り値:  
なし

## 10.2.3.22 UART\_RxFIFOClear

受信 FIFO クリア

関数のプロトタイプ宣言:

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

受信 FIFO をクリアします。

戻り値:  
なし

## 10.2.3.23 UART\_TxFIFOFillLevel

送信割り込みが発生する送信 FIFO の fill レベルの設定

関数のプロトタイプ宣言:

```
void  
UART_TxFIFOFillLevel (TSB_SC_TypeDef * UARTx,
```

uint32\_t *TxFIFOLevel*)

引数:

**UARTx**: UART チャンネルを指定します。

**TxFIFOLevel**: 受信 FIFO の fill レベルを選択します。

<i>TxFIFOLevel</i>	半二重	全二重
UART_TXFIFO4B_FLEVLE_0_0B	Empty	Empty
UART_TXFIFO4B_FLEVLE_1_1B	1 バイト	1 バイト
UART_TXFIFO4B_FLEVLE_2_0B	2 バイト	Empty
UART_TXFIFO4B_FLEVLE_3_1B	3 バイト	1 バイト

機能:

送信割り込みが発生する送信 FIFO の fill レベルを選択します。

機能:

送信割り込みが発生する送信 FIFO の fill レベルを選択します。

戻り値:

なし

## 10.2.3.24 UART\_TxFIFOINTSel

送信割り込み発生条件の選択

関数のプロトタイプ宣言:

```
void
UART_TxFIFOINTSel (TSB_SC_TypeDef * UARTx,
uint32_t TxINTCondition)
```

引数:

**UARTx**: UART チャンネルを指定します。

**TxINTCondition**: 受信 割り込み発生条件を選択します。

- **UART\_TFIS\_REACH\_FLEVEL**: FIFO fill レベル==割り込み発生 fill レベル
- **UART\_TFIS\_REACH\_EXCEED\_FLEVEL**: FIFO fill レベル≤割り込み発生 fill レベル

機能:

送信割り込み発生条件を選択します。

機能:

送信割り込み発生条件を選択します。

戻り値:

なし

## 10.2.3.25 UART\_TxFIFOClear

送信 FIFO クリア

関数のプロトタイプ宣言:

```
void
```

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

送信 FIFO をクリアします。

戻り値:

なし

## 2.1.1.1. UART\_GetRxFIFOFillLevelStatus

受信 FIFO の fill レベルの取得

関数のプロトタイプ宣言:

uint32\_t

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

受信 FIFO の fill レベルを取得します。

戻り値:

- **UART\_TRXFIFO\_EMPTY**: Empty
- **UART\_TRXFIFO\_1B**: 1 バイト
- **UART\_TRXFIFO\_2B**: 2 バイト
- **UART\_TRXFIFO\_3B**: 3 バイト
- **UART\_TRXFIFO\_4B**: 4 バイト

## 2.1.1.2. UART\_GetRxFIFOOverRunStatus

受信 FIFO オーバーラン状態の取得

関数のプロトタイプ宣言:

uint32\_t

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

受信 FIFO オーバーラン状態を取得します。

戻り値:

**UART\_RXFIFO\_OVERRUN**: オーバーラン発生

## 2.1.1.3. UART\_GetTxFIFOFillLevelStatus

送信 FIFO の fill レベルの取得

関数のプロトタイプ宣言:

uint32\_t

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

送信 FIFO の fill レベルの取得

戻り値:

- **UART\_TRXFIFO\_EMPTY**: Empty
- **UART\_TRXFIFO\_1B**: 1 バイト
- **UART\_TRXFIFO\_2B**: 2 バイト
- **UART\_TRXFIFO\_3B**: 3 バイト
- **UART\_TRXFIFO\_4B**: 4 バイト

## 2.1.1.4. UART\_GetTxFIFOUnderRunStatus

送信 FIFO アンダーラン状態の取得

関数のプロトタイプ宣言:

uint32\_t

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

引数:

**UARTx**: UART チャンネルを指定します。

機能:

送信 FIFO アンダーラン状態を取得します。

戻り値:

**UART\_TXFIFO\_UNDERRUN**: アンダーラン発生

## 10.2.3.26 SIO\_Enable

SIO 動作の許可

関数のプロトタイプ宣言:

void

SIO\_Enable (TSB\_SC\_TypeDef\* **SIOx**)

引数:

**SIOx**: SIO チャンネルを指定します。

機能:

SIO 動作を許可します。

戻り値:  
なし

## 10.2.3.27 SIO\_Disable

SIO 動作の禁止

関数のプロトタイプ宣言:  
void  
SIO\_Disable(TSB\_SC\_TypeDef\* **SIOx**)

引数:  
**SIOx**: SIO チャンネルを指定します。

機能:  
SIO 動作を禁止します。

戻り値:  
なし

## 10.2.3.28 SIO\_GetRxData

受信用バッファの取得

関数のプロトタイプ宣言:  
uint32\_t  
SIO\_GetRxData(TSB\_SC\_TypeDef\* **SIOx**)

引数:  
**SIOx**: SIO チャンネルを指定します。

機能:  
受信用バッファを取得します。

戻り値:  
受信用バッファ(値の範囲は 0x00 ~ 0xFF です)

## 10.2.3.29 SIO\_SetTxData

送信用バッファの設定

関数のプロトタイプ宣言:  
void  
SIO\_SetTxData(TSB\_SC\_TypeDef\* **SIOx**,  
uint8\_t **Data**)

引数:  
**SIOx**: SIO チャンネルを指定します。  
**Data**: 送信用バッファ

機能:



送信用バッファを指定します。

戻り値:  
なし

### 10.2.3.30 SIO\_Init

SIO チャンネルの初期化

関数のプロトタイプ宣言:

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

引数:

***SIOx***: SIO チャンネルを指定します。

***IOClkSel***: クロックを選択します。

➤ **SIO\_CLK\_BAUDRATE**: ポーレートジェネレータ

➤ **SIO\_CLK\_SCLKINPUT**: SCLKx 端子入力

***InitStruct***: SIO に関する構造体です。(詳細は“データ構造”を参照)

機能:

ポーレート、転送方向、転送モードなどの初期設定を行います。

戻り値:  
なし

## 10.2.4 データ構造

### 10.2.4.1 UART\_InitTypeDef

メンバ

uint32\_t

***BaudRate***: UART 通信ポーレートを 2400(bps) から 115200(bps) に設定。(\*)

uint32\_t

***DataBits***: 転送ビット数を選択します。

➤ **UART\_DATA\_BITS\_7**: 7 ビットモード

➤ **UART\_DATA\_BITS\_8**: 8 ビットモード

➤ **UART\_DATA\_BITS\_9**: 9 ビットモード

uint32\_t

***StopBits***: ストップビット長を選択します。

➤ **UART\_STOP\_BITS\_1**: 1 ビット

➤ **UART\_STOP\_BITS\_2**: 2 ビット

uint32\_t

***Parity***: パリティを選択します。

➤ **UART\_NO\_PARITY**: パリティなし

- **UART\_EVEN\_PARITY:** 偶数(Even) パリティ
- **UART\_ODD\_PARITY:** 奇数(Odd) パリティ

uint32\_t

**Mode:** 転送モードを選択します。送受信の場合は、送信と受信をOR 演算子によって組み合わせてください。

- **UART\_ENABLE\_TX:** 送信許可
- **UART\_ENABLE\_RX:** 受信許可

uint32\_t

**FlowCtrl:** フロー制御モードを選択します。(\*\*)

- **UART\_NONE\_FLOW\_CTRL:** フロー制御 無効

(\*)補足:

fperiph の周波数が高すぎる、または、低すぎると、ボーレートが正しく設定できない場合があります。

(\*\*)補足:

UART\_NONE\_FLOW\_CTRLのみ選択可能です。

## 10.2.4.2 SIO\_InitTypeDef

メンバ:

uint32\_t

**InputClkEdge:** 入力クロックエッジを選択します。"0"(SIO\_SCLKS\_TXDF\_RXDR)のみ指定可能です。

uint32\_t

**IntervalTime:** 連続転送時のインターバル時間を選択します。

- **SIO\_SINT\_TIME\_NONE:** なし
- **SIO\_SINT\_TIME\_SCLK\_1:** 1\*SCLK
- **SIO\_SINT\_TIME\_SCLK\_2:** 2\*SCLK
- **SIO\_SINT\_TIME\_SCLK\_4:** 4\*SCLK
- **SIO\_SINT\_TIME\_SCLK\_8:** 8\*SCLK
- **SIO\_SINT\_TIME\_SCLK\_16:** 16\*SCLK
- **SIO\_SINT\_TIME\_SCLK\_32:** 32\*SCLK
- **SIO\_SINT\_TIME\_SCLK\_64:** 64\*SCLK

uint32\_t

**TransferMode:** 転送モードを選択します。

- **SIO\_TRANSFER\_PROHIBIT:** 転送禁止
- **SIO\_TRANSFER\_HALFDPX\_RX:** 半二重(受信)
- **SIO\_TRANSFER\_HALFDPX\_TX:** 半二重(送信)
- **SIO\_TRANSFER\_FULLDPX:** 全二重

uint32\_t

**TransferDir:** 転送方向を選択します。

- **SIO\_LSB\_FRIST:** LSB FRIST
- **SIO\_MSB\_FRIST:** MSB FRIST

uint32\_t

**Mode:** 送受信を制御します。有効ビットの組み合わせが可能です。

- **SIO\_ENABLE\_TX:** 送信許可
- **SIO\_ENABLE\_RX:** 受信許可

uint32\_t

**DoubleBuffer:** ダブルバッファの許可/禁止を選択します。

- **SIO\_WBUF\_ENABLE:** 許可
- **SIO\_WBUF\_DISABLE:** 禁止

uint32\_t

**BaudRateClock:** ボーレートジェネレータ入力クロックを選択します。

- **SIO\_BR\_CLOCK\_T1:**  $\phi T1$
- **SIO\_BR\_CLOCK\_T4:**  $\phi T4$
- **SIO\_BR\_CLOCK\_T16:**  $\phi T16$
- **SIO\_BR\_CLOCK\_T64:**  $\phi T64$

uint32\_t

**Divider:** 分周値"N"を選択します。

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

## 11. USB D

### 11.1 概要

本デバイスは USB デバイスコントローラを内蔵し、その機能は下記の様になります。

1. Universal Serial Bus Specification Rev.2.0 に準拠
2. Full-Speed をサポート (Low-Speed は非対応)
3. USB プロトコル処理
4. SOF/USB\_RESET/SUSPEND/RESUME の検出
5. パケット ID の生成とチェック
6. CRC5 チェック、CRC16 の生成とチェック
7. 4 種類の(Control/ Interrupt/ Bulk/ Isochronous)転送モードをサポート
8. エンドポイントのサポート

Endpoint0	コントロール	64byte x 1 FIFO
Endpoint1	コントロール/インターラプト / バルク / アイソクロナス(IN)	64byte x 2 FIFO
Endpoint2	コントロール/インターラプト / バルク / アイソクロナス(OUT)	64byte x 2 FIFO
Endpoint3	コントロール/インターラプト / バルク / アイソクロナス(IN)	64byte x 2 FIFO
Endpoint4	コントロール/インターラプト / バルク / アイソクロナス(OUT)	64byte x 2 FIFO
Endpoint5	コントロール/インターラプト / バルク / アイソクロナス(IN)	64byte x 2 FIFO
Endpoint6	コントロール/インターラプト / バルク / アイソクロナス(OUT)	64byte x 2 FIFO
Endpoint7	コントロール/インターラプト / バルク / アイソクロナス(IN)	64byte x 2 FIFO

9. デュアルパケットモード対応 (エンドポイント 0 は除く)
10. 割り込みコントローラへの割り込み要因信号: INTUSB, INTUSBWKUP

全ての基本ドライバ API はマクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。

\Libraries\TX03\_USBD\_Driver\src\usbd\_hw.c  
 \Libraries\TX03\_USBD\_Driver\inc\usbd\_hw.h

### 11.2 API 関数

#### 11.2.1 関数一覧

- ◆ USBD\_INTStatus USBD\_GetINTStatus(void)
- ◆ void USBD\_SetINTMask(USBD\_INTSrc **IntSrc**, FunctionalState **NewState**)
- ◆ void USBD\_ClearINT(USBD\_INTSrc **IntSrc**)
- ◆ USBD\_DMACKConfig USBD\_GetDMACKConfig(void)
- ◆ void USBD\_ConfigDMACK(USBD\_DMACKConfig **Config**)
- ◆ USBD\_DMACKStatus USBD\_GetDMACKStatus(void)
- ◆ void USBD\_ReadUDC2Reg(uint32\_t **Addr**, uint32\_t \* **Data**)
- ◆ void USBD\_WriteUDC2Reg(uint32\_t **Addr**, const uint32\_t **Data**)
- ◆ USBD\_DMACKAddr USBD\_GetDMACKMasterAddr(USBD\_MasterMode **MasterMode**)
- ◆ void USBD\_SetDMACKMasterAddr(USBD\_MasterMode **MasterMode**, USBD\_DMACKAddr **Addr**)
- ◆ USBD\_PowerCtrl USBD\_GetPowerCtrlStatus(void)

- ◆ void USBD\_SetPowerCtrl(USB\_D\_PowerCtrl **PowerCtrl**)
- ◆ void USBD\_SetEPCMD(USB\_D\_EPx **EPx**, USB\_D\_EPCMD **Cmd**)
- ◆ USB\_D\_EP0Status USBD\_GetEP0Status(void)
- ◆ USB\_D\_EPxStatus USBD\_GetEPxStatus(USB\_D\_EPx **EPx**)
- ◆ void USBD\_ConfigEPx(USB\_D\_EPx **EPx**, USB\_D\_EPxConfig **Config**)

## 11.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

- 1) 初期化と共通関数:  
USB\_D\_SetINTMask(), USB\_D\_ClearINT(), USB\_D\_SetEPCMD(),  
USB\_D\_ConfigEPx(), USB\_D\_SetPowerCtrl()
- 2) ステータス関係の関数:  
USB\_D\_GetINTStatus(), USB\_D\_GetEP0Status(), USB\_D\_GetEPxStatus(),  
USB\_D\_GetPowerCtrlStatus()
- 3) DMA 動作関係の関数:  
USB\_D\_GetDMACConfig(), USB\_D\_ConfigDMAC(), USB\_D\_GetDMACStatus(),  
USB\_D\_GetDMACMasterAddr(), USB\_D\_SetDMACMasterAddr()
- 4) UDC2 モジュールのレジスタへアクセスするための特殊関数:  
USB\_D\_ReadUDC2Reg(), USB\_D\_WriteUDC2Reg()

## 11.2.3 関数仕様

### 11.2.3.1 USB\_D\_GetINTStatus

USB\_D 割り込み状態の取得

**関数のプロトタイプ宣言:**

USB\_D\_INTStatus  
USB\_D\_GetINTStatus(void)

**引数:**

なし。

**機能:**

USB\_D の割り込み状態を取得します。

例えば、bit0 に UDC2 割り込み **USB\_D\_INT\_SETUP**、bit8 に UDC2AB 割り込み **USB\_D\_INT\_SUSPEND\_RESUME** など。

**戻り値:**

USB\_D 割り込み要因。詳細は"データ説明"の USB\_D\_INTStatus を参照してください。

### 11.2.3.2 USB\_D\_SetINTMask

USB\_D 割り込みマスクの設定

**関数のプロトタイプ宣言:**

void  
USB\_D\_SetINTMask(USB\_D\_INTSrc **IntSrc**,  
FunctionalState **NewState**)

**引数:**

**IntSrc:** 割り込み要因を選択します。

- **USBD\_INT\_SETUP:** UDC2 の int\_setup 信号
- **USBD\_INT\_STATUS\_NAK:** UDC2 の int\_status\_nak 信号
- **USBD\_INT\_STATUS:** UDC2 の int\_status 信号
- **USBD\_INT\_RX\_ZERO:** UDC2 の int\_rx\_zero 信号
- **USBD\_INT\_SOF:** UDC2 の int\_sof signal 信号
- **USBD\_INT\_EP0:** UDC2 の int\_ep0 signal 信号
- **USBD\_INT\_EP:** UDC2 の int\_ep signal 信号
- **USBD\_INT\_NAK:** UDC2 の int\_nak signal 信号
- **USBD\_INT\_SUSPEND\_RESUME:**  
UDC2 の suspend\_x 信号が変更されるたびに割り込み発生
- **USBD\_INT\_USB\_RESET:**  
UDC2 の usb\_reset アサート時に、割り込み発生
- **USBD\_INT\_USB\_RESET\_END:**  
UDC2 の usb\_reset デアサート時に、割り込み発生
- **USBD\_INT\_MW\_SET\_ADD:**  
マスタライト転送アドレス要求時に割り込み発生
- **USBD\_INT\_MW\_END\_ADD:**  
マスタライト転送完了時に割り込み発生
- **USBD\_INT\_MW\_TIMEOUT:**  
マスタライト転送タイムアウト時に割り込み発生
- **USBD\_INT\_MW\_AHBERR:**  
マスタライト転送中の AHB エラー発生時に割り込み発生
- **USBD\_INT\_MR\_END\_ADD:**  
マスタリード転送終了時に割り込み発生
- **USBD\_INT\_MR\_EP\_DSET:**  
マスタリード転送に使用される UDC2 Tx 用エンドポイントの FIFO が書き込み可になると割り込み発生
- **USBD\_INT\_MR\_AHBERR:**  
マスタリード転送中に AHB エラーが発生すると割り込み発生
- **USBD\_INT\_UDC2\_REG\_READ:**  
UDC2 レジスタのリード/ライトが完了すると割り込み発生
- **USBD\_INT\_DMACH\_REG\_READ:**  
DMAC 専用レジスタへのリード/ライトが完了すると割り込み発生
- **USBD\_INT\_MW\_READERROR:**  
マスタライトでエンドポイントリードエラーが発生すると割り込み発生

**NewState:** 割り込み要因マスク状態を下記から選択します。

- **ENABLE :** IntSrc で割り込みを許可
- **DISABLE:** IntSrc で割り込みを禁止

## 機能:

USBD の割り込みのマスクを設定します。

本関数により使用していない割り込みを禁止することができます。

パワーオンリセット後の UDC2 割り込み(**USBD\_INT\_SETUP** から **USBD\_INT\_NAK**)の基本設定は、すべて“**ENABLE**”です。

パワーオンリセット後の UDC2AB 割り込み (**USBD\_INT\_SUSPEND\_RESUME** から **USBD\_INT\_MW\_READERROR**) の基本設定は、すべて“**DISABLE**”です。

## 戻り値:

なし

## 11.2.3.3 USBD\_ClearINT

USB D の割り込みフラグのクリア

関数のプロトタイプ宣言:

void

USB D\_ClearINT(USB D\_INTSrc *IntSrc*)

引数:

*IntSrc*: 下記から割り込み要因を設定

- **USB D\_INT\_SETUP:** UDC2 の int\_setup 信号
- **USB D\_INT\_STATUS\_NAK:** UDC2 の int\_status\_nak 信号
- **USB D\_INT\_STATUS:** UDC2 の int\_status 信号
- **USB D\_INT\_RX\_ZERO:** UDC2 の int\_rx\_zero 信号
- **USB D\_INT\_SOF:** UDC2 の int\_sof 信号
- **USB D\_INT\_EP0:** UDC2 の int\_ep0 信号
- **USB D\_INT\_EP:** UDC2 の int\_ep 信号
- **USB D\_INT\_NAK:** UDC2 の int\_nak 信号
- **USB D\_INT\_SUSPEND\_RESUME:**  
UDC2 の suspend\_x 信号が変化するたびに割り込み発生
- **USB D\_INT\_USB\_RESET:**  
UDC2 が usb\_reset 信号をアサートするたびに割り込み発生
- **USB D\_INT\_USB\_RESET\_END:**  
UDC2 が usb\_reset 信号をデアサートするたびに割り込み発生
- **USB D\_INT\_MW\_SET\_ADD:**  
マスタライト転送アドレス要求のたびに割り込み発生
- **USB D\_INT\_MW\_END\_ADD:**  
マスタライト転送完了時に割り込み発生
- **USB D\_INT\_MW\_TIMEOUT:**  
マスタライト転送タイムアウト時に割り込み発生
- **USB D\_INT\_MW\_AHBERR:**  
マスタライト転送中に AHB エラーが発生すると割り込み発生
- **USB D\_INT\_MR\_END\_ADD:**  
マスタライト転送中完了時に割り込み発生
- **USB D\_INT\_MR\_EP\_DSET:**  
マスタリード転送に使用される UDC2 Tx 用エンドポイントの FIFO が書き込み可になると割り込み発生
- **USB D\_INT\_MR\_AHBERR:**  
マスタリード転送中に AHB エラーが発生すると割り込み発生
- **USB D\_INT\_UDC2\_REG\_READ:**  
UDC2 レジスタへのリード/ライトが完了すると割り込み発生
- **USB D\_INT\_DMAC\_REG\_READ:**  
DMAC 専用レジスタへのリード/ライトが完了すると割り込み発生
- **USB D\_INT\_POWERDETECT:**  
VBUSPOWER 入力のステータスが変化すると割り込み発生
- **USB D\_INT\_MW\_READERROR:**  
マスタライト中にエンドポイントエラーが発生すると割り込み発生

機能:

USB D の割り込みフラグをクリアします。

本関数によって割り込みフラグをクリアしてください。

戻り値:

なし

## 11.2.3.4 USBD\_GetDMACConfig

USB の DMA コントローラ設定状態の読み込み

関数のプロトタイプ宣言:

USBDMACConfig

USBDMACConfig(void)

引数:

なし。

機能:

“マスタライト許可”、“マスタリード許可”、“マスタライトリセット”、“マスタリードリセット”など、USB の DMA コントローラ設定を読み込みます。

戻り値:

USB の DMA コントローラ設定。詳細は“データ構造”の USBDMACConfig を参照してください。

## 11.2.3.5 USBDMACConfig

USB の DMA コントローラの設定

関数のプロトタイプ宣言:

void

USBDMACConfig(USBDMACConfig **Config**)

引数:

**Config**: USB の DMA コントローラ設定。詳細は“データ構造”の USBDMACConfig を参照してください。

機能:

“マスタライト許可”、“マスタリード許可”、“マスタライトリセット”、“マスタリードリセット”など、USB の DMA コントローラの設定をします。

戻り値:

なし

## 11.2.3.6 USBDMACStatus

DMA コントローラマスタ動作状態の読み出し

関数のプロトタイプ宣言:

USBDMACStatus

USBDMACStatus(void)

引数:

なし。



**機能:**

DMA コントローラマスタ動作状態を読み出します。

たとえば、“マスタライトバッファは空”は “MW\_Buf\_Empty” ビット、“マスタリードエンドポイント用データがセットされました”は “MR\_EP\_DSet”ビットになります。

**戻り値:**

DMAC マスタ動作の状態。詳細は“データ構造”の USB\_DMACStatus を参照してください。

### 11.2.3.7 USB\_ReadUDC2Reg

UDC2AB の“UDC2 リード要求レジスタ”経由での UDC2 レジスタのデータ読み出し

**関数のプロトタイプ宣言:**

```
void  
USB_ReadUDC2Reg(uint32_t Addr,  
                uint32_t * Data)
```

**引数:**

**Addr:** UDC2 モジュールのレジスタアドレスを選択します。

- **UDC2\_ADDR:** アドレス状態、デバイスアドレスレジスタ
- **UDC2\_FRAME:** フレームレジスタ
- **UDC2\_COMMAND:** エンドポイント へのコマンドレジスタ
- **UDC2\_BREQ\_BMREQTYPE:**  
設定パッケージの bRequest-bmRequest タイプレジスタ
- **UDC2\_WVALUE:** 設定パッケージの wValue レジスタ
- **UDC2\_WINDEX:** 設定パッケージの wIndex レジスタ
- **UDC2\_WLENGTH:** 設定パッケージの wLength レジスタ
- **UDC2\_INT:**  
UDC2 割り込み信号とマスクビットの INT レジスタ
- **UDC2\_INTEP:**  
エンドポイント (EP0 を除く)の送受信状態用のフラグ
- **UDC2\_INTEP\_MASK:** UDC2\_INTEP のマスク設定
- **UDC2\_INTRX0:**  
エンドポイントで受信される Zero-Length データを示すフラグ。
- **UDC2\_EPxMAXPACKETSIZE( x = 0 to 7 ):** EPx\_MaxPacketSize レジスタ
- **UDC2\_EPxSTATUS( x = 0 to 7 ):** EPx status レジスタ
- **UDC2\_EPxDATASIZE( x = 0 to 7 ):** EPx datasize レジスタ
- **UDC2\_EPxFIFO( x = 0 to 7 ):** EPx FIFO レジスタ
- **UDC2\_INTNAK:**  
エンドポイント (EP0 を除外)の NAK 送信状態を示すフラグ
- **UDC2\_INTNAK\_MASK:** UDC2\_INTNAK のマスク設定制御

**Data:** レジスタ値が格納されるポインターポイント。UDC2 レジスタは uint32\_t 型の low 16 ビットのみを使用します。

**機能:**

UDC2 レジスタからのデータを読み出します。

例えば、USB バスからの全セットアップパッケージは下記パラメータを順に呼び出すことで取得できます。

**UDC2\_BREQ\_BMREQTYPE,**

UDC2\_WVALUE,  
UDC2\_WINDEX,  
UDC2\_WLENGTH

戻り値:

なし

## 11.2.3.8 USBD\_WriteUDC2Reg

UDC2 レジスタへのデータ書き込み

関数のプロトタイプ宣言:

```
void  
USB_D_WriteUDC2Reg(uint32_t Addr,  
                    const uint32_t Data)
```

引数:

**Addr**: UDC2 モジュールのレジスタアドレスを選択します。

- **UDC2\_ADDR**: アドレスステートレジスタ
- **UDC2\_FRAME**: フレームレジスタ
- **UDC2\_COMMAND**: エンドポイント レジスタへのコマンド
- **UDC2\_INT**:  
UDC2 割り込み信号とそのマスクビットの INT レジスタ
- **UDC2\_INTEP**:  
エンドポイント (EP0 を除く) 送受信状態のフラグ.
- **UDC2\_INTEP\_MASK**: **UDC2\_INTEP** の制御マスク
- **UDC2\_INTRX0**:  
エンドポイントで受信された Zero-Length データを表示するフラグ
- **UDC2\_EP0MAXPACKETSIZE**: EP0\_MaxPacketSize レジスタ
- **UDC2\_EP0FIFO**: EP0\_FIFO レジスタ
- **UDC2\_EPxMAXPACKETSIZE (x = 1 to 7)**: EPx Max Packet Size レジスタ
- **UDC2\_EPxSTATUS (x = 0 to 7)**: EPx ステータスレジスタ
- **UDC2\_EPxFIFO (x = 1 to 7)**: EPx FIFO レジスタ
- **UDC2\_INTNAK**:  
エンドポイント (EP0 を除く) の NAK 送信状態を表すフラグ
- **UDC2\_INTNAK\_MASK**: **UDC2\_INTNAK** の制御マスク設定

**Data**: UDC2 レジスタに書き込まれるデータ

機能:

データを上記 “**Addr**” によって指定される UDC2 レジスタに書き込みます。

戻り値:

なし

## 11.2.3.9 USBD\_GetDMACMasterAddr

DMAC マスター動作のアドレスを取得

関数のプロトタイプ宣言:

```
USB_D_DMCAAddr
```

USBD\_GetDMACMasterAddr(USBD\_MasterMode **MasterMode**)

引数:

**MasterMode**: DMAC マスタアドレスタイプを選択します。

- **USBD\_MASTER\_WRITE**:       マスタライト
- **USBD\_MASTER\_READ**:       マスタリード

機能:

“Master Write Start Address”, “Master Write End Address”, “Master Read Start Address”, “Master Read End Address” を含む DMAC マスター動作に関連したアドレスレジスタを取得します。

戻り値:

DMAC マスター動作のアドレス。詳細は、“データ構造”の USBD\_DMCAAddr を参照してください。

### 11.2.3.10 USBD\_SetDMACMasterAddr

DMAC マスター動作のアドレス設定

関数のプロトタイプ宣言:

```
void  
USBD_SetDMACMasterAddr(USBD_MasterMode MasterMode,  
                        USBD_DMCAAddr Addr)
```

引数:

**MasterMode**: DMAC マスターアドレスタイプを下記から設定

- **USBD\_MASTER\_WRITE**:       マスタライト
- **USBD\_MASTER\_READ**:       マスタリード

**Addr**: DMAC マスター動作のアドレス

詳細は、“データ構造”の USBD\_DMCAAddr を参照してください。

機能:

“Master Write Start Address”, “Master Write End Address”, “Master Read Start Address”, “Master Read End Address”を含む DMAC マスター動作に関連したアドレスレジスタの設定をします。

戻り値:

なし

### 11.2.3.11 USBD\_GetPowerCtrlStatus

USB 電源検出制御値の取得

関数のプロトタイプ宣言:

```
USBD_PowerCtrl  
USBD_GetPowerCtrlStatus(void)
```

引数:

なし。

**機能:**

“USB\_Reset”, “PHY\_Suspend”, “PHY\_Resetb” ビットなどの USB\_D 電源検出制御レジスタ値を取得します。

**戻り値:**

USB\_D 電源検出制御のステータス。詳細は、“データ構造”の USB\_D\_PowerCtrl を参照してください。

## 11.2.3.12 USB\_D\_SetPowerCtrl

USB\_D 電源検出制御

**関数のプロトタイプ宣言:**

```
void  
USB_D_SetPowerCtrl(USB_D_PowerCtrl PowerCtrl)
```

**引数:**

**PowerCtrl:** USB\_D 電源検出制御の設定。詳細は、“データ構造”の USB\_D\_PowerCtrl を参照してください。

**機能:**

“USB\_Reset”, “PHY\_Suspend”, “PHY\_Resetb” ビットなど USB\_D 電源検出制御レジスタを設定します。

**戻り値:**

なし

## 11.2.3.13 USB\_D\_SetEPCMD

エンドポイントへのコマンド送信

**関数のプロトタイプ宣言:**

```
void  
USB_D_SetEPCMD(USB_D_EPx EPx,  
                USB_D_EPCMD Cmd)
```

**引数:**

**EPx:** ターゲットのエンドポイントを選択します。

- **USB\_D\_EP0:** USB\_D エンドポイント 0
- **USB\_D\_EP1:** USB\_D エンドポイント 1
- **USB\_D\_EP2:** USB\_D エンドポイント 2
- **USB\_D\_EP3:** USB\_D エンドポイント 3
- **USB\_D\_EP4:** USB\_D エンドポイント 4
- **USB\_D\_EP5:** USB\_D エンドポイント 5
- **USB\_D\_EP6:** USB\_D エンドポイント 6
- **USB\_D\_EP7:** USB\_D エンドポイント 7

**Cmd:** エンドポイントに送られるコマンドを選択します。

**EPx** が **USB\_D\_EP0** の場合:

- **USB\_D\_CMD\_SETUP\_FIN :**

DATA ステージ終了または INT\_STATUS\_NAK 受信時にコマンドを発行してください。

- **USB\_CMD\_EP\_RESET:**  
エンドポイントのデータおよびステータスをクリアするコマンドです。
- **USB\_CMD\_EP\_STALL:**  
エンドポイント のステータスを"Stall"にセットするコマンドです。
- **USB\_CMD\_ALL\_EP\_INVALID:**  
EP0 以外の全エンドポイントのステータスを"Invalid"にセットするコマンドです。
- **USB\_CMD\_USB\_READY:**  
USB ケーブルへの接続をするためのコマンドです。ケーブルに接続されたことを確認後、ホストとの通信が可能になった時点でこのコマンドを発行して下さい。
- **USB\_CMD\_SETUP\_RECEIVED:**  
Control 転送の SETUP-Stage を認識したことを UDC2 へ知らせるためのコマンドです。INT\_SETUP 割込みを受け付けて、リクエストコードを認識した後にこのコマンドを発行して下さい。
- **USB\_CMD\_EP\_EOP:**  
送信データ書き込み終了を UDC2 へ知らせるためのコマンドです。最大転送バイト数よりも少ないバイト数を送信したい場合、このコマンドを発行して下さい。
- **USB\_CMD\_EP\_FIFO\_CLEAR:**  
エンドポイントのデータを削除します。
- **USB\_CMD\_EP\_TX\_0DATA:**  
エンドポイントに Zero-Length データをセットするコマンドです。Zero-Length データを送信したい場合、このコマンドを発行して下さい。

**EPx が USB\_EP<sub>y</sub> (y = 1 ~ 7) の場合:**

- **USB\_CMD\_SET\_DATA0:**  
エンドポイントのトグルをクリアするコマンドです。通常の転送時のトグル更新は UDC2 により自動的に行われますが、ソフトからクリアする必要がある場合はこのコマンドを発行して下さい。
- **USB\_CMD\_EP\_RESET:**  
エンドポイントのデータおよびステータスをクリアするコマンドです。
- **USB\_CMD\_EP\_STALL:**  
エンドポイントのステータスを"Stall"にセットするコマンドです。
- **USB\_CMD\_EP\_INVALID:**  
エンドポイントのステータスを"Invalid"にセットするコマンドです。
- **USB\_CMD\_EP\_DISABLE:**  
エンドポイントを Disable にするコマンドです。
- **USB\_CMD\_EP\_ENABLE:**  
エンドポイントを Enable にするコマンドです。
- **USB\_CMD\_ALL\_EP\_INVALID:**  
EP0 以外の全 EP のステータスを"Invalid"にセットするコマンドです。
- **USB\_CMD\_EP\_EOP:**  
送信データ書き込み終了を UDC2 へ知らせるためのコマンドです。最大転送バイト数よりも少ないバイト数を送信したい場合、このコマンドを発行して下さい。
- **USB\_CMD\_EP\_FIFO\_CLEAR:**  
エンドポイントのデータをクリアするコマンドです。
- **USB\_CMD\_EP\_TX\_0DATA:**

エンドポイントに Zero-Length データをセットするコマンドです。。Zero-Length データを送信したい場合、このコマンドを発行して下さい。

**機能:**

エンドポイントにコマンドを送信します。

**戻り値:**

なし

## 11.2.3.14 USBD\_GetEP0Status

EP0 ステータスの読み出し

**関数のプロトタイプ宣言:**

USB\_EP0Status

USBD\_GetEP0Status(void)

**引数:**

なし。

**機能:**

Endpoint0 の現在のステータスを読み出します。

“Ready”, “Busy”, “Error”, “Stall” などの EP0 ステータスです。「EP0\_FIFO 書き込み可」という情報も読み出します。

**戻り値:**

Endpoint0 ステータス。詳細は、USB\_EP0Status の"データ構造"を参照ください。

## 11.2.3.15 USBD\_GetEPxStatus

EPx ステータスの読み出し (x = 1 ~ 7)

**関数のプロトタイプ宣言:**

USB\_EPxStatus

USBD\_GetEPxStatus(USB\_EPx **EPx**)

**引数:**

**EPx:** ターゲットのエンドポイントを選択します。

- **USB\_EP1:** USBD エンドポイント 1
- **USB\_EP2:** USBD エンドポイント 2
- **USB\_EP3:** USBD エンドポイント 3
- **USB\_EP4:** USBD エンドポイント 4
- **USB\_EP5:** USBD エンドポイント 5
- **USB\_EP6:** USBD エンドポイント 6
- **USB\_EP7:** USBD エンドポイント 7

**機能:**

エンドポイント x (x = 1~7)のステータスを読み出します。

“Ready”, “Busy”, “Error”, “Stall” などの EPx ステータスです。EPx の転送方向、転送モードも取得可能です。

戻り値:

エンドポイント x (x = 1 から 7) のステータス。詳細は、"データ構造" の USBD\_EPxStatus を参照してください。

## 11.2.3.16 USBD\_ConfigEPx

EPx (x = 1 ~ 7) 設定

関数のプロトタイプ宣言:

```
void  
USB_D_ConfigEPx(USB_D_EPx EPx,  
                 USB_D_EPxConfig Config)
```

引数:

**EPx**: ターゲットのエンドポイントを選択します。

- **USB\_D\_EP1**: USB\_D エンドポイント 1
- **USB\_D\_EP2**: USB\_D エンドポイント 2
- **USB\_D\_EP3**: USB\_D エンドポイント 3
- **USB\_D\_EP4**: USB\_D エンドポイント 4
- **USB\_D\_EP5**: USB\_D エンドポイント 5
- **USB\_D\_EP6**: USB\_D エンドポイント 6
- **USB\_D\_EP7**: USB\_D エンドポイント 7

**Config**: EPx の設定情報。詳細は、USB\_D\_EPxConfig の"データ構造"を参照してください。

機能:

エンドポイント x (x = 1 ~ 7) の設定をします。

EPx の転送方向、転送モードを設定することができます。

スタンダードリクエスト設定時に呼び出され、Set\_Configuration と Set\_Interface が受信されます。

戻り値:

なし

## 11.2.4 データ構造

### 11.2.4.1 USB\_D\_DMACHdr

メンバ:

uint32\_t

**StartAddr**: DMAC マスター動作の開始アドレスを選択します。

- **USB\_D\_MW\_START\_ADDR**: マスタライト転送のスタートアドレス
- **USB\_D\_MR\_START\_ADDR**: マスタリード転送のスタートアドレス

uint32\_t

**EndAddr**: DMAC マスター動作のエンドアドレスを選択します。

- **USB\_D\_MW\_END\_ADDR**: マスタライト転送のエンドアドレス
- **USB\_D\_MR\_END\_ADDR**: マスタリード転送のエンドアドレス

## 11.2.4.2 USBD\_INTStatus

メンバ:

uint32\_t

**All**: 下記すべてのビットの OR 値

### Bit

uint32\_t

**Setup**: 1

UDC2 の int\_setup 信号

uint32\_t

**Status\_NAK**: 1

UDC2 の int\_status\_nak 信号

uint32\_t

**Status**: 1

UDC2 の int\_status 信号

uint32\_t

**Rx\_Zero**: 1

UDC2 の int\_rx\_zero 信号

uint32\_t

**SOF**: 1

UDC2 の int\_sof 信号 signal

uint32\_t

**EP0**: 1

UDC2 の int\_ep0 信号

uint32\_t

**EP**: 1

UDC2 の int\_ep 信号

uint32\_t

**NAK**: 1

UDC2 の int\_nak 信号

uint32\_t

**Suspend\_Resume**: 1

UDC2 の suspend\_x 信号が変わるたびに 1 をアサートする

0: ステータス変更なし

1: ステータス変更あり

uint32\_t

**USB\_Reset**: 1

UDC2 が usb\_reset 信号をアサートしたかどうかを表示

0: ビットのクリア後、UDC2 による usb\_reset 信号のアサートなし

1: UDC2 による usb\_reset 信号のアサートあり

uint32\_t

**USB\_Reset\_End**: 1



UDC2 が usb\_reset 信号をデアサートしたかどうかを表示

0: ビットのクリア後、UDC2 による usb\_reset 信号のデアサートなし

1: UDC2 による usb\_reset 信号のデアサートあり

uint32\_t

**Reserved1 :** 6

6 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**MW\_Set\_Add :** 1

マスタライト転送がディセーブル状態で、該当する Rx 用 EP にマスタライト転送されるべきデータがセットされると 1 にセットされます。

0: 未検出

1: マスタライト転送アドレス要求

uint32\_t

**MW\_End\_Add :** 1

マスタライト転送が終了した際に、1 にセットされます。

0: 未検出

1: マスタライト転送終了

uint32\_t

**MW\_TimeOut :** 1

マスタライト転送動作中に、タイムアウトした場合、本ステータスが 1 にセットされます。

0: 未検出

1: マスタライト転送タイムアウト

uint32\_t

**MW\_AHBErr :** 1

マスタライト転送動作中に、AHB エラーが発生した場合、本ステータスが 1 にセットされます。この割り込み発生後は、DMAC 設定レジスタの mw\_resetl によりマスタライト転送ブロックをリセットする必要があります。

0: 未検出

1: AHB エラー発生

uint32\_t

**MR\_End\_Add :** 1

マスタリード転送が終了した際に、1 にセットされます。

0: 未検出

1: マスタリード転送終了

uint32\_t

**MR\_EP\_DSet :** 1

マスタリード時で使用する、UDC2 Tx 用エンドポイントの FIFO がライト可能(Full ではない状態)となった時に、1 にセットされます。

0: FIFO ライト不可

1: FIFO ライト可

uint32\_t

**MR\_AHBErr :** 1

マスタライト転送動作中に、AHB エラーが発生した場合、本ステータスが1 にセットされます。この割り込み発生後は、DMAC 設定レジスタのmr\_resetビットによりマスタライト転送ブロックをリセットする必要があります。

- 0: 未検出
- 1: AHB エラー発生

uint32\_t

**UDC2\_Reg\_Read :** 1

UDC2 リード要求レジスタの設定により実行された UDC2 アクセスが完了して、UDC2 リード値レジスタに読み出した値が設定された時、1 がセットされます。また、DC2 内部レジスタへのライトアクセスが完了した時に、1 にセットされます。

- 0: 未検出
- 1: レジスタリード/ライト完了

uint32\_t

**DMAC\_Reg\_Read :** 1

DMAC 設定レジスタの設定により実行されたレジスタアクセスが完了して、DMAC リードレジスタに読み出した値がセットされたときに、1 にセットされます。

- 0: 未検出
- 1: レジスタリード完了

uint32\_t

**Reserved2 :** 2

2 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**PowerDetect :** 1

VBUSPOWER 入力のステータスが変化した時に、"1"にセットされます。

- 0: 変化無し
- 1: ステータス変化

uint32\_t

**MW\_ReadError :** 1

共通バスアクセスの設定中にエンドポイントへのアクセスによりマスタライト転送が開始された場合、1 が設定されます。(EPx\_Status レジスタの bus\_sel ビットは 0)

- 0: 未検出
- 1: マスタライトにエンドポイントリードエラー発生

uint32\_t

**Reserved3 :** 2

2 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

## 11.2.4.3 USBDMACConfig

メンバ:

uint32\_t

**All :** 下記すべてのビットの OR 値

**Bit**

uint32\_t

**MW\_Enable :** 1

マスタライト転送を制御します。転送アドレスのセット完了時にイネーブルにして下さい。マスタ転送の終了とともに、自動的にディセーブルされます。本レジスタではマスタライト動作のディセーブルを行うことはできませんのでマスタライト転送を停止させる際は、<mw\_abort>を使用して下さい。

0: 禁止

1: 許可

uint32\_t

**MW\_Abort :** 1

マスタライト転送を制御します。本ビットに 1 をセットすることによりマスタライト動作を停止させることができます。転送途中にアボートした場合、UDC2 からマスタライト用バッファへの転送を中断して<mw\_enable>がクリアされ、マスタライト転送は停止されます。

0: ノーオペレーション

1: アボート

uint32\_t

**MW\_Reset :** 1

マスタライト転送ブロックを初期化します。ただしエンドポイントの FIFO は初期化されませんので、本リセットとは別に UDC2 のコマンドレジスタへアクセスして、対応するエンドポイントの初期化を行う必要があります。

本リセットはマスタ動作を停止させてから使用して下さい。

本ビットを 1 へセット後、自動的に 0 にクリアされます。クリアされるまで次のマスタライト転送を行わないで下さい。

0: ノーオペレーション

1: リセット

uint32\_t

**Reserved1 :** 1

1 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**MR\_Enable :** 1

マスタリード転送を制御します。転送アドレスのセット完了時にイネーブルにして下さい。マスタ転送の終了とともに、自動的にディセーブルされます。本レジスタではマスタリード動作のディセーブルを行うことはできませんので、マスタリード転送を停止させる際は<mr\_abort>を使用して下さい。

0: 禁止

1: 許可

uint32\_t

**MR\_Abort :** 1

マスタリード転送を制御します。本ビットに 1 をセットすることによりマスタリード動作を停止させることができます。

転送途中にアボートした場合、マスタリード用バッファの UDC2 への転送を中断し<mr\_enable>がクリアされ、マスタリード転送は停止されます。

0: ノーオペレーション

1: アボート

uint32\_t

**MR\_Reset :** 1

マスタライト転送ブロックを初期化します。ただしエンドポイントの FIFO は初期化されませんので、本リセットとは別に UDC2 の UDFS2CMD へアクセスして、対応するエンドポイントの初期化を行う必要があります。

本リセットはマスタ動作を停止させてから使用して下さい。

本ビットを 1 ヘット後、自動的に 0 にクリアされます。クリアされるまで次のマスタライト転送を行わないで下さい。

0: ノーオペレーション

1: リセット

uint32\_t

**Reserved2 :** 1

1 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**M\_Burst\_Type :** 1

マスタライト/リード転送時のバースト転送実行時の HBURST[2:0]のタイプを選択します。UDC2AB が行うバースト転送のタイプは INCR8(8 ビート インクリメント式バースト)となります。従って、通常は初期値で

ある 0 を設定して下さい。但し、システムの AHB 仕様によりバースト転送のタイプとして INCR しか使用できない場合には、このビットに 1 を設定して下さい。この場合、UDC2AB は 8 ビートの INCR 転送を実行し

ます。なお、バースト転送のビット数を変更することはできません。

このビットの設定は UDC2AB への初期設定にて行って下さい。マスタライト/リード転送を開始してからは変更しないで下さい。

注) UDC2AB はマスタライト/リード転送でバースト転送のみを行うわけではなく、バースト転送とシングル転送を組み合わせで転送します。このビットはあくまでバースト転送実行時にのみ影響します。

0: INCR8

1: INCR

uint32\_t

**Reserved3 :** 23

23 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

## 11.2.4.4 USB\_DMACStatus

メンバ:

uint32\_t

**All :** 下記すべてのビットの OR 値

**Bit**

uint32\_t

**MW\_EP\_DSet :** 1

UDC2 の Rx 用エンドポイント へ受信データがセットされると 1 にセットされます。全データがマスタライト用 DMA により読み出されると 0 になります。

0: Endpoint 内にデータはありません

1: Endpoint 内に読み出すべきデータがあります

uint32\_t

**MR\_EP\_DSet: 1**

マスタリード DMA 転送により、UDC2 の Tx 用エンドポイントへ送信データがセットされ、Endpoint に書き込むスペースがなくなると 1 にセットされます。ホストからの IN-Token により UDC2 からデータが転送されると 0 になります。このビットが 0 であるときは Endpoint への DMA 転送が可能です。(本ビットは eptx\_dataset 入力信号を CLK\_H 同期したものです。)

0: Endpoint 内にデータを転送可能です。

1: Endpoint 内にデータを転送するスペースがありません。

uint32\_t

**MW\_Buf\_Empty: 1**

UDC2AB のマスタライト DMA 用バッファの空きを示します。

0: バッファにデータがあります。

1: バッファにデータはありません。

uint32\_t

**MR\_Buf\_Empty: 1**

UDC2AB のマスタリード DMA 用バッファの空きを示します。

0: バッファにデータがあります。

1: バッファにデータはありません。

uint32\_t

**MR\_EP\_Empty: 1**

UDC2Rx のエンドポイントに空きがあるかどうかを示します。UDC2 設定レジスタの tx0 ビットを使用しての NULL パケット送信時に 1 にセットされます。(本ビットは eptx\_dataset 入力信号を CLK\_H 同期したものです。)

0: Endpoint 内にデータがあります

1: Endpoint 内にデータがありません

uint32\_t

**Reserved: 27**

27 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

## 11.2.4.5 USB\_D\_PowerCtrl

メンバ:

uint32\_t

**All:** 下記すべてのビットの OR 値

**Bit**

uint32\_t

**USB\_Reset: 1**

UDC2 からの usb\_reset 信号を同期した値 (リードのみ)

0: usb\_reset = 0

1: usb\_reset = 1

uint32\_t

**PW\_Resetb: 1**

UDC2AB 用のソフトウェアリセットです。本ビットを 0 にセットする

ことで、PW\_RESETB 出力端子が 0 にアサートされます。  
マスタ動作が停止した状態でリセットを行って下さい。  
このビットは自動解除されませんので、必ずクリアして下さい。

- 0: リセットアサート
- 1: リセットデアサート

uint32\_t

**PW\_Detect:** 1

UDC2AB の VBUSPOWER 入力の状態を示します。

- 0: USB バスディスコネクト (VBUSPOWER = 0)
- 1: USB バスコネクト (VBUSPOWER = 1)

uint32\_t

**PHY\_Suspend:** 1

本ビットを 1 にセットすることで、PHYSUSPEND 出力信号が 0 へアサート (CLK\_H 同期)されます。PHY をサスペンドする時の端子として使用可能です。  
本ビットを 1 にセットすると、UDC2 レジスタと DMAC Read Request レジスタへのアクセスが禁止となります。

レジューム時(UDC2 の suspend\_x デアサート時)に自動的に 0 にクリアされます。

- 0: 非サスペンド状態
- 1: サスペンド状態

uint32\_t

**Suspend\_x:** 1

サスペンド信号を検出します(UDC2 からの suspend\_x 信号を同期化した値です)。(リードのみ)

- 0: サスペンド状態 (suspend\_x = 0)
- 1: 非サスペンド状態 (suspend\_x = 1)

uint32\_t

**PHY\_Resetb:** 1

本ビットを 0 にセットすることで、PW\_RESETB 出力端子が 0 にアサートされます。  
PHYRESET 信号は PHY のリセットに使用されます。自動的に解除されませんので、PHY のリセット時間経過後に必ずクリアしてください。

- 0: リセットアサート
- 1: リセットデアサート

uint32\_t

**PHY\_Remote\_Wakeup:** 1

USB のリモートウェイクアップ機能を実行するために使用します。本ビットに 1 をセットすることで、udc2\_wakeup 出力信号(UDC2 の wakeup 入力端子)を 1 にアサートすることができます。但し、UDC2 がサスペンドを検出していない時 (<suspend\_x>= 1 の時)に本ビットを 1 にセットした場合は無視されます(1 にセットされません)ので、サスペンド検出時にのみセットして下さい。USB レジューム完了時(<suspend\_x>デアサート時)に自動的に 0 にクリアされます。

- 0: ノーオペレーション
- 1: ウェイクアップ

uint32\_t

**Wakeup\_En:** 1

USB のサスペンド時に、本デバイスを低消費電力モードに移行させて CLK\_H を停止する時には、本ビットを 1 にセットしてください。

本ビットを 1 にセットしているとサスペンドが解除された時(<suspend\_x>=1)に、WAKEUP 信号が非同期で 0 にアサートされますので、INTUSBWKUP による本デバイスの低消費電力モードからの復帰に利用可能です。

0: WAKEUPn 信号をアサートしない

1: WAKEUPn 信号をアサートする

uint32\_t

**Reserved:** 24

24 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

## 11.2.4.6 USBD\_EP0Status

メンバ:

uint32\_t

**All:** 下記すべてのビットの OR 値

**Bit**

uint32\_t

**Reserved1:** 9

9 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**Status:** 3

EP0 の現在のステイタスを示します。Setup-Token 受信後、“Ready”にクリアされます。

000: Ready (通常の状態)

001: Busy (STATUS-Stage で“NAK”を受信した際にセットされます)

010: Error (受信データが CRC エラーの場合、およびデータ送信後タイムアウトした際にセットされます)

011: Control-RD 転送において Length 以上のデータを要求された場合に“STALL”を返信し、status がセットされます。また、コマンドレジスタにより“EP0-STALL”を発行した場合もセットされます

100 ~ 111: Reserved

uint32\_t

**Toggle:** 2

現在の EP0 のトグル値を示します。

00: DATA0

01: DATA1

10: Reserved

11: Reserved

uint32\_t

**Reserved2:** 1

1 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**EP0\_Mask:** 1

Setup-Token 受信後、1 にセットされます。"Setup\_Received"コマンドを発行することにより 0 にクリアされます。この bit が 1 の間は、EP0\_FIFO への書き込みが行われません。

0: EP0\_FIFO への書き込み可  
1: EP0\_FIFO への書き込み不可

uint32\_t

**Reserved3:** 16

16 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

## 11.2.4.7 USBD\_EPxConfig, USBD\_EPxStatus

メンバ:

uint32\_t

**All:** 下記すべてのビットの OR 値

**Bit**

uint32\_t

**Num\_MF:** 2

Isochronous 転送を選択した場合、μ フレーム中に何回転送をするかを設定します。

00: 1-transaction  
01: 2-transaction  
10: 3-transaction  
11: Reserved

uint32\_t

**T\_Type:** 2

このエンドポイントの転送モードを設定します。

00: Control  
01: Isochronous  
10: Bulk  
11: Interrupt

uint32\_t

**Reserved1:** 3

3 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

uint32\_t

**Dir:** 1

このエンドポイントに対する転送方向を設定します

0: OUT (Host-to-device)  
1: IN (Device-to-host)

注:

エンドポイント 1,3,5,7 は必ず '1' に設定 (IN のみ)  
エンドポイント 2,4,6 は必ず '0' に設定 (OUT のみ)

uint32\_t

**Disable:** 1

EPx の転送許可状態を示します。"禁止"状態にある場合、このエンドポイントに対する Token に対しては "NAK" を返信し続けます。



0: 許可  
1: 禁止

uint32\_t

**Status:** 3

現在の EPx の状態を示します。UDFS2CMD より EP\_Reset を発行することにより status は"Ready"となります。

000: Ready (通常の状態)

001: Reserved

010: エラー(データパケットに受信エラーが発生した時、または送信後タイムアウトが発生した時にセットされます。但し、"Stall"、"Invalid"がセットされている場合にはセットされません)

011: Stall (Command register により"EP\_Stall"を発行した場合にセットされます)

100 ~110: Reserved

uint32\_t

**Toggle:** 2

現在の EPx のトグル値を示します。

00: DATA0

01: DATA1

10: DATA2

11: MDATA

uint32\_t

**Bus\_Sel:** 1

EP1 の FIFO へのアクセスをするバスを選択します。

0: 共通バスアクセス

1: 直接アクセス

uint32\_t

**Pkt\_Mode:** 1

EPx のパケットモードを選択します。Dual モードを選択することにより、EPx に対する 2 つのパケットデータを保持することが可能となります。

0: Single モード

1: Dual モード

uint32\_t

**Reserved2:** 16

16 ビットのギャップを未使用エリアに配置します。未検出の場合、ゼロが書かれます。

## 12. WDT

### 12.1 概要

ウォッチドッグタイマ(WDT)は、ノイズなどの原因によりCPU が誤動作(暴走)を始めた場合、これを検出し正常な状態に戻すことを目的としています。

WDTドライバの API は、検出時間、カウンタのオーバーフロー時の出力、IDLE モードでの動作設定などの引数等、ウォッチドッグタイマの設定を行う関数を提供します。

本ドライバは、以下のファイルで構成されています。

\\Libraries\\TX03\_Periph\_Driver\\src\\tmpm365\_wdt.c  
\\Libraries\\TX03\_Periph\_Driver\\inc\\tmpm365\_wdt.h

### 12.2 API 関数

#### 12.2.1 関数一覧

- ◆ void WDT\_SetDetectTime(uint32\_t **DetectTime**)
- ◆ void WDT\_SetIdleMode(FunctionalState **NewState**)
- ◆ void WDT\_SetOverflowOutput(uint32\_t **OverflowOutput**)
- ◆ void WDT\_Init(WDT\_InitTypeDef \* **InitStruct**)
- ◆ void WDT\_Enable(void)
- ◆ void WDT\_Disable(void)
- ◆ void WDT\_WriteClearCode(void)

#### 12.2.2 関数の種類

関数は、主に以下の 2 種類に分かれています。

- 1) ウォッチドッグタイマ設定:  
WDT\_SetDetectTime(), DT\_SetOverflowOutput(), WDT\_Init(), WDT\_Enable(),  
WDT\_Disable(), WDT\_WriteClearCode()
- 2) IDLE モード時の開始・停止:  
WDT\_SetIdleMode()

#### 12.2.3 関数仕様

##### 12.2.3.1 WDT\_SetDetectTime

WDT 検出時間の設定

関数のプロトタイプ宣言:

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

引数:

**DetectTime**: 以下から検出時間を選択します。

- WDT\_DETECT\_TIME\_EXP\_15: 2<sup>15</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_17: 2<sup>17</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_19: 2<sup>19</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_21: 2<sup>21</sup>/fsys

- WDT\_DETECT\_TIME\_EXP\_23: 2<sup>23</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_25: 2<sup>25</sup>/fsys

**機能:**

WDT の検出時間を設定します。

**戻り値:**

なし

## 12.2.3.2 WDT\_SetIdleMode

IDLE 時の動作選択

**関数のプロトタイプ宣言:**

```
void  
WDT_SetIdleMode(FunctionalState NewState)
```

**引数:**

**NewState**: 以下から IDLE 時の WDT 動作を選択します。

- **ENABLE**: 動作
- **DISABLE**: 停止

**機能:**

本関数は、IDLE モード時の WDT カウンタの動作を設定します。

**NewState** が **ENABLE** の時は WDT カウンタ停止

**NewState** が **DISABLE** の時は WDT カウンタ作動

**補足:**

CPU が IDLE モードに入る前に、引数を選択して本関数を呼び出してください。

**戻り値:**

なし

## 12.2.3.3 WDT\_SetOverflowOutput

暴走検出後の動作選択

**関数のプロトタイプ宣言:**

```
void  
WDT_SetOverflowOutput(uint32_t OverflowOutput)
```

**引数:**

**OverflowOutput**: 以下から暴走検出後の動作を選択します。

- **WDT\_NMIINT**: INTWDT 割り込み要求を発生します。
- **WDT\_WDOUT**: マイコンをリセットします。

**機能:**

カウンタオーバーフロー時の NMI 割り込み/リセットの設定を行います。  
**OverflowOutput** が **WDT\_NMIINT** の時、カウンタオーバーフローが発生すると NMI 割り込みが発生し、**OverflowOutput** が **WDT\_WDOUT** の時、カウンタオーバーフローが発生するとリセットが発生します。

戻り値:  
なし

## 12.2.3.4 WDT\_Init

WDT の初期化

関数のプロトタイプ宣言:

void  
WDT\_Init (WDT\_InitTypeDef\* **InitStruct**)

引数:

**InitStruct**: カウンタオーバーフロー発生時の WDT 検出時間、WDT 出力の設定を含む WDT 設定に関する構造体。(詳細は“データ構造:”を参照)

機能:

カウンタオーバーフロー発生時の WDT 検出時間、WDT 出力の設定を含む WDT 初期設定。**WDT\_SetDetectTime()**, **WDT\_SetOverflowOutput()** が呼び出されます。

戻り値:  
なし

## 12.2.3.5 WDT\_Enable

WDT 動作の許可

関数のプロトタイプ宣言:

void  
WDT\_Enable(void)

引数:

なし

機能:

WDT 動作を許可します。

戻り値:  
なし

## 12.2.3.6 WDT\_Disable

WDT 動作の禁止

関数のプロトタイプ宣言:

void  
WDT\_Disable(void)

引数:

なし

**機能:**

WDT 動作を禁止します。

**戻り値:**

なし

## 12.2.3.7 WDT\_WriteClearCode

クリアコードの書き込み

**関数のプロトタイプ宣言:**

void

WDT\_WriteClearCode (void)

**引数:**

なし

**機能:**

クリアコードをライトします。

**戻り値:**

なし

## 12.2.4 データ構造

### 12.2.4.1 WDT\_InitTypeDef

**メンバ:**

uint32\_t

**DetectTime** 以下から検出時間を選択します。

- WDT\_DETECT\_TIME\_EXP\_15: 2<sup>15</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_17: 2<sup>17</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_19: 2<sup>19</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_21: 2<sup>21</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_23: 2<sup>23</sup>/fsys
- WDT\_DETECT\_TIME\_EXP\_25: 2<sup>25</sup>/fsys

uint32\_t

**OverflowOutput** 以下から、カウンタオーバーフロー時の動作を選択します。

- WDT\_WDOUT: マイコンをリセットします。
- WDT\_NMIINT: INTNMI 割り込み要求を発生します。