

# **TOSHIBA**

## **TX04 ペリフェラルドライバ ユーザーガイド (TMPM440)**

**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。

## 目次

1.	はじめに .....	1
2.	TX04 ペリフェラルドライバの構成 .....	2
3.	ADC .....	3
3.1	概要 .....	3
3.2	API 関数 .....	4
3.2.1	関数一覧 .....	4
3.2.2	関数の種類 .....	4
3.2.3	関数仕様 .....	5
3.2.4	データ構造 .....	20
4.	CG .....	23
4.1	概要 .....	23
4.2	API 関数 .....	23
4.2.1	関数一覧 .....	23
4.2.2	関数の種類 .....	24
4.2.3	関数仕様 .....	25
4.2.4	データ構造 .....	52
5.	DAC .....	53
5.1	概要 .....	53
5.2	API 関数 .....	53
5.2.1	関数一覧 .....	53
5.2.2	関数の種類 .....	53
5.2.3	関数仕様 .....	53
5.2.4	データ構造 .....	55
6.	DMAC .....	56
6.1	概要 .....	56
6.2	API 関数 .....	56
6.2.1	関数一覧 .....	56
6.2.2	関数の種類 .....	57
6.2.3	関数仕様 .....	57
6.2.4	データ構造 .....	71
7.	EPHC .....	77
7.1	概要 .....	77
7.2	API 関数 .....	77
7.2.1	関数一覧 .....	77
7.2.2	関数の種類 .....	78
7.2.3	関数仕様 .....	78
7.2.4	データ構造 .....	88
8.	ESIO .....	92
8.1	概要 .....	92
8.2	API 関数 .....	92
8.2.1	関数一覧 .....	92

8.2.2	関数の種類 .....	93
8.2.3	関数仕様 .....	93
8.2.4	データ構造 .....	106
9.	EXB .....	110
9.1	概要 .....	110
9.2	API 関数 .....	110
9.2.1	関数一覧 .....	110
9.2.2	関数の種類 .....	110
9.2.3	関数仕様 .....	111
9.2.4	データ構造 .....	114
10.	FC .....	117
10.1	概要 .....	117
10.2	API 関数 .....	117
10.2.1	関数一覧 .....	117
10.2.2	関数の種類 .....	117
10.2.3	関数仕様 .....	118
10.2.4	データ構造 .....	123
11.	FUART .....	124
11.1	概要 .....	124
11.2	API 関数 .....	124
11.2.1	関数一覧 .....	124
11.2.2	関数の種類 .....	125
11.2.3	関数仕様 .....	125
11.2.4	データ構造 .....	136
12.	GPIO .....	139
12.1	概要 .....	139
12.2	API 関数 .....	139
12.2.1	関数一覧 .....	139
12.2.2	関数の種類 .....	140
12.2.3	関数仕様 .....	140
12.2.4	データ構造 .....	161
13.	KSCAN .....	163
13.1	概要 .....	163
13.2	API 関数 .....	163
13.2.1	関数一覧 .....	163
13.2.2	関数の種類 .....	163
13.2.3	関数仕様 .....	164
13.2.4	データ構造 .....	172
14.	KWUP .....	173
14.1	概要 .....	173
14.2	API 関数 .....	173
14.2.1	関数一覧 .....	173
14.2.2	関数の種類 .....	173

14.2.3	関数仕様 .....	173
14.2.4	データ構造 .....	176
15.	PHC.....	178
15.1	概要.....	178
15.2	API 関数 .....	178
15.2.1	関数一覧 .....	178
15.2.2	関数の種類.....	178
15.2.3	関数仕様 .....	179
15.2.4	データ構造 .....	186
16.	RTC.....	188
16.1	概要.....	188
16.2	API 関数 .....	188
16.2.1	関数一覧 .....	188
16.2.2	関数の種類.....	189
16.2.3	関数仕様 .....	189
16.2.4	データ構造 .....	209
17.	SBI.....	212
17.1	概要.....	212
17.2	API 関数 .....	212
17.2.1	関数一覧 .....	212
17.2.2	関数の種類.....	213
17.2.3	関数仕様 .....	213
17.2.4	データ構造 .....	220
18.	TMRB.....	222
18.1	概要.....	222
18.2	API 関数 .....	222
18.2.1	関数一覧 .....	222
18.2.2	関数の種類.....	223
18.2.3	関数仕様 .....	223
18.2.4	データ構造 .....	236
19.	TMRC.....	238
19.1	概要.....	238
19.2	API 関数 .....	238
19.2.1	関数一覧 .....	238
19.2.2	関数の種類.....	238
19.2.3	関数仕様 .....	239
19.2.4	データ構造 .....	245
20.	TMRD.....	248
20.1	概要.....	248
20.2	API 関数 .....	248
20.2.1	関数一覧 .....	248
20.2.2	関数の種類.....	249
20.2.3	関数仕様 .....	250

	20.2.4 データ構造 .....	263
21.	UART.....	266
21.1	概要 .....	266
21.2	API 関数 .....	266
21.2.1	関数一覧 .....	266
21.2.2	関数の種類 .....	267
21.2.3	関数仕様 .....	267
21.2.4	データ構造 .....	286
22.	WDT .....	289
22.1	概要 .....	289
22.2	API 関数 .....	289
22.2.1	関数一覧 .....	289
22.2.2	関数の種類 .....	289
22.2.3	関数仕様 .....	289
22.2.4	データ構造 .....	293
23.	PSC.....	294
23.1	概要 .....	294
23.2	API 関数 .....	294
23.2.1	関数一覧 .....	294
23.2.2	関数の種類 .....	295
23.2.3	関数仕様 .....	296
23.2.4	データ構造 .....	319

## 1. はじめに

本ソフトウェアは、東芝製 TX04 シリーズマイコン TMPM440 用ペリフェラルドライバセットです。

TX04 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。

TMPM440 ペリフェラルドライバは以下の仕様に基づいています。

- 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。
- すべての周辺機能をカバーしています。

## 2. TX04 ペリフェラルドライバの構成

### **/Libraries**

TX04 CMSIS ファイルと TPM440 ペリフェラルドライバが格納されています。

### **/Libraries/ TX04\_CMSIS**

このフォルダには TPM440 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/TX04\_Periph\_Driver**

TPM440 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/TX04\_Periph\_Driver/inc**

TPM440 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/TX04\_Periph\_Driver/src**

TPM440 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**

TPM440 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**

TPM440 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**

TPM440 ペリフェラルドライバの使用例が格納されています。

### **/Utilities/TPM440-EVAL**

TPM440 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)を内蔵しています。本 AD コンバータは、ユニット A とユニット B に 8 チャンネルのアナログ入力をそれぞれ持ち、ユニット C に 4 チャンネルのアナログ入力を持っています。これらのアナログ入力チャンネルは、専用入力ポートです。

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(ADTRGAn, ADTRGBn, ADTRGSNCn)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

(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  $f_c$  または  $f_{PLLAD}$  から選択し、ADC 内部のプリスケアラにて 1/2～1/16 に分周可能

(8) AD コンバータ回路 ON/OFF 機能

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。

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。

/Libraries/TX04\_Periph\_Driver/src\tmpm440\_adc.c

/Libraries/TX04\_Periph\_Driver/inc\tmpm440\_adc.h

## 3.2 API 関数

### 3.2.1 関数一覧

- ◆ void ADC\_SWReset(TSB\_AD\_TypeDef \* ADx);
- ◆ void ADC\_SetClk(TSB\_AD\_TypeDef \* ADx, uint32\_t Sample\_HoldTime, uint32\_t Prescaler\_Output);
- ◆ void ADC\_Start(TSB\_AD\_TypeDef \* ADx);
- ◆ void ADC\_SetScanMode(TSB\_AD\_TypeDef \* ADx, FunctionalState NewState);
- ◆ void ADC\_SetRepeatMode(TSB\_AD\_TypeDef \* ADx, FunctionalState NewState);
- ◆ void ADC\_SetINTMode(TSB\_AD\_TypeDef \* ADx, uint8\_t INTMode);
- ◆ void ADC\_SetInputChannel(TSB\_AD\_TypeDef \* ADx, uint8\_t InputChannel);
- ◆ void ADC\_SetScanChannel(TSB\_AD\_TypeDef \* ADx, uint8\_t StartChannel, uint8\_t Range);
- ◆ void ADC\_SetVrefCut(TSB\_AD\_TypeDef \* ADx, uint8\_t VrefCtrl);
- ◆ void ADC\_SetIdleMode(TSB\_AD\_TypeDef \* ADx, FunctionalState NewState);
- ◆ void ADC\_SetVref(TSB\_AD\_TypeDef \* ADx, FunctionalState NewState);
- ◆ void ADC\_SetInputChannelTop(TSB\_AD\_TypeDef \* ADx, uint8\_t TopInputChannel);
- ◆ void ADC\_StartTopConvert(TSB\_AD\_TypeDef \* ADx);
- ◆ void ADC\_SetMonitor(TSB\_AD\_TypeDef \* ADx, ADC\_CMPCRx ADCMPx, FunctionalState NewState);
- ◆ void ADC\_ConfigMonitor(TSB\_AD\_TypeDef \* ADx, ADC\_CMPCRx ADCMPx, ADC\_MonitorTypeDef \* Monitor);
- ◆ void ADC\_SetHWTrg(TSB\_AD\_TypeDef \* ADx, uint8\_t HwSource, FunctionalState NewState);
- ◆ void ADC\_SetHWTrgTop(TSB\_AD\_TypeDef \* ADx, uint8\_t HwSource, FunctionalState NewState);
- ◆ ADC\_State ADC\_GetConvertState(TSB\_AD\_TypeDef \* ADx);
- ◆ ADC\_Result ADC\_GetConvertResult(TSB\_AD\_TypeDef \* ADx, uint8\_t ADREGx);
- ◆ ADC\_SetDMAReq(TSB\_AD\_TypeDef \* **ADx**, 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\_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(TSB_AD_TypeDef * ADx)
```

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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(TSB_AD_TypeDef * ADx,  
            uint32_t Sample_HoldTime,  
            uint32_t Prescaler_Output)
```

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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>
- **ADC\_CONVERSION\_CLK\_160:** 160x <ADCLK>
- **ADC\_CONVERSION\_CLK\_320:** 320x <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\_2:**  $f_c / 2$
- **ADC\_FC\_DIVIDE\_LEVEL\_4:**  $f_c / 4$
- **ADC\_FC\_DIVIDE\_LEVEL\_8:**  $f_c / 8$
- **ADC\_FC\_DIVIDE\_LEVEL\_16:**  $f_c / 16$

**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。

**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(TSB\_AD\_TypeDef \* **ADx**)

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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**(TSB\_AD\_TypeDef \* **ADx**, FunctionalState **NewState**)

**引数:**

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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(TSB\_AD\_TypeDef \* **ADx**, FunctionalState **NewState**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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(TSB\_AD\_TypeDef \* **ADx**, uint8\_t **INTMode**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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

アナログ入力チャンネルの選択

**関数のプロトタイプ宣言:**

void

**ADC\_SetInputChannel(TSB\_AD\_TypeDef \* *ADx*, uint8\_t *InputChannel*)**

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

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

- ユニット A、またはユニット B の場合:  
**ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,**  
**ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07**
- ユニット C の場合:  
**ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03**

**機能:**

アナログ入力チャンネルを選択します。

**補足:**

通常変換入力の場合 1 チャンネルのみ選択できます。

戻り値:  
なし

### 3.2.3.8 ADC\_SetScanChannel

スキャンチャンネルの設定

関数のプロトタイプ宣言:

```
void
ADC_SetScanChannel(TSB_AD_TypeDef * ADx,
                   uint8_t StartChannel,
                   uint8_t Range)
```

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

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

- ユニット A、またはユニット B の場合:
  - ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,**
  - ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07**
- ユニット C の場合:
  - ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03**

**Range**: チャンネルスキャンの範囲を設定します。

ユニット A またはユニット B の場合、1~8 を、ユニット C の場合 1~4 を選択できます。

機能:

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

補足:

設定可能なチャン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 ~ 8
ADC_AN_01	1 ~ 7
ADC_AN_02	1 ~ 6
ADC_AN_03	1 ~ 5
ADC_AN_04	1 ~ 4
ADC_AN_05	1 ~ 3
ADC_AN_06	1 ~ 2



ADC_AN_07	1 ~ 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(TSB\_AD\_TypeDef \* **ADx**, uint8\_t **VrefCtrl**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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(TSB\_AD\_TypeDef \* **ADx**, FunctionalState **NewState**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A

- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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(TSB\_AD\_TypeDef \* **ADx**, FunctionalState **NewState**)

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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:**

STOP1/STOP2 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(TSB\_AD\_TypeDef \* **ADx**, uint8\_t **TopInputChannel**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

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

- ユニット A、またはユニット B の場合:  
ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,  
ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07
- ユニット C の場合:  
ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03

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(TSB\_AD\_TypeDef \* **ADx**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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(TSB_AD_TypeDef * ADx,  
               ADC_CMPCRx ADCMPx,  
               FunctionalState NewState)
```

**引数:**

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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(大小判定カウンタ数はクリア)

**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(TSB_AD_TypeDef * ADx,  
                  ADC_CMPCRx ADCMPx,  
                  ADC_MonitorTypeDef * Monitor)
```

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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ンネルが含まれます。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。

**ADCMP<sub>x</sub>** 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(TSB_AD_TypeDef * ADx,  
              uint8_t HwSource,  
              FunctionalState NewState)
```

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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:**

ADC\_EXT\_TRG は 2 端子(ADTRGx、ADTRGSNC)あります。

ADGGSNC (PH3)端子は、ユニット A とユニット B で共通で、両ユニットを同時に起動することが可能です。

➤ **DC\_MATCH\_TMRB\_NORMAL:**

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。

ユニット A の通常 AD 変換の場合のタイマは TB17RG0 です。

ユニット B の通常 AD 変換の場合のタイマは TB18RG0 です。

ユニット C の通常 AD 変換の場合のタイマは TB19RG0 です。

**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(TSB\_AD\_TypeDef \* **ADx**,  
uint8\_t **HwSource**,

FunctionalState **NewState**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**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**:

ADC\_EXT\_TRG は 2 端子(ADTRGx、ADTRGSNC)あります。

ADGGSNC (PH3)端子は、ユニット A とユニット B で共通で、両ユニットを同時に起動することが可能です。

- **ADC\_MATCH\_TMRB\_TOP**:

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 一致割り込みによるで起動することが可能です。

ユニット A の最優先 AD 変換の場合のタイマは TB14RG0 です。

ユニット B の最優先 AD 変換の場合のタイマは TB15RG0 です。

ユニット C の最優先 AD 変換の場合のタイマは TB16RG0 です。

**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。

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(TSB\_AD\_TypeDef \* **ADx**)

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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(TSB\_AD\_TypeDef \* **ADx**,  
uint8\_t **ADREGx**)

**引数:**

**ADx:** AD 変換のユニットを指定します。

- **TSB\_ADA:** ユニット A
- **TSB\_ADB:** ユニット B
- **TSB\_ADC:** ユニット C

**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\_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

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
1 回毎、割り込み発生	ADC_REG_00
2 回毎、割り込み発生	ADC_REG_00 ~ ADC_REG_01
3 回毎、割り込み発生	ADC_REG_00 ~ ADC_REG_02
4 回毎、割り込み発生	ADC_REG_00 ~ ADC_REG_03
5 回毎、割り込み発生	ADC_REG_00 ~ ADC_REG_04
6 回毎、割り込み発生	ADC_REG_00 ~ ADC_REG_05
7 回毎、割り込み発生	ADC_REG_00 ~ ADC_REG_06
8 回毎、割り込み発生	ADC_REG_00 ~ 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	15 channels	ADC_REG_00 ~ ADC_REG_07
ADC_AN_01	14 channels	ADC_REG_01 ~ ADC_REG_07
ADC_AN_02	13 channels	ADC_REG_02 ~ ADC_REG_07
ADC_AN_03	12 channels	ADC_REG_03 ~ ADC_REG_07
ADC_AN_04	11 channels	ADC_REG_04 ~ ADC_REG_07
ADC_AN_05	10 channels	ADC_REG_05 ~ ADC_REG_07
ADC_AN_06	9 channels	ADC_REG_06 ~ ADC_REG_07
ADC_AN_07	8 channels	ADC_REG_07 ~ ADC_REG_07

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:

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

**ADRF** (Bit 12) : AD 変換結果格納フラグ

**ADOVRF** (Bit 13) : オーバーランフラグ

**ADRF\_MR** (Bit 16) **ADRF** ミラーレジスタ

**ADOVRF\_MR** (Bit 17) **ADOVRF** ミラーレジスタ

**ADR\_MR** (Bit 20 ~ Bit 31) **ADR** ミラーレジスタ

### 3.2.3.1 ADC\_SetDMAReq

DMA 起動要因の許可/禁止

関数のプロトタイプ:

void

ADC\_SetDMAReq(TSB\_AD\_TypeDef \* **ADx**, uint8\_t **DMAReq**,  
FunctionalState **NewState**)

引数:

**ADx**: AD 変換のユニットを指定します。

- **TSB\_ADA**: ユニット A
- **TSB\_ADB**: ユニット B
- **TSB\_ADC**: ユニット C

**DMAReq**: 以下から DMA 要求を行う AD 変換割り込みのタイプを選択します。

- **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 要求の許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

通常 AD 変換割り込み、または最優先 AD 変換割り込みをトリガに DMAC を起動します。

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ンネルを選択します:

ADC\_AN\_00, ADC\_AN\_01, ADC\_AN\_02, ADC\_AN\_03,  
ADC\_AN\_04, ADC\_AN\_05, ADC\_AN\_06, ADC\_AN\_07,

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) : reserved

uint32\_t

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

uint32\_t

**Reserved1** (Bit 2) : reserved

uint32\_t

**TopComplete** (Bit 3) : 最優先 AD 変換終了フラグ

uint32\_t

**Reserved2** (Bit 4 ~ Bit 31): reserved

## 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

**Reserved** (Bit 14 ~ Bit 15): reserved

uint32\_t

**Stored\_MR** (Bit 16) : **Stored** のミラー

uint32\_t

**OverRun\_MR** (Bit 17) : **OverRun** のミラー

uint32\_t

**Reserved** (Bit 18 ~ Bit 19) : reserved

uint32\_t

**ADResult\_MR** (Bit 20~Bit 31): **ADResult.**のミラー

## 4. CG

### 4.1 概要

本 CG API は TMPM440 CG において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ム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/TX04\_Periph\_Driver¥src¥tmpM440\_cg.c

/Libraries/TX04\_Periph\_Driver¥inc¥tmpM440\_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**)
- ◆ Result CG\_SetPLL(FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetPLLState(void)
- ◆ Result CG\_SetFPLLForADCValue(uint32\_t **NewValue**)
- ◆ Result CG\_SetFosc(CG\_FoscSrc **Source**, FunctionalState **NewState**)
- ◆ void CG\_SetFadcSrc(CG\_FadcSrc **FadcSrc**)
- ◆ void CG\_SetPLL1ForADC(FunctionalState **NewState**)
- ◆ void CG\_SetFoscSrc(CG\_FoscSrc **Source**)
- ◆ CG\_FoscSrc CG\_GetFoscSrc(void)
- ◆ FunctionalState CG\_GetFoscState(CG\_FoscSrc **Source**)
- ◆ Result CG\_SetFs(FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetFsState(void)
- ◆ void CG\_SetSTBYMode(CG\_STBYMode **Mode**)
- ◆ CG\_STBYMode CG\_GetSTBYMode(void)
- ◆ void CG\_SetPinStateInStop1Mode(FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetPinStateInStop1Mode(void)
- ◆ void CG\_SetPortKeepInStop2Mode(FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetPortKeepInStop2Mode(void)
- ◆ Result CG\_SetFcSrc(CG\_FcSrc **Source**)
- ◆ CG\_FcSrc CG\_GetFcSrc(void)
- ◆ void CG\_SetFtmrdSrc(CG\_TmrdUnit **TmrdUnit**, CG\_FtmrdSrc **FtmrdSrc**)
- ◆ CG\_FtmrdSrc CG\_GetFtmrdSrc(CG\_TmrdUnit **TmrdUnit**)
- ◆ void CG\_SetTMRDClk(CG\_TmrdUnit **TmrdUnit**, FunctionalState **NewState**)
- ◆ FunctionalState CG\_GetTMRDClkState(CG\_TmrdUnit **TmrdUnit**)
- ◆ 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\_ResetFlag CG\_GetResetFlag(void)
- ◆ void CG\_SetPeriphClkSupply(uint32\_t **Periph**)
- ◆ void CG\_SetFclkPeriphA(uint32\_t **Periph**, FunctionalState **NewState**)
- ◆ void CG\_SetFclkPeriphB(uint32\_t **Periph**, FunctionalState **NewState**)
- ◆ void CG\_SetFcPeriphA(uint32\_t **Periph**, FunctionalState **NewState**)
- ◆ void CG\_SetFcPeriphB(uint32\_t **Periph**, FunctionalState **NewState**)

## 4.2.2 関数の種類

上記関数は以下の 4 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\_SetPLL(), CG\_GetPLLState(), CG\_SetFosc(), CG\_SetFoscSrc(),  
CG\_GetFoscSrc(), CG\_GetFoscState(), CG\_SetFcSrc(), CG\_GetFcSrc(),  
CG\_SetProtectCtrl(), CG\_SetFclkPeriphA(), CG\_SetFclkPeriphB(), CG\_SetFcPeriphA(),  
CG\_SetFcPeriphB(), CG\_SetFPLLForADCValue(), CG\_SetPLL1ForADC(),  
CG\_SetFadcSrc(), CG\_SetFs(), CG\_GetFsState()

2) スタンバイモードの設定:

CG\_SetSTBYMode(), CG\_GetSTBYMode(),  
CG\_SetPinStateInStop1Mode(), CG\_GetPinStateInStop1Mode(),  
CG\_SetPortKeepInStop2Mode(), CG\_GetPortKeepInStop2Mode()

3) 割り込みの設定:

CG\_SetSTBYReleaseINTSrc(), CG\_GetSTBYReleaseINTState(), CG\_ClearINTReq(),  
CG\_GetResetFlag()

4) その他周辺機能へのクロック制御:

CG\_SetPeriphClkSupply(), CG\_SetFtmrdSrc(), CG\_GetFtmrdSrc(), CG\_SetTMRDClk(),  
CG\_GetTMRDClkState()

## 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

PhiT0( $\Phi T0$ ), 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\_FC\_DIVIDE\_4**: fc/4 に設定
- **CG\_SCOUT\_FC\_DIVIDE\_8**: fc/8 に設定
- **CG\_SCOUT\_FOSC**: fosc に設定
- **CG\_SCOUT\_LOW**: "Low"を出力に設定

**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 出力ソースクロック設定:

- **CG\_SCOUT\_FC\_DIVIDE\_4**: fc/4 に設定
- **CG\_SCOUT\_FC\_DIVIDE\_8**: fc/8 に設定
- **CG\_SCOUT\_FOSC**: fosc に設定
- **CG\_SCOUT\_LOW**: "Low"を出力に設定

#### 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:

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

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

PLL0 (fsys 用)の通倍数を設定。

関数のプロトタイプ宣言:  
Result  
CG\_SetFPLLValue(CG\_FpllValue **newValue**)

引数:  
**newValue**:

- **CG\_FPLL0\_IN8\_OUT40**: 入力クロック 8MHz、出力クロック 40MHz(5 通倍)
- **CG\_FPLL0\_IN8\_OUT48**: 入力クロック 8MHz、出力クロック 48MHz(6 通倍)
- **CG\_FPLL0\_IN8\_OUT64**: 入力クロック 8MHz、出力クロック 64MHz(8 通倍)
- **CG\_FPLL0\_IN8\_OUT80**: 入力クロック 8MHz、出力クロック 80MHz(10 通倍)
- **CG\_FPLL0\_IN10\_OUT50**: 入力クロック 10MHz、出力クロック 50MHz(5 通倍)
- **CG\_FPLL0\_IN10\_OUT60**: 入力クロック 10MHz、出力クロック 60MHz(6 通倍)
- **CG\_FPLL0\_IN10\_OUT80**: 入力クロック 10MHz、出力クロック 80MHz(8 通倍)
- **CG\_FPLL0\_IN10\_OUT100**: 入力クロック 10MHz、出力クロック 100MHz(10 通倍)

機能:  
PLL0 (fsys 用)の通倍数を設定します。

戻り値:

**SUCCESS:** 成功

**ERROR:** 失敗

## 4.2.3.13 CG\_SetPLL

PLL 0 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

## 4.2.3.14 CG\_GetPLLState

PLL0 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

## 4.2.3.15 CG\_SetFPLLForADCValue

PLL1 (ADC 用)の通倍数を設定

関数のプロトタイプ宣言:

Result

CG\_SetFPLLForADCValue(uint32\_t **NewValue**)

引数:

**NewValue**: 以下から PLL の通倍数を選択します。

- **CG\_FPLL1\_IN8\_OUT64**: 入力クロック 8MHz, 出力クロック 64MHz(6 通倍)
- **CG\_FPLL1\_IN8\_OUT80**: 入力クロック 8MHz, 出力クロック 80MHz(10 通倍)
- **CG\_FPLL1\_IN10\_OUT80**: 入力クロック 10MHz, 出力クロック 80MHz(8 通倍)
- **CG\_FPLL1\_IN10\_OUT100**: 入力クロック 10MHz, 出力クロック 100MHz(10 通倍)

機能:

PLL1 (ADC 用)の通倍数を設定します。

戻り値:

**SUCCESS**: 成功

**ERROR**: 失敗

## 4.2.3.16 CG\_SetFadcSrc

AD コンバーターのクロックソース設定

関数のプロトタイプ宣言:

void

CG\_SetFadcSrc(CG\_FadcSrc **FadcSrc**)

引数:

**FadcSrc**:以下から、AD コンバーターのクロックソースを選択します。

- **CG\_FADC\_SRC\_FC**: fc
- **CG\_FADC\_SRC\_FPLL**: fplladc

機能:

AD コンバーターのクロックソースを設定します。

戻り値:

なし

## 4.2.3.17 CG\_SetPLL1ForADC

AD コンバーターの PLL1 設定

関数のプロトタイプ宣言:

void

CG\_SetPLL1ForADC(FunctionalState **NewState**)

引数:

**NewState**: 以下から、PLL1 設定を変更します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

AD コンバーター用の PLL1 を設定します。

戻り値:

なし

## 4.2.3.18 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



## 4.2.3.19 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

## 4.2.3.20 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

## 4.2.3.21 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

## 4.2.3.22 CG\_SetFs

低速発振器(fs)の設定

**関数のプロトタイプ宣言:**

Result

CG\_SetFs(FunctionalState **NewState**)

**引数:**

**NewState:** 以下から、低速発振器の有効/無効を設定します。

- **ENABLE:** 有効
- **DISABLE:** 無効

**機能:**

低速発振器(fs)の有効/無効を設定します。

**戻り値:**

**SUCCESS:** 成功

**ERROR:** 失敗

## 4.2.3.23 CG\_GetFsState

低速発振器(fs)の状態取得

**関数のプロトタイプ宣言:**

FunctionalState

CG\_GetFsState(void)

**引数:**

なし

**機能:**

低速発振器(fs)の状態を取得します。

**戻り値:**

fs の状態です。

**ENABLE:** 有効

**DISABLE:** 無効

#### 4.2.3.24 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\_STOP2:** STOP2 モード (一部の機能を保持して内部電源を遮断)
- **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。

#### 4.2.3.25 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\_STOP2: STOP2 モード

CG\_STBY\_MODE\_IDLE: IDLE モード

CG\_STBY\_MODE\_UNKNOWN: 無効なモード

#### 4.2.3.26 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。

#### 4.2.3.27 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

#### 4.2.3.28 CG\_SetPortKeepInStop2Mode

STOP2 モード中の I/O 制御信号保持状態の設定。

**関数のプロトタイプ宣言:**

void

CG\_SetPortKeepInStop2Mode(FunctionalState **NewState**)

**引数:**

**NewState:**

➤ **DISABLE:** ポートによる制御

➤ **ENABLE:** DISABLE->ENABLE 設定時の状態を保持

STOP2 モード中の I/O 制御信号保持の詳細については、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:**

STOP2 モード中の I/O 制御信号保持の有効/無効を切り替えます。

**戻り値:**

なし

#### 4.2.3.29 CG\_GetPortKeepInStop2Mode

STOP2 モード中の I/O 制御信号保持状態の取得

**関数のプロトタイプ宣言:**

FunctionalState

CG\_GetPinStateInStopMode(void)

**引数:**

なし。

**機能:**

STOP2 モード中の I/O 制御信号保持状態を取得します。

戻り値:

STOP2 モード時の端子状態:

**DISABLE:** ポートによる制御

**ENABLE:** DISABLE->ENABLE 設定時の状態を保持

#### 4.2.3.30 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 使用

➤ **CG\_FC\_SRC\_FPLL:** fpll 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

#### 4.2.3.31 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:

**CG\_FC\_SRC\_FOSC:** fosc 選択

**CG\_FC\_SRC\_FPLL:** fpll 選択

## 4.2.3.32 CG\_SetFtmrdSrc

高精度タイマ TMRD のクロックソース設定

関数のプロトタイプ宣言:

```
void  
CG_SetFtmrdSrc(CG_TmrdUnit TmrdUnit,  
               CG_FtmrdSrc FtmrdSrc)
```

引数:

*TmrdUnit*: 以下から、TMRD のユニットを選択します。

**CG\_TMRD\_UNIT\_A**: ユニット A

*FtmrdSrc*: 以下から、クロックソースを選択します。

- **CG\_FTMRD\_SRC\_FPLL**: fpll
- **CG\_FTMRD\_SRC\_HALF\_FPLL**: fpll/2
- **CG\_FTMRD\_SRC\_QUARTER\_FPLL**: fpll/4

機能:

高精度タイマ TMRD のクロックソースを設定します。

戻り値:

なし

## 4.2.3.33 CG\_GetFtmrdSrc

高精度タイマ TMRD のクロックソースの取得

関数のプロトタイプ宣言:

```
CG_FtmrdSrc  
CG_GetFtmrdSrc(CG_TmrdUnit TmrdUnit)
```

引数:

*TmrdUnit*: 以下から、TMRD のユニットを選択します。

**CG\_TMRD\_UNIT\_A**: ユニット A

機能:

高精度タイマ TMRD のクロックソースを取得します。

戻り値:

TMRD のクロックソースです。

**CG\_FTMRD\_SRC\_FPLL**: fpll

**CG\_FTMRD\_SRC\_HALF\_FPLL**: fpll/2

CG\_FTMRD\_SRC\_QUARTER\_FPLL: fppll/4

CG\_FTMRD\_SRC\_UNKNOWN: 無効な値

#### 4.2.3.34 CG\_SetTMRDClk

TMRD クロックの許可/禁止

関数のプロトタイプ宣言:

void

CG\_SetTMRDClk(CG\_TmrUnit *TmrUnit*,  
FunctionalState *NewState*)

引数:

*TmrUnit*: 以下から、TMRD のユニットを選択します。

CG\_TMRD\_UNIT\_A: ユニット A

*NewState*: 以下から、TMRD クロックの許可/禁止を選択します。

DISABLE: 許可

ENABLE: 禁止

機能:

TMRD クロックの許可/禁止を設定します。

戻り値:

なし

#### 4.2.3.35 CG\_GetTMRDClkState

TMRD クロック設定の状態取得

関数のプロトタイプ宣言:

FunctionalState

CG\_GetTMRDClkState(CG\_TmrUnit *TmrUnit*)

引数:

*TmrUnit*: 以下から、TMRD のユニットを選択します。

CG\_TMRD\_UNIT\_A: ユニット A

機能:

TMRD クロック設定の状態を取得します。

戻り値:

TMRD クロックの設定状態:

ENABLE: 許可



DISABLE: 禁止

## 4.2.3.36 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。

## 4.2.3.37 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\_A:** INTA

- **CG\_INT\_SRC\_B**: INTB
- **CG\_INT\_SRC\_C**: INTC
- **CG\_INT\_SRC\_D**: INTD
- **CG\_INT\_SRC\_E**: INTE
- **CG\_INT\_SRC\_F**: INTF
- **CG\_INT\_SRC\_10**: INT10
- **CG\_INT\_SRC\_11**: INT11
- **CG\_INT\_SRC\_12**: INT12
- **CG\_INT\_SRC\_13**: INT13
- **CG\_INT\_SRC\_14**: INT14
- **CG\_INT\_SRC\_15**: INT15
- **CG\_INT\_SRC\_INTKWUP0**: INTKWUP0
- **CG\_INT\_SRC\_INTKWUP1**: INTKWUP1
- **CG\_INT\_SRC\_INTKSCAN**: INTKSCAN
- **CG\_INT\_SRC\_INTRTC**: INTRTC
- **CG\_INT\_SRC\_INTPHC00**: INTPHC00
- **CG\_INT\_SRC\_INTPHC01**: INTPHC01
- **CG\_INT\_SRC\_INTPHC0EVRY**: INTPHC0EVRY
- **CG\_INT\_SRC\_INTPHC10**: INTPHC10
- **CG\_INT\_SRC\_INTPHC11**: INTPHC11
- **CG\_INT\_SRC\_INTPHC1EVRY**: INTPHC1EVRY

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

**INTSource** が **CG\_INT\_SRC\_0** ~ **CG\_INT\_SRC\_A** のいずれの場合、本パラメータは下記いずれかとなります。

- **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。

#### 4.2.3.38 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**: 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\_A**: INTA
- **CG\_INT\_SRC\_B**: INTB
- **CG\_INT\_SRC\_C**: INTC
- **CG\_INT\_SRC\_D**: INTD
- **CG\_INT\_SRC\_E**: INTE
- **CG\_INT\_SRC\_F**: INTF
- **CG\_INT\_SRC\_10**: INT10
- **CG\_INT\_SRC\_11**: INT11
- **CG\_INT\_SRC\_12**: INT12
- **CG\_INT\_SRC\_13**: INT13
- **CG\_INT\_SRC\_14**: INT14
- **CG\_INT\_SRC\_15**: INT15
- **CG\_INT\_SRC\_INTKWUP0**: INTKWUP0
- **CG\_INT\_SRC\_INTKWUP1**: INTKWUP1
- **CG\_INT\_SRC\_INTKSCAN**: INTKSCAN
- **CG\_INT\_SRC\_INTRTC**: INTRTC
- **CG\_INT\_SRC\_INTPHC00**: INTPHC00
- **CG\_INT\_SRC\_INTPHC01**: INTPHC01
- **CG\_INT\_SRC\_INTPHC0EVRY**: INTPHC0EVRY
- **CG\_INT\_SRC\_INTPHC10**: INTPHC10
- **CG\_INT\_SRC\_INTPHC11**: INTPHC11
- **CG\_INT\_SRC\_INTPHC1EVRY**: INTPHC1EVRY

## 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

#### 4.2.3.39 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:** 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\_A:** INTA
- **CG\_INT\_SRC\_B:** INTB
- **CG\_INT\_SRC\_C:** INTC
- **CG\_INT\_SRC\_D:** INTD
- **CG\_INT\_SRC\_E:** INTE
- **CG\_INT\_SRC\_F:** INTF
- **CG\_INT\_SRC\_10:** INT10
- **CG\_INT\_SRC\_11:** INT11
- **CG\_INT\_SRC\_12:** INT12
- **CG\_INT\_SRC\_13:** INT13
- **CG\_INT\_SRC\_14:** INT14
- **CG\_INT\_SRC\_15:** INT15
- **CG\_INT\_SRC\_INTKWUP0:** INTKWUP0
- **CG\_INT\_SRC\_INTKWUP1:** INTKWUP1
- **CG\_INT\_SRC\_INTKSCAN:** INTKSCAN

- **CG\_INT\_SRC\_INTRTC:** INTRTC
- **CG\_INT\_SRC\_INTPHC00:** INTPHC00
- **CG\_INT\_SRC\_INTPHC01:** INTPHC01
- **CG\_INT\_SRC\_INTPHC0EVRY:** INTPHC0EVRY
- **CG\_INT\_SRC\_INTPHC10:** INTPHC10
- **CG\_INT\_SRC\_INTPHC11:** INTPHC11
- **CG\_INT\_SRC\_INTPHC1EVRY:** INTPHC1EVRY

**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。

#### 4.2.3.40 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:

**PowerOnReset1** (Bit0) パワーオンリセット

**PinReset** (Bit1) リセット端子によるリセット

**WDTReset** (Bit 2) WDT リセット

**STOP2Reset**(Bit3) STOP2 モード解除

**SYSReset**(Bit4) <SYSResetREQ>によるリセット

**PowerOnReset2** (Bit6) パワーオンリセット

#### 4.2.3.41 CG\_SetPeriphClkSupply

PSC および AD コンバーター(ユニット A, B, C)用 fc クロックソースの設定

**関数のプロトタイプ宣言:**

void

CG\_SetPeriphClkSupply (uint32\_t **Periph**, FunctionalState **NewState**)

引数:

**Periph:** 以下から、CG のfcクロック供給先となる周辺回路を選択します。

- **CG\_PSC\_CLK\_SUPPLY:** PSC
- **CG\_ADC\_C\_CLK\_SUPPLY:** ADC ユニット C
- **CG\_ADC\_B\_CLK\_SUPPLY:** ADC ユニット B
- **CG\_ADC\_A\_CLK\_SUPPLY:** ADC ユニット A

**NewState:** クロック fc 供給設定変更

**DISABLE:** 許可

**ENABLE:** 禁止

機能:

PSC および AD コンバーター(ユニット A, B, C)用クロック fc の供給先を設定します。

戻り値:

なし

#### 4.2.3.42 CG\_SetFclkPeriphA

グループ A モジュールへのクロック fclk 供給の設定

関数のプロトタイプ宣言:

void

CG\_SetFclkPeriphA(uint32\_t **Periph**, FunctionalState **NewState**)

引数:

**Periph:** 以下から、CG の fclk クロック供給先となる周辺回路を設定します。

- **FCLK\_A\_TMRB10:** TMRB10
- **FCLK\_A\_TMRB11:** TMRB11
- **FCLK\_A\_TMRB12:** TMRB12
- **FCLK\_A\_TMRB13:** TMRB13
- **FCLK\_A\_TMRB14:** TMRB14
- **FCLK\_A\_TMRB15:** TMRB15
- **FCLK\_A\_TMRB16:** TMRB16
- **FCLK\_A\_TMRB17:** TMRB17
- **FCLK\_A\_TMRB18:** TMRB18
- **FCLK\_A\_TMRB19:** TMRB19
- **FCLK\_A\_DAC0** : DAC チャンネル 0
- **FCLK\_A\_DAC1** : DAC チャンネル 1
- **FCLK\_A\_EBIF** : EBIF
- **FCLK\_A\_UART0** : UART チャンネル 0
- **FCLK\_A\_UART1** : UART チャンネル 1
- **FCLK\_A\_DMACA** : DMAC ユニット A

- **FCLK\_A\_DMACB** : DMAC ユニット B
- **FCLK\_A\_DMACC** : DMAC ユニット C
- **FCLK\_A\_PORTA** : port A
- **FCLK\_A\_PORTB** : port B
- **FCLK\_A\_PORTC** : port C
- **FCLK\_A\_PORTD** : port D
- **FCLK\_A\_PORTE** : port E
- **FCLK\_A\_PORTF** : port F
- **FCLK\_A\_PORTG** : port G
- **FCLK\_A\_PORTH** : port H
- **FCLK\_A\_PORTJ** : port J
- **FCLK\_A\_PORTK** : port K
- **FCLK\_A\_PORTL** : port L
- **FCLK\_A\_PORTM** : port M
- **FCLK\_A\_PORTN** : port N
- **FCLK\_A\_ALL** : すべて

**NewState**: 以下から、クロック供給の許可/禁止を選択します。

**DISABLE**: 許可

**ENABLE**: 禁止

**機能**:

グループ A モジュールへのクロック fclk 供給を設定します。

**戻り値**:

なし

#### 4.2.3.43 CG\_SetFclkPeriphB

グループ B モジュールへのクロック fclk 供給の設定

**関数のプロトタイプ宣言**:

void

CG\_SetFclkPeriphB(uint32\_t **Periph**, FunctionalState **NewState**)

**引数**:

**Periph**: 以下から、CG の fclk クロック供給先となる周辺回路を設定します。

- **FCLK\_B\_PORTP** : port P
- **FCLK\_B\_PORTR** : port R
- **FCLK\_B\_PORTT** : port T
- **FCLK\_B\_PORTU** : port U
- **FCLK\_B\_PORTV** : port V
- **FCLK\_B\_PORTW** : port W
- **FCLK\_B\_PORTY** : port Y

- **FCLK\_B\_PORTAA:** port AA
- **FCLK\_B\_PORTAB:** port AB
- **FCLK\_B\_PORTAC:** port AC
- **FCLK\_B\_PORTAD:** port AD
- **FCLK\_B\_PORTAE:** port AE
- **FCLK\_B\_PORTAF:** port AF
- **FCLK\_B\_PORTAG:** port AG
- **FCLK\_B\_PORTAH:** port AH
- **FCLK\_B\_PORTAJ:** port AJ
- **FCLK\_B\_ADCA** : ADC ユニット A
- **FCLK\_B\_ADCB** : ADC ユニット B
- **FCLK\_B\_ADCC** : ADC ユニット C
- **FCLK\_B\_EPHC** : EPHC
- **FCLK\_B\_SBI** : SBI
- **FCLK\_B\_WDT** : WDT
- **FCLK\_B\_ALL** : すべて

**NewState:** 以下から、クロック供給の許可/禁止を選択します。

**DISABLE:** 許可

**ENABLE:** 禁止

**機能:**

グループ B モジュールへのクロック fclk 供給を設定します。

**戻り値:**

なし

#### 4.2.3.44 CG\_SetFcPeriphA

グループ A モジュールへのクロック fc 供給の設定

**関数のプロトタイプ宣言:**

void

CG\_SetFcPeriphA(uint32\_t **Periph**, FunctionalState **NewState**)

**引数:**

**Periph:** 以下から、CG の fc クロック供給先となる周辺回路を選択します。

- **FC\_A\_ESIO0** : ESIO チャンネル 0
- **FC\_A\_ESIO1** : ESIO チャンネル 1
- **FC\_A\_ESIO2** : ESIO チャンネル 2
- **FC\_A\_TMRD** : TMRD
- **FC\_A\_SIO\_UART0** : SIO/UART チャンネル 0
- **FC\_A\_SIO\_UART1** : SIO/UART チャンネル 1
- **FC\_A\_SIO\_UART2** : SIO/UART チャンネル 2



- **FC\_A\_SIO\_UART3** : SIO/UART チャンネル 3
- **FC\_A\_SIO\_UART4** : SIO/UART チャンネル 4
- **FC\_A\_SIO\_UART5** : SIO/UART チャンネル 5
- **FC\_A\_TMRB00** : TMRB00
- **FC\_A\_TMRB01** : TMRB01
- **FC\_A\_TMRB02** : TMRB02
- **FC\_A\_TMRB03** : TMRB03
- **FC\_A\_TMRB04** : TMRB04
- **FC\_A\_TMRB05** : TMRB05
- **FC\_A\_TMRB06** : TMRB06
- **FC\_A\_TMRB07** : TMRB07
- **FC\_A\_TMRB08** : TMRB08
- **FC\_A\_TMRB09** : TMRB09
- **FC\_A\_TMRCCAP0** : TMRC キャプチャチャンネル 0
- **FC\_A\_TMRCCAP1** : TMRC キャプチャチャンネル 1
- **FC\_A\_TMRCCAP2** : TMRC キャプチャチャンネル 2
- **FC\_A\_TMRCCAP3** : TMRC キャプチャチャンネル 3
- **FC\_A\_TMRCCMP0** : TMRC コンペア回路のチャンネル 0
- **FC\_A\_TMRCCMP1** : TMRC コンペア回路のチャンネル 1
- **FC\_A\_TMRCCMP2** : TMRC コンペア回路のチャンネル 2
- **FC\_A\_TMRCCMP3** : TMRC コンペア回路のチャンネル 3
- **FC\_A\_TMRCCMP4** : TMRC コンペア回路のチャンネル 4
- **FC\_A\_TMRCCMP5** : TMRC コンペア回路のチャンネル 5
- **FC\_A\_TMRCCMP6** : TMRC コンペア回路のチャンネル 6
- **FC\_A\_TMRCCMP7** : TMRC コンペア回路のチャンネル 7
- **FC\_A\_ALL** : すべて

**NewState:** 以下から、クロック供給の許可/禁止を選択します。

**DISABLE:** 許可

**ENABLE:** 禁止

**機能:**

グループ A モジュールへのクロック fc 供給を設定します。

**戻り値:**

なし

#### 4.2.3.45 CG\_SetFcPeriphB

グループ B モジュールへのクロック fc 供給の設定

**関数のプロトタイプ宣言:**

void

CG\_SetFcPeriphB (uint32\_t **Periph**, FunctionalState **NewState**)

引数:

**Periph:** 以下から、CG の fc クロック供給先となる周辺回路を選択します。

➤ **FCLK\_B\_TMRCTBT:** TMRC の TBT 回路

**NewState:** 以下から、クロック供給の許可/禁止を選択します。

**DISABLE:** 許可

**ENABLE:** 禁止

機能:

グループ B モジュールへのクロック fc 供給を設定します。

戻り値:

なし

## 4.2.4 データ構造

### 4.2.4.1 CG\_ResetFlag

メンバ:

uint32\_t

**All** CG リセット要因を指定します。

ビットフィールド:

uint32\_t

**PowerOnReset1**(Bit0)    パワーオンリセット

uint32\_t

**PinReset** (Bit1)    RESET 端子によるリセット

uint32\_t

**WDTReset**(Bit2)    WDT によるリセット

uint32\_t

**STOP2Reset**(Bit3)    STOP2 モード解除

uint32\_t

**SYSReset** (Bit4)    <SYSResetREQ>によるリセット

uint32\_t

**Reserved1** (Bit5)    Reserved

uint32\_t

**PowerOnReset2**(Bit6)    パワーオンリセット

uint32\_t

**Reserved2** (Bit7~bit31)    Reserved

## 5. DAC

### 5.1 概要

本デジタルアナログコンバータは、下記の機能を持っています。

- 分解能 10 ビット
- バッファアンプ内蔵
- 低消費電力モード

本ドライバは、以下のファイルで構成されています。

/Libraries/TX04\_Periph\_Driver/src/tmpm440\_dac.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_dac.h

### 5.2 API 関数

#### 5.2.1 関数一覧

- ◆ void DAC\_SetOutputCode(TSB\_DA\_TypeDef \* **DACx**,uint16\_t **OutputCode**);
- ◆ void DAC\_Start(TSB\_DA\_TypeDef \* **DACx**);
- ◆ void DAC\_Stop(TSB\_DA\_TypeDef \* **DACx**);
- ◆ void DAC\_SetVOutHoldTime(TSB\_DA\_TypeDef \* **DACx**);

#### 5.2.2 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) DAC の設定と出力値の設定:  
DAC\_SetOutputCode(), DAC\_SetVOutHoldTime()
- 2) 開始と停止制御:  
DAC\_Start(), DAC\_Stop()

#### 5.2.3 関数仕様

補足: 下記の全 API において、パラメータ“TSB\_DA\_TypeDef \* **DACx**” は TSB\_DAA または TSB\_DAB となります。

##### 5.2.3.1 DAC\_SetOutputCode

出力電圧設定

関数のプロトタイプ宣言:

void

```
DAC_SetOutputCode(TSB_DA_TypeDef * DACx,  
uint16_t OutputCode)
```

引数:

**DACx**: DAC チャンネルを選択してください。

**OutputCode**: 出力するアナログ電圧値を設定します。ビット幅が10ビットなので、最大設定値は 0x3ff となります。

機能:

出力するアナログ電圧値を設定します。

戻り値:

なし

## 5.2.3.2 DAC\_Start

DAC 動作の開始

関数のプロトタイプ宣言:

```
void  
DAC_Start(TSB_DA_TypeDef * DACx);
```

引数:

**DACx**: DAC チャンネルを選択してください。

機能:

デジタルアナログコンバータの動作を開始します。

戻り値:

なし

## 5.2.3.3 DAC\_Stop

DAC 動作の停止

関数のプロトタイプ宣言:

```
void  
DAC_Stop(TSB_DA_TypeDef * DACx);
```

引数:

**DACx**: DAC チャンネルを選択してください。

**機能:**

デジタルアナログ動作を停止します。

**戻り値:**

なし

## 5.2.3.4 DAC\_SetVOutHoldTime

VOut 保持時間の調整

**関数のプロトタイプ宣言:**

void

DAC\_SetVOutHoldTime(TSB\_DA\_TypeDef \* **DACx**);

**引数:**

**DACx**: DAC チャンネルを選択してください。

**機能:**

VOut 保持時間を調整します。

**戻り値:**

なし

## 5.2.4 データ構造

なし

## 6. DMAC

### 6.1 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 3 ユニットの DMA コントローラ (UNITA, UNITB, UNITC) を内蔵しています。各ユニットは、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。各ユニットは 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\\TX04\_Periph\_Driver\\src\\tmpm440\_dmac.c

\\Libraries\\TX04\_Periph\_Driver\\inc\\tmpm440\_dmac.h

### 6.2 API 関数

#### 6.2.1 関数一覧

- ◆ void DMAC\_Enable(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ void DMAC\_Disable(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ DMAL\_INTReq DMAL\_GetINTReq(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ DMAL\_TxINTReq DMAL\_GetTxINTReq(TSB\_DMAL\_TypeDef \* **DMACx**,  
DMAL\_Channel **Chx**);
- ◆ void DMAL\_ClearTxINTReq(TSB\_DMAL\_TypeDef \* **DMACx**, DMAL\_Channel **Chx**,  
DMAL\_INTSrc **INTSource**);
- ◆ DMAL\_TxINTReq DMAL\_GetRawTxINTReq(TSB\_DMAL\_TypeDef \* **DMACx**,  
DMAL\_Channel **Chx**);
- ◆ WorkState DMAL\_GetChannelTxState(TSB\_DMAL\_TypeDef \* **DMACx**,  
DMAL\_Channel **Chx**);
- ◆ void DMAL\_SetSWBurstReq(DMAL\_ReqNum **BurstReq**);
- ◆ void DMALB\_SetSWBurstReq(DMALB\_ReqNum **BurstReq**);
- ◆ void DMALCC\_SetSWBurstReq(DMALCC\_ReqNum **BurstReq**);
- ◆ DMAL\_BurstReqState DMAL\_GetSWBurstReqState(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ void DMALB\_SetSWSingleReq(DMALB\_ReqNum **SingleReq**);
- ◆ void DMALCC\_SetSWSingleReq(DMALCC\_ReqNum **SingleReq**);

- ◆ DMAC\_SingleReqState DMAC\_GetSWSingleReqState(TSB\_DMAL\_TypeDef \* **DMACx**);
- ◆ void DMAC\_SetLinkedList(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**, uint32\_t **LinkedAddr**);
- ◆ WorkState DMAC\_GetFIFOState(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**);
- ◆ void DMAC\_SetDMAHalt(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**, FunctionalState **NewState**);
- ◆ void DMAC\_SetLockedTx(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**, FunctionalState **NewState**);
- ◆ void DMAC\_SetTxINTConfig(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**, DMAL\_INTSrc **INTSource**, FunctionalState **NewState**);
- ◆ void DMAC\_SetDMAChannel(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**, FunctionalState **NewState**);
- ◆ void DMAC\_Init(TSB\_DMAL\_TypeDef \* **DMACx**, DMAC\_Channel **Chx**, DMAL\_InitTypeDef \* **InitStruct**);

## 6.2.2 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) DMAL 基本設定:  
DMAL\_Enable(),DMAL\_Disable(),DMAL\_SetDMAChannel(), DMAL\_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:  
DMAL\_GetINTReq(),DMAL\_GetTxINTReq(),DMAL\_GetRawTxINTReq(),  
DMAL\_GetChannelTxState(), DMAL\_GetFIFOState()
- 3) DMA 割り込み設定、DMA 割り込み要求のクリア:  
DMAL\_ClearTxINTReq(), DMAL\_SetTxINTConfig()
- 4) DMA ソフトウェア要求の設定、および取得:  
DMACA\_SetSWBurstReq(),DMACB\_SetSWBurstReq(),DMACC\_SetSWBurstReq(),DMAL\_GetSWBurstReqState(),DMACB\_SetSWSingleReq(),DMACC\_SetSWSingleReq(),DMAL\_SetLinkedList, DMAL\_GetSWSingleReqState()
- 5) その他の設定:  
DMAL\_SetDMAHalt (), DMAL\_SetLockedTx()

## 6.2.3 関数仕様

### 6.2.3.1 DMAL\_Enable

DMA 回路動作の許可

関数のプロトタイプ宣言:

```
void  
DMAL_Enable(TSB_DMAL_TypeDef * DMACx);
```

**引数:**

**DMACx:** 以下からユニットを選択します。

- **DMAC\_UNIT\_A:** ユニット A
- **DMAC\_UNIT\_B:** ユニット B
- **DMAC\_UNIT\_C:** ユニット C

**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

## 6.2.3.2 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\_DMCA\_TypeDef \* **DMACx**);

**引数:**

**DMACx:** 以下からユニットを選択します。

- **DMAC\_UNIT\_A:** ユニット A
- **DMAC\_UNIT\_B:** ユニット B
- **DMAC\_UNIT\_C:** ユニット C

**機能:**

DMA 回路動作を禁止します。

**戻り値:**

なし

## 6.2.3.3 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\_GetlINTReq(TSB\_DMAC\_TypeDef \* **DMACx**);

**引数:**

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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。

## 6.2.3.4 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\_DMAC\_TypeDef \* **DMACx**,  
DMAC\_Channel **Chx**);

**引数:**

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.5 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_DMAL_TypeDef * DMACx,  
                    DMAC_Channel Chx,  
                    DMAC_INTSrc INTSource);
```

引数:

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.6 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_DMAL_TypeDef * DMACx,  
                     DMAC_Channel Chx);
```

引数:

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B

- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.7 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_DMACH_TypeDef * DMACx,  
                        DMACH_Channel Chx);
```

**引数**:

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.8 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\_ESIO0\_RX**: ESIO0 受信
- **DMACA\_ESIO0\_TX**: ESIO0 送信
- **DMACA\_NORMAL\_UNITA\_ADC**: ユニット A の通常 AD 変換終了
- **DMACA\_SIO3\_UART3\_RX**: SIO3/UART3 受信
- **DMACA\_SIO3\_UART3\_TX**: SIO3/UART3 送信
- **DMACA\_SIO0\_UART0\_RX**: SIO0/UART0 受信
- **DMACA\_SIO0\_UART0\_TX**: SIO0/UART0 送信
- **DMACA\_TMRB00\_CMP\_MATCH**: TMRB00 コンペア一致
- **DMACA\_TMRB04\_CMP\_MATCH**: TMRB04 コンペア一致
- **DMACA\_TMRB10\_CMP\_MATCH**: TMRB10 コンペア一致
- **DMACA\_TMRB14\_CMP\_MATCH**: TMRB14 コンペア一致
- **DMACA\_TMRC\_CMP0\_MATCH**: TMRC コンペア 0 一致
- **DMACA\_TMRC\_CMP1\_MATCH**: TMRC コンペア 1 一致
- **DMACA\_HIGHEST\_UNITA\_ADC**: ユニット A 最優先 AD 変換終了
- **DMACA\_PHCNT0\_CMP0\_MATCH**: PHCNT0 コンペア 0 一致
- **DMACA\_PIN**: DREQA 端子

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

## 6.2.3.9 DMACB\_SetSWBurstReq

ソフトウェアによるユニット B 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

DMACB\_SetSWBurstReq(DMACB\_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。

- **DMACB\_ESIO1\_RX**: ESIO1 受信
- **DMACB\_ESIO1\_TX**: ESIO1 送信
- **DMACB\_NORMAL\_UNITB\_ADC**: ユニット B の通常 AD 変換終了
- **DMACB\_SIO4\_UART4\_RX**: SIO4/UART4 受信
- **DMACB\_SIO4\_UART4\_TX**: SIO4/UART4 送信
- **DMACB\_SIO1\_UART1\_RX**: SIO1/UART1 受信
- **DMACB\_SIO1\_UART1\_TX**: SIO1/UART1 送信
- **DMACB\_UART0\_RX**: UART0 受信
- **DMACB\_UART0\_TX**: UART0 送信
- **DMACB\_TMRB08\_CAPTURE0**: TMRB08 キャプチャ 0 割り込み
- **DMACB\_TMRC0\_CAPTURE0**: TMRC0 キャプチャ 0 割り込み
- **DMACB\_TMRC0\_CAPTURE1**: TMRC0 キャプチャ 1 割り込み
- **DMACB\_HIGHEST\_UNITB\_ADC**: ユニット B の最優先 AD 変換終了
- **DMACB\_PCNT1\_CMP0\_MATCH**: PHCNT1 コンペア 0 一致
- **DMACB\_TMRD0\_CMP\_MATCH**: TMRD0 コンペア一致
- **DMACB\_PIN**: DREQB 端子

機能:

ソフトウェアによるユニット B の DMA 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

## 6.2.3.10 DMACC\_SetSWBurstReq

ソフトウェアによるユニット C 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

DMACC\_SetSWBurstReq(DMACC\_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。

- **DMACC\_ESIO2\_RX**: ESIO2 受信
- **DMACC\_ESIO2\_TX**: ESIO2 送信
- **DMACC\_NORMAL\_UNITC\_ADC**: ユニット C の通常 AD 変換終了
- **DMACC\_SIO5\_UART5\_RX**: SIO5/UART5 受信
- **DMACC\_SIO5\_UART5\_TX**: SIO5/UART5 送信

- **DMACC\_SIO2\_UART2\_RX** : SIO2/UART2 受信
- **DMACC\_SIO2\_UART2\_TX** : SIO2/UART2 送信
- **DMACC\_UART1\_RX** : UART1 受信
- **DMACC\_UART1\_TX** : UART1 送信
- **DMACC\_TMRB19\_CAPTURE0** : TMRC19 キャプチャ 0 割り込み
- **DMACC\_TMRC0\_CAPTURE2** : TMRC0 キャプチャ 2 割り込み
- **DMACC\_TMRC0\_CAPTURE3** : TMRC0 キャプチャ 3 割り込み
- **DMACC\_HIGHEST\_UNITC\_ADC**: ユニット C の最優先 AD 変換終了
- **DMACC\_PHCP\_CYCLE0** : PHCP サイクル 0 割り込み
- **DMACC\_TMRD10\_CMP\_MATCH** : TMRD10 キャプチャー致
- **DMACC\_PIN**: DREQC 端子

**機能:**

ソフトウェアによるユニット C の DMA 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

## 6.2.3.11 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\_DMAL\_TypeDef \* **DMACx**);

**引数:**

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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。

## 6.2.3.12 DMACB\_SetSWSingleReq

ソフトウェアによるユニット B の DMA シングル転送要求の設定

関数のプロトタイプ宣言:

void

DMACB\_SetSWSingleReq(DMACB\_ReqNum *SingleReq*);

引数:

**SingleReq:** 以下から、シングル要求番号を選択します。

- **DMACB\_UART0\_RX:** UART0 受信
- **DMACB\_UART0\_TX:** UART0 送信

機能:

ソフトウェアによる DMA シングル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

## 6.2.3.13 DMACC\_SetSWSingleReq

ソフトウェアによるユニット C の DMA シングル転送要求の設定

関数のプロトタイプ宣言:

void

DMACC\_SetSWSingleReq(DMACC\_ReqNum *SingleReq*);

引数:

**SingleReq:** 以下から、バースト要求番号を選択します。

- **DMACB\_UART1\_RX:** UART1 受信
- **DMACB\_UART1\_TX:** UART1 送信

機能:

ソフトウェアによる DMA シングル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

## 6.2.3.14 DMAC\_GetSWSingleReqState

ソフトウェアによる DMA シングル要求状態の取得

**関数のプロトタイプ宣言:**

DMAC\_SingleReqState

DMAC\_GetSWSingleReqState(TSB\_DMAC\_TypeDef \* **DMACx**);

**引数:**

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**機能:**

ソフトウェアによる DMA シングル要求状態を取得します。

**戻り値:**

DMA シングル要求状態です。構造体 "DMAC\_SingleReqState"の詳細は"データ構造"を参照してください。

## 6.2.3.15 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\_DMAC\_TypeDef \* **DMACx**,  
DMAC\_Channel **Chx**,  
uint32\_t **LinkedAddr**);

**引数:**

**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

## 6.2.3.16 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。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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**

## 6.2.3.17 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。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.18 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_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。

- **DMAC\_UNIT\_A**: ユニット A

- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.19 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_DMACH_TypeDef * DMACx,  
                    DMACH_Channel Chx,  
                    DMACH_INTSrc INTSource,  
                    FunctionalState NewState);
```

**引数**:

**DMACx**: 以下からユニットを選択します。

- **DMAC\_UNIT\_A**: ユニット A
- **DMAC\_UNIT\_B**: ユニット B
- **DMAC\_UNIT\_C**: ユニット C

**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

## 6.2.3.20 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_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。

- **DMACH\_UNIT\_A:** ユニット A
- **DMACH\_UNIT\_B:** ユニット B
- **DMACH\_UNIT\_C:** ユニット C

**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

**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、**DMACH\_SetDMAHalt()** をコールし、DMA 要求を無視した後、**DMACH\_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

## 6.2.3.21 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。

- **DMACH\_UNIT\_A:** ユニット A
- **DMACH\_UNIT\_B:** ユニット B
- **DMACH\_UNIT\_C:** ユニット C

**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:**

**DMACH\_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

## 6.2.4 データ構造

### 6.2.4.1 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

- **DMAC\_MEMORY\_TO\_PERIPH:** メモリ->周辺回路
- **DMAC\_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**: 以下から、転送先の周辺回路を設定します。

- **DMACA\_ESIO0\_RX** : ESIO0 受信
- **DMACA\_ESIO0\_TX** : ESIO0 送信
- **DMACA\_NORMAL\_UNITA\_ADC** : ユニット A の通常 AD 変換終了
- **DMACA\_SIO3\_UART3\_RX** : SIO3/UART3 受信
- **DMACA\_SIO3\_UART3\_TX** : SIO3/UART3 送信
- **DMACA\_SIO0\_UART0\_RX** : SIO0/UART0 受信
- **DMACA\_SIO0\_UART0\_TX** : SIO0/UART0 送信
- **DMACA\_TMRB00\_CMP\_MATCH** : TMRB00 コンペア一致
- **DMACA\_TMRB04\_CMP\_MATCH** : TMRB04 コンペア一致
- **DMACA\_TMRB10\_CMP\_MATCH** : TMRB10 コンペア一致
- **DMACA\_TMRB14\_CMP\_MATCH** : TMRB14 コンペア一致
- **DMACA\_TMRC\_CMP0\_MATCH** : TMRC コンペア 0 一致
- **DMACA\_TMRC\_CMP1\_MATCH** : TMRC コンペア 1 一致
- **DMACA\_HIGHEST\_UNITA\_ADC** : ユニット A の AD 変換終了
- **DMACA\_PHCNT0\_CMP0\_MATCH** : PHCNT0 コンペア 0 一致
- **DMACA\_PIN** : DREQA 端子

DMACA\_ReqNum

**A\_TxSrcPeriph**: 以下から、転送元の周辺回路を設定します。

- **DMACA\_ESIO0\_RX** : ESIO0 受信
- **DMACA\_ESIO0\_TX** : ESIO0 送信
- **DMACA\_NORMAL\_UNITA\_ADC** : ユニット A の通常 AD 変換終了
- **DMACA\_SIO3\_UART3\_RX** : SIO3/UART3 受信
- **DMACA\_SIO3\_UART3\_TX** : SIO3/UART3 送信
- **DMACA\_SIO0\_UART0\_RX** : SIO0/UART0 受信
- **DMACA\_SIO0\_UART0\_TX** : SIO0/UART0 送信
- **DMACA\_TMRB00\_CMP\_MATCH** : TMRB00 コンペア一致
- **DMACA\_TMRB04\_CMP\_MATCH** : TMRB04 コンペア一致
- **DMACA\_TMRB10\_CMP\_MATCH** : TMRB10 コンペア一致
- **DMACA\_TMRB14\_CMP\_MATCH** : TMRB14 コンペア一致
- **DMACA\_TMRC\_CMP0\_MATCH** : TMRC コンペア 0 一致
- **DMACA\_TMRC\_CMP1\_MATCH** : TMRC コンペア 1 一致
- **DMACA\_HIGHEST\_UNITA\_ADC** : ユニット A の最優先 AD 変換終了
- **DMACA\_PHCNT0\_CMP0\_MATCH** : PHCNT0 コンペア 0 一致
- **DMACA\_PIN** : DREQA 端子

DMACB\_ReqNum

**B\_TxDstPeriph:**以下から、転送先の周辺回路を設定します。

- DMACB\_ESIO1\_RX : ESIO1 受信
- DMACB\_ESIO1\_TX : ESIO0 送信
- DMACB\_NORMAL\_UNITB\_ADC: ユニット B の通常 AD 変換終了
- DMACB\_SIO4\_UART4\_RX : SIO4/UART4 受信
- DMACB\_SIO4\_UART4\_TX : SIO4/UART4 送信
- DMACB\_SIO1\_UART1\_RX : SIO1/UART1 受信
- DMACB\_SIO1\_UART1\_TX : SIO1/UART1 送信
- DMACB\_UART0\_RX : UART0 受信
- DMACB\_UART0\_TX : UART0 送信
- DMACB\_TMRB08\_CAPTURE0 : TMRB08 キャプチャ割り込み
- DMACB\_TMRC0\_CAPTURE0 : TMRC0 キャプチャ 0 割り込み
- DMACB\_TMRC0\_CAPTURE1 : TMRC0 キャプチャ 1 割り込み
- DMACB\_HIGHEST\_UNITB\_ADC: ユニット B の最優先 AD 変換終了
- DMACB\_PCNT1\_CMP0\_MATCH : PHCNT1 キャプチャ 0 一致
- DMACB\_TMRD0\_CMP\_MATCH : TMRD0 キャプチャ一致
- DMACB\_PIN : DREQB 端子

DMACB\_ReqNum

**B\_TxSrcPeriph:**以下から、転送元の周辺回路を設定します。

- DMACB\_ESIO1\_RX : ESIO1 受信
- DMACB\_ESIO1\_TX : ESIO0 送信
- DMACB\_NORMAL\_UNITB\_ADC: ユニット B の通常 AD 変換終了
- DMACB\_SIO4\_UART4\_RX : SIO4/UART4 受信
- DMACB\_SIO4\_UART4\_TX : SIO4/UART4 送信
- DMACB\_SIO1\_UART1\_RX : SIO1/UART1 受信
- DMACB\_SIO1\_UART1\_TX : SIO1/UART1 送信
- DMACB\_UART0\_RX : UART0 受信
- DMACB\_UART0\_TX : UART0 送信
- DMACB\_TMRB08\_CAPTURE0 : TMRB08 キャプチャ割り込み
- DMACB\_TMRC0\_CAPTURE0 : TMRC0 キャプチャ 0 割り込み
- DMACB\_TMRC0\_CAPTURE1 : TMRC0 キャプチャ 1 割り込み
- DMACB\_HIGHEST\_UNITB\_ADC: ユニット B の最優先 AD 変換終了
- DMACB\_PCNT1\_CMP0\_MATCH : PHCNT1 キャプチャ 0 一致
- DMACB\_TMRD0\_CMP\_MATCH : TMRD0 キャプチャ一致
- DMACB\_PIN : DREQB 端子

DMACC\_ReqNum

**C\_TxDstPeriph:**以下から、転送先の周辺回路を設定します。

- DMACC\_ESIO2\_RX : ESIO2 受信
- DMACC\_ESIO2\_TX : ESIO2 送信



- **DMACC\_NORMAL\_UNITC\_ADC**: ユニット C の通常 AD 変換終了
- **DMACC\_SIO5\_UART5\_RX**: SIO5/UART5 受信
- **DMACC\_SIO5\_UART5\_TX**: SIO5/UART5 送信
- **DMACC\_SIO2\_UART2\_RX**: SIO2/UART2 受信
- **DMACC\_SIO2\_UART2\_TX**: SIO2/UART2 送信
- **DMACC\_UART1\_RX**: UART1 受信
- **DMACC\_UART1\_TX**: UART1 送信
- **DMACC\_TMRB19\_CAPTURE0**: TMRC19 キャプチャ割り込み
- **DMACC\_TMRC0\_CAPTURE2**: TMRC0 キャプチャ 2 割り込み
- **DMACC\_TMRC0\_CAPTURE3**: TMRC0 キャプチャ 3 割り込み
- **DMACC\_HIGHEST\_UNITC\_ADC**: ユニット C の最優先 AD 変換終了
- **DMACC\_PHCP\_CYCLE0**: PHCP サイクル 0 割り込み
- **DMACC\_TMRD10\_CMP\_MATCH**: TMRD10 キャプチャー致
- **DMACC\_PIN**: DREQC 端子

DMACC\_ReqNum

**C\_TxSrcPeriph**: 以下から、転送元の周辺回路を設定します

- **DMACC\_ESIO2\_RX**: ESIO2 受信
- **DMACC\_ESIO2\_TX**: ESIO2 送信
- **DMACC\_NORMAL\_UNITC\_ADC**: ユニット C の通常 AD 変換終了
- **DMACC\_SIO5\_UART5\_RX**: SIO5/UART5 受信.
- **DMACC\_SIO5\_UART5\_TX**: SIO5/UART5 送信
- **DMACC\_SIO2\_UART2\_RX**: SIO2/UART2 受信
- **DMACC\_SIO2\_UART2\_TX**: SIO2/UART2 送信
- **DMACC\_UART1\_RX**: UART1 受信
- **DMACC\_UART1\_TX**: UART1 送信
- **DMACC\_TMRB19\_CAPTURE0**: TMRC19 キャプチャ割り込み
- **DMACC\_TMRC0\_CAPTURE2**: TMRC0 キャプチャ 2 割り込み
- **DMACC\_TMRC0\_CAPTURE3**: TMRC0 キャプチャ 3 割り込み
- **DMACC\_HIGHEST\_UNITC\_ADC**: ユニット C の最優先 AD 変換終了
- **DMACC\_PHCP\_CYCLE0**: PHCP サイクル 0 割り込み
- **DMACC\_TMRD10\_CMP\_MATCH**: TMRD10 キャプチャー致
- **DMACC\_PIN**: DREQC 端子

FunctionalState

**TxINT**: 以下から、転送割り込みステートを選択します。

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

## 6.2.4.2 DMAC\_INTRReq

**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 チャンネル 0 の割り込み発生状態です。

uint32\_t

**CH1\_INTReq**

: 1 DMAC チャンネル 1 の割り込み発生状態です。

## 7. EPHC

### 7.1 概要

本デバイスは、1 チャンネルの高機能 2 相パルス入力カウンタを内蔵しています。

2 相パルス入力カウンタは、2 相の入力パルスの組み合わせの変化、または、1 相の入力パルスの状態の変化により、アップダウンカウンタをインクリメントまたはデクリメントする機能です。

周期位相測定は 2 相の入力パルスの周期やエッジ間の位相差を測定する機能です。

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/TX04\_Periph\_Driver/src/tmpm440\_ephc.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_ephc.h

### 7.2 API 関数

#### 7.2.1 関数一覧

- ◆ void EPHC\_Enable(TSB\_EPHC\_TypeDef \* **EPHCx**);
- ◆ void EPHC\_Disable(TSB\_EPHC\_TypeDef \* **EPHCx**);
- ◆ void EPHC\_Init(TSB\_EPHC\_TypeDef \* **EPHCx**, EPHC\_InitTypeDef \* **InitStruct**);
- ◆ void EPHC\_EnableInterrupt(TSB\_EPHC\_TypeDef \* **EPHCx**, uint32\_t **EnableINT**);
- ◆ void EPHC\_DisableInterrupt(TSB\_EPHC\_TypeDef \* **EPHCx**, uint32\_t **DisableINT**);
- ◆ EPHC\_INTFactor EPHC\_GetINTFactor(TSB\_EPHC\_TypeDef \* **EPHCx**);
- ◆ void EPHC\_ClearINTFactor(TSB\_EPHC\_TypeDef \* **EPHCx**, uint32\_t **ClearINT**);
- ◆ void EPHC\_ASetRunState(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t **Cmd**);
- ◆ void EPHC\_AClearPulseCntValue(TSB\_EPHC\_TypeDef \* **EPHCx**);
- ◆ uint16\_t EPHC\_AGetCompareValue(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t **CmpReg**);
- ◆ void EPHC\_ASetCompareValue(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t **CmpReg**,  
uint16\_t **CmpValue**);
- ◆ uint16\_t EPHC\_AGetPulseCntValue(TSB\_EPHC\_TypeDef \* **EPHCx**);
- ◆ void EPHC\_BSetRunState(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t **Cmd**);
- ◆ void EPHC\_BSetDMAReq(TSB\_EPHC\_TypeDef \* **EPHCx**, FunctionalState  
**NewState**, uint8\_t **DMAReq**);
- ◆ uint32\_t EPHC\_BGetReadCntValue(TSB\_EPHC\_TypeDef \* **EPHCx**);
- ◆ uint32\_t EPHC\_BGetCapRegValue(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t  
**CapReg**);
- ◆ uint32\_t EPHC\_BGetCapRegOverflow(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t  
**CapReg**);
- ◆ uint32\_t EPHC\_BGetCycleCntRegValue(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t  
**CntReg**);

◆ uint32\_t EPHC\_BGetPhaseDiffRegValue(TSB\_EPHC\_TypeDef \* **EPHCx**, uint8\_t **PhaseReg**);

## 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) 各 EPHC チャンネルの共通機能を設定と制御:  
EPHC\_Enable (), EPHC\_Disable (), EPHC\_Init(), EPHC\_ASetRunState(),  
EPHC\_BSetRunState()
- 2) 各 EPHC チャンネルの状態表示:  
EPHC\_GetINTFactor(), EPHC\_AGetPulseCntValue(), EPHC\_AGetCompareValue(),  
EPHC\_BgetReadCntValue(), EPHC\_BGetCapRegValue(), EPHC\_BgetCapRegOverflow(),  
EPHC\_BgetCycleCntRegValue(), EPHC\_BgetPhaseDiffRegValue()
- 3) その他機能の設定:  
EPHC\_ClearINTFactor(), EPHC\_EnableInterrupt(), EPHC\_DisableInterrupt(),  
EPHC\_AClearPulseCntValue(), EPHC\_ASetCompareValue(), EPHC\_BSetDMAReq()

## 7.2.3 関数仕様

補足: 下記すべての API において、パラメーター“TSB\_EPHC\_TypeDef \* **EPHCx**”には、下記いずれかの値が入ります。

TSB\_EPHC

### 7.2.3.1 EPHC\_Enable

EPHC の動作許可

関数のプロトタイプ宣言:

void  
EPHC\_Enable(TSB\_EPHC\_TypeDef\* **EPHCx**)

引数:

**EPHCx**: EPHC チャンネルを選択します。

機能:

EPHC の動作を許可します。

戻り値:

なし

### 7.2.3.2 EPHC\_Disable

EPHC の動作禁止

**関数のプロトタイプ宣言:**

void

EPHC\_Disable(TSB\_EPHC\_TypeDef\* **EPHCx**)

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**機能:**

EPHC の動作を禁止します。

**戻り値:**

なし

### 7.2.3.3 EPHC\_ASetRunState

16 ビットカウンタの RUN/STOP 制御

**関数のプロトタイプ宣言:**

void

EPHC\_ASetRunState(TSB\_EPHC\_TypeDef \* **EPHCx**,  
uint32\_t **Cmd**);

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**Cmd**: 以下からアップダウンカウンタ用コマンドを選択します。

- **EPHC\_RUN**: RUN
- **EPHC\_STOP**: STOP

**機能:**

16 ビットカウンタの RUN/STOP を制御します。

**戻り値:**

なし

### 7.2.3.4 EPHC\_Init

EPHC の初期化

**関数のプロトタイプ宣言:**

void

```
EPHC_Init (TSB_EPHC_TypeDef * EPHCx,  
           EPHC_InitTypeDef * InitStruct);
```

引数:

**EPHCx**: EPHC チャンネルを選択します。

**InitStruct** は EPHC 基本構成を含む構造体です。(詳細は“データ構成説明”を参照)

機能:

EPHC を初期化します。

戻り値:

なし

### 7.2.3.5 EPHC\_GetINTFactor

割り込み要因の取得

関数のプロトタイプ宣言:

```
EPHC_INTFactor
```

```
EPHC_GetINTFactor(TSB_EPHC_TypeDef* EPHCx)
```

引数:

**EPHCx**: EPHC チャンネルを選択します。

機能:

ISR において割り込み要因を取得します。

戻り値:

EPHC 割り込み要因。各ビットの意味は次の通りです。

**Compare0** (Bit0): 2 相パルスコンペア 0 一致割り込み

**Compare1** (Bit1): 2 相パルスコンペア 1 一致割り込み

**Overflow** (Bit2): 2 相パルス 16 ビットカウンタオーバーフロー

**Underflow** (Bit3): 2 相パルス 16 ビットカウンタアンダーフロー

**IN0Falling**(Bit4): EPHCxIN0 立下りエッジ

**IN1Falling**(Bit5): EPHCxIN1 立下りエッジ

**IN0Rising**(Bit6): EPHCxIN0 立ち上がりエッジ

**IN1Rising**(Bit7): EPHCxIN1 立ち上がりエッジ

**CycleMeasurement** (Bit8): 2 相パルス周期位相測定周期エラー

### 7.2.3.6 EPHC\_ClearINTFactor

割り込み要因フラグのクリア

**関数のプロトタイプ宣言:**

```
void  
EPHC_ClearINTFactor(TSB_EPHC_TypeDef * EPHCx,  
                    uint32_t ClearINT)
```

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**ClearINT**: 以下からクリアする EPHC 割り込み要因を選択します。有効ビットの組み合わせが可能です。

- **EPHC\_FLG\_CMP0F**: 2 相パルスコンペア 0 一致割り込み
- **EPHC\_FLG\_CMP1F**: 2 相パルスコンペア 1 一致割り込み
- **EPHC\_FLG\_OVFF**: 2 相パルス 16 ビットオーバーフロー
- **EPHC\_FLG\_UBFF**: 2 相パルス 16 ビットアンダーフロー
- **EPHC\_FLG\_SB0F**: EPHCxIN0 立下リエッジ
- **EPHC\_FLG\_SB1F**: EPHCxIN1 立下リエッジ
- **EPHC\_FLG\_SB2F**: EPHCxIN0 立ち上がりエッジ
- **EPHC\_FLG\_SB3F**: EPHCxIN1 立ち上がりエッジ
- **EPHC\_FLG\_DIRF**: 2 相パルス周期位相測定周期エラー
- **EPHC\_FLG\_ALL**: 全割り込み要因

**機能:**

各割り込み要因は自動的にクリアされないため、本 API で割り込み要因をクリアします。

**戻り値:**

なし

**補足:**

各割り込み要因は自動的にクリアされないため、使用前に初期化してください。

## 7.2.3.7 EPHC\_EnableInterrupt

EPHC 割り込みの許可

**関数のプロトタイプ宣言:**

```
void  
EPHC_EnableInterrupt(TSB_EPHC_TypeDef * EPHCx,  
                     uint32_t EnableINT);
```

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**EnableINT**: 以下から許可する EPHC 割り込みを選択します。有効ビットの組み合わせが可能です。

- **EPHC\_IE\_INT\_PCCP0**: 2 相パルスコンペアー一致 0

- EPHC\_IE\_INT\_PCCP1: 2 相パルスコンペアー致 1
- EPHC\_IE\_INT\_PCOVF: 2 相パルス 16 ビットカウンタオーバーフロー割り込み
- EPHC\_IE\_INT\_PCUDF: 2 相パルス 16 ビットカウンタアンダーフロー割り込み
- EPHC\_IE\_INT\_PCDT0: 2 相パルス周期位相測定周期 0
- EPHC\_IE\_INT\_PCDT1: 2 相パルス周期位相測定周期 1
- EPHC\_IE\_INT\_PCDT2: 2 相パルス周期位相測定周期 2
- EPHC\_IE\_INT\_PCDT3: 2 相パルス周期位相測定周期 3
- EPHC\_IE\_INT\_PCDIR: 2 相パルス周期位相測定周期エラー割り込み
- EPHC\_IE\_INT\_PCUOVF: 2 相パルス周期位相測定オーバーフロー割り込み
- EPHC\_IE\_INT\_ALL: 全割り込み

**機能:**

EPHC 割り込みを許可します。

**戻り値:**

なし

## 7.2.3.8 EPHC\_DisableInterrupt

EPHC 割り込みの禁止

**関数のプロトタイプ宣言:**

void

```
EPHC_DisableInterrupt(TSB_EPHC_TypeDef * EPHCx,  
                      uint32_t DisableINT);
```

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**DisableINT**: 以下から禁止する EPHC 割り込みを選択します。有効ビットの組み合わせが可能です。

- EPHC\_IE\_INT\_PCCP0: 2 相パルスコンペアー致 0
- EPHC\_IE\_INT\_PCCP1: 2 相パルスコンペアー致 1
- EPHC\_IE\_INT\_PCOVF: 2 相パルス 16 ビットカウンタオーバーフロー割り込み
- EPHC\_IE\_INT\_PCUDF: 2 相パルス 16 ビットカウンタアンダーフロー割り込み
- EPHC\_IE\_INT\_PCDT0: 2 相パルス周期位相測定周期 0
- EPHC\_IE\_INT\_PCDT1: 2 相パルス周期位相測定周期 1
- EPHC\_IE\_INT\_PCDT2: 2 相パルス周期位相測定周期 2
- EPHC\_IE\_INT\_PCDT3: 2 相パルス周期位相測定周期 3
- EPHC\_IE\_INT\_PCDIR: 2 相パルス周期位相測定周期エラー割り込み
- EPHC\_IE\_INT\_PCUOVF: 2 相パルス周期位相測定オーバーフロー割り込み
- EPHC\_IE\_INT\_ALL: 全割り込み

**機能:**



EPHC 割り込みを禁止します。

戻り値:

なし

## 7.2.3.9 EPHC\_AGetPulseCntValue

2 相パルス入力カウンタ値のリード

関数のプロトタイプ宣言:

uint16\_t

EPHC\_AGetPulseCntValue(TSB\_EPHC\_TypeDef \* **EPHCx**);

引数:

**EPHCx**: EPHC チャンネルを選択します。

機能:

2 相パルス入力カウンタ値をリードします。

戻り値:

EPHC カウンター値

## 7.2.3.10 EPHC\_AClearPulseCntValue

2 相パルス入力カウンタのクリア

関数のプロトタイプ宣言:

void

EPHC\_AClearPulseCntValue(TSB\_EPHC\_TypeDef \* **EPHCx**);

引数:

**EPHCx**: EPHC チャンネルを選択します。

機能:

2 相パルス入力カウンタをクリアします。

戻り値:

なし

## 7.2.3.11 EPHC\_AGetCompareValue

コンペア値のリード

**関数のプロトタイプ宣言:**

```
uint16_t  
EPHC_AGetCompareValue(TSB_EPHC_TypeDef * EPHCx,  
                      uint8_t CmpReg)
```

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**CmpReg**: 以下のいずれからコンペアレジスタを選択します。

- **EPHC\_COMP\_0**: コンペアレジスタ 0
- **EPHC\_COMP\_1**: コンペアレジスタ 1

**機能:**

コンペア値をリードします。

**戻り値:**

コンペア値

## 7.2.3.12 EPHC\_ASetCompareValue

コンペア値の設定

**関数のプロトタイプ宣言:**

```
void  
EPHC_SetCompareValue(TSB_EPHC_TypeDef * EPHCx,  
                    uint8_t CmpReg,  
                    uint16_t CmpValue);
```

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**CmpReg**: 以下のいずれかのコンペアレジスタを選択します。

- **EPHC\_COMP\_0**: コンペアレジスタ 0
- **EPHC\_COMP\_1**: コンペアレジスタ 1

**CmpValue**: 設定する 2 相パルス入力カウンタコンペア値を設定します。

**機能:**

コンペア値を設定します。

**戻り値:**

なし

## 7.2.3.13 EPHC\_BSetRunState

24 ビットカウンタの RUN/STOP 制御

関数のプロトタイプ宣言:

void

```
EPHC_BSetRunState(TSB_EPHC_TypeDef * EPHCx,  
                  uint32_t Cmd);
```

引数:

**EPHCx**: EPHC チャンネルを選択します。

**Cmd**: 以下のいずれかのアップダウンカウンタ用コマンドを選択します。

- **EPHC\_RUN**: カウントスタート
- **EPHC\_STOP**: 停止&クリア

機能:

24 ビットカウンタの RUN/STOP を制御します。

戻り値:

なし

## 7.2.3.14 EPHC\_BSetDMAReq

INTPHCY0 による DMAC 起動要求の設定

関数のプロトタイプ宣言:

void

```
EPHC_SetDMAReq(TSB_EPHC_TypeDef * EPHCx,  
               FunctionalState NewState,  
               uint8_t DMAReq);
```

引数:

**EPHCx**: EPHC チャンネルを選択します。

**NewState**: 以下のいずれかの DMAC 起動要求を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

**DMAReq**: 外部入力の DMA 要求を設定します。以下を指定してください。

- **EPHC\_DMA\_REQ\_CAPTURE\_0**: DMA 要求入力キャプチャ 0(INTPHCY0)

機能:

INTPHCY0 による DMAC 起動要求を設定します。

戻り値:

なし

**Note:**

割り込み要求を禁止している場合、DMAC 起動要求を許可しても DMAC 要求は発行されません。

## 7.2.3.15 EPHC\_BGetReadCntValue

24 ビットカウンタ値の取得

**関数のプロトタイプ宣言:**

Uint32\_t

EPHC\_BGetReadCntValue (TSB\_EPHC\_TypeDef \* **EPHCx**);

**引数:**

**EPHCx:** EPHC チャンネルを選択します。

**機能:**

24 ビットカウンタ値を取得します。

**戻り値:**

24 ビットカウンターの値

## 7.2.3.16 EPHC\_BGetCapRegValue

キャプチャ値の取得

**関数のプロトタイプ宣言:**

Uint32\_t

EPHC\_BGetCapRegValue (TSB\_EPHC\_TypeDef \* **EPHCx**,  
uint8\_t CapReg)

**引数:**

**EPHCx:** EPHC チャンネルを選択します。

**CapReg:** 以下のいずれかのキャプチャレジスタを選択します。

- **EPHC\_BCAP00\_IN0RISING:** EPHCxIN0 立ち上がりエッジ
- **EPHC\_BCAP10\_IN1RISING:** EPHCxIN1 立ち上がりエッジ
- **EPHC\_BCAP20\_IN0FALLING:** EPHCxIN0 立下りのエッジ
- **EPHC\_BCAP30\_IN1FALLING:** EPHCxIN1 立下りのエッジ

**機能:**

2 相パルス入力カウンタのキャプチャレジスタの値を取得します。

戻り値:

キャプチャレジスタの値

## 7.2.3.17 EPHC\_BGetCapRegOverflow

オーバーフロー発生の有無の取得

関数のプロトタイプ宣言:

Uint32\_t

EPHC\_BGetCapRegOverflow (TSB\_EPHC\_TypeDef \* **EPHCx**,  
uint8\_t CapReg)

引数:

**EPHCx**: EPHC チャンネルを選択します。

**CapReg**: 以下のいずれかのキャプチャレジスタを選択します。

- **EPHC\_BCAP00\_IN0RISING**: EPHCxIN0 立ち上がりエッジ
- **EPHC\_BCAP10\_IN1RISING**: EPHCxIN1 立ち上がりエッジ
- **EPHC\_BCAP20\_IN0FALLING**: EPHCxIN0 立下りエッジ
- **EPHC\_BCAP30\_IN1FALLING**: EPHCxIN1 立下りエッジ

機能:

キャプチャレジスタがオーバーフローの状態であるかを取得します。

戻り値:

0: オーバーフロー発生なし

1: オーバーフロー発生あり

## 7.2.3.18 EPHC\_BGetCycleCntRegValue

周期カウンタ値の取得

関数のプロトタイプ宣言:

Uint32\_t

EPHC\_BGetCycleCntRegValue (TSB\_EPHC\_TypeDef \* EPHCx,  
uint8\_t CntReg)

引数:

**EPHCx**: EPHC チャンネルを選択します。

**CntReg**: 以下のいずれかのキャプチャレジスタを選択します。

- **EPHC\_B0DAT\_IN0RISING**: EPHCxIN0 の立ち上がりエッジ間の周期カウンタ値
- **EPHC\_B1DAT\_IN1RISING**: EPHCxIN0 の立ち上がりエッジ間の周期カウンタ値

- **EPHC\_B2DAT\_IN0FALLING**: EPHCxIN0 の立下りエッジ間の周期カウント値
- **EPHC\_B3DAT\_IN1FALLING**: EPHCxIN1 の立下りエッジ間の周期カウント値
- **EPHC\_BCDAT\_COMMON**: EPHCxB0DAT ~ EPHCxB3DAT の取り込み時の周期カウント値

**機能:**

周期カウント値を取得します。

**戻り値:**

サイクルカウント数

## 7.2.3.19 EPHC\_BGetPhaseDiffRegValue

位相差周期カウンタ値の取得

**関数のプロトタイプ宣言:**

Uint32\_t

EPHC\_BGetPhaseDiffRegValue (TSB\_EPHC\_TypeDef \* **EPHCx**,  
uint8\_t PhaseReg)

**引数:**

**EPHCx**: EPHC チャンネルを選択します。

**CntReg**: 以下のいずれかの EPHCx 位相差レジスタを選択します。

- **EPHC\_B0PDT**: EPHCx 位相差 0 レジスタ
- **EPHC\_B1PDT**: EPHCx 位相差 1 レジスタ
- **EPHC\_B2PDT**: EPHCx 位相差 2 レジスタ
- **EPHC\_B3PDT**: EPHCx 位相差 3 レジスタ

**機能:**

位相差周期カウンタ値を取得します。

**戻り値:**

位相差周期カウンタ値

## 7.2.4 データ構造

### 7.2.4.1 EPHC\_InitTypeDef

**メンバ:**

uint32\_t

**CountDownEdgeSelfForP1**: 2 相パルス入力カウンタ機能での 1 相パルスカウンタモードのカウンタダウンエッジを選択します。

- **EPHC\_CNT\_MA1DN\_NOCNTDOWN0**: EPHCxIN0、EPHCxIN1 が変化してもカウントダウンしません
- **EPHC\_CNT\_MA1DN\_IN0RISING**: EPHCxIN0 の立ち上がりエッジ
- **EPHC\_CNT\_MA1DN\_IN1RISING**: EPHCxIN1 の立ち上がりエッジ
- **EPHC\_CNT\_MA1DN\_IN0FALLING**: EPHCxIN0 の立下りエッジ
- **EPHC\_CNT\_MA1DN\_IN1FALLING**: EPHCxIN1 の立下りエッジ
- **EPHC\_CNT\_MA1DN\_IN0BOTH**: EPHCxIN0 の両エッジ
- **EPHC\_CNT\_MA1DN\_IN1BOTH**: EPHCxIN1 の両エッジ
- **EPHC\_CNT\_MA1DN\_NOCNTDOWN7**: EPHCxIN0、EPHCxIN1 が変化してもカウントダウンしません

uint32\_t

**CountUpEdgeSelForP1**: 2 相パルス入力カウンタ機能での 1 相パルスカウンタモードのカウントアップエッジを、以下のいずれかより選択します。

- **EPHC\_CNT\_MA1UP\_NOCNTUP0**: EPHCxIN0、EPHCxIN1 が変化してもカウントアップしません
- **EPHC\_CNT\_MA1UP\_IN0RISING**: EPHCxIN0 の立ち上がりエッジ
- **EPHC\_CNT\_MA1UP\_IN1RISING**: EPHCxIN1 の立ち上がりエッジ
- **EPHC\_CNT\_MA1UP\_IN0FALLING**: EPHCxIN0 の立下りエッジ
- **EPHC\_CNT\_MA1UP\_IN1FALLING**: EPHCxIN1 の立下りエッジ
- **EPHC\_CNT\_MA1UP\_IN0BOTH**: EPHCxIN0 の両エッジ
- **EPHC\_CNT\_MA1UP\_IN1BOTH**: EPHCxIN0 の両エッジ
- **EPHC\_CNT\_MA1UP\_NOCNTUP7**: EPHCxIN0、EPHCxIN1 が変化してもカウントアップしません

uint32\_t

**InputClkSelection**: 以下のいずれかの 24 ビットカウンタの入クロックを選択します。

- **EPHC\_CNT\_BRCK\_FC**: fc
- **EPHC\_CNT\_BRCK\_FC\_2**: fc/2
- **EPHC\_CNT\_BRCK\_FC\_4**: fc/4
- **EPHC\_CNT\_BRCK\_FC\_8**: fc/8

uint32\_t

**PhaseSelection**: 周期位相差測定機能の位相を選択します。

- **EPHC\_CNT\_PBDIR\_POSITIVEPHASE**: 正位相
- **EPHC\_CNT\_PBDIR\_NEGATIVEPHASE**: 逆位相

uint32\_t

**ModeSetting**: 2 相パルスカウンタ機能でのモードを、以下のいずれかより選択します。

- **EPHC\_CNT\_MA12\_PULSE\_2**: 2 相パルスカウンタモード
- **EPHC\_CNT\_MA12\_PULSE\_1**: 1 相パルスカウンタモード

uint32\_t

**DirectionSetting:** 2 相パルスカウント機能での 2 相パルスカウンタモードの方向を選択します。

- EPHC\_CNT\_MA2DIR\_POSITIVEDIR: 正方向
- EPHC\_CNT\_MA2DIR\_NEGATIVEDIR: 逆方向

uint32\_t

**NoiseFilterCtrl:** EPHCxIN0、EPHCxIN1 のノイズ除去時間を、以下のいずれかより選択します。

- EPHC\_CNT\_NOISEFILTER\_NONE: なし
- EPHC\_CNT\_NOISEFILTER\_2: 2/fsys 以下の信号をノイズとして取り除きます。
- EPHC\_CNT\_NOISEFILTER\_4: 4/fsys 以下の信号をノイズとして取り除きます。

uint32\_t

**CountClearCtrl:** 2 相パルス入力カウンタをクリアする/しないを選択します。

- EPHC\_COUNT\_CONTINUE: Don't care
- EPHC\_COUNT\_CLR: クリア

## 7.2.4.2 EPHC\_INTFactor

メンバ:

uint32\_t

**All:** EPHC 割り込み状態

**Bit**

uint32\_t

**Compare0:** 1 2 相パルスコンペア 0 一致割り込み

uint32\_t

**Compare1:** 1 2 相パルスコンペア 1 一致割り込み

uint32\_t

**Overflow:** 1 2 相パルス 16 ビットカウンタオーバーフロー

uint32\_t

**UnderFlow:** 1 2 相パルス 16 ビットカウンタアンダーフロー

uint32\_t

**IN0Falling:** 1 EPHCxIN0 立下リエッジ

uint32\_t

**IN1Falling:** 1 EPHCxIN1 立下リエッジ

uint32\_t

**IN0Rising:** 1 EPHCxIN0 立ち上がりエッジ

uint32\_t

**IN0Rising:** 1 EPHCxIN1 立ち上がりエッジ

uint32\_t

**CycleMeasurement:** 1 2 相パルス周期位相測定周期エラー



uint32\_t

**Reserverd** : 23 Reserverd

## 8. ESIO

### 8.1 概要

本デバイスは、3 チャンネルの拡張シリアル I/O(ESIO)を内蔵しています。ESIO は全二重同期通信が可能なシリアルインタフェースで SPI モードと SIO モードの 2 種類の通信方法を備えており、様々な周辺デバイスとの高速なシリアル転送が可能です。

ESIO は、1 ユニットにチップセレクト信号(ESIOxCS0、ESIOxCS1)、シリアルクロック信号(ESIOxSCK)、および送受信信号線(ESIOxTXD0~3 と ESIOxRXD0~3) を備えています。8 本の送受信信号線は送信 1 本と受信 1 本とが 1 組として扱われ、1~4 組(以降ラインと称する)まで任意に構成できます。これにより、転送レートはシリアルクロック速度の 4 倍まで可能です。また、データ長を 8 ビットから 32 ビットまで 1 ビット単位で可変することができます。

### 8.2 API 関数

#### 8.2.1 関数一覧

- ◆ void ESIO\_SWReset(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ void ESIO\_Enable(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ void ESIO\_Disable(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ void ESIO\_SetTxRxCtrl(TSB\_ESIO\_TypeDef \* ESIOx, FunctionalState NewState);
- ◆ FunctionalState ESIO\_GetTxRxCtrl(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ void ESIO\_SelectMode(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_Mode Mode);
- ◆ void ESIO\_SelectTransferMode(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_TransferMode TrMode);
- ◆ void ESIO\_SetCS(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_CSx CSx);
- ◆ void ESIO\_SetTransferNum(TSB\_ESIO\_TypeDef \* ESIOx, uint8\_t Num);
- ◆ void ESIO\_SetTxPinInIdle(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_TxPinInIdle PinMode);
- ◆ void ESIO\_SetFIFOLevelINT(TSB\_ESIO\_TypeDef \* ESIOx, uint8\_t TxRx, uint8\_t Level);
- ◆ void ESIO\_SetDMA(TSB\_ESIO\_TypeDef \* ESIOx, uint8\_t TxRx, FunctionalState NewState);
- ◆ void ESIO\_SetINT(TSB\_ESIO\_TypeDef \* ESIOx, uint32\_t IntSrc, FunctionalState NewState);
- ◆ void ESIO\_InitFIFO(TSB\_ESIO\_TypeDef \* ESIOx, uint8\_t TxRx);
- ◆ void ESIO\_SetBaudRate(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_BaudClock Clk, uint8\_t Divider);
- ◆ void ESIO\_Init(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_InitTypeDef \* Init);
- ◆ void ESIO\_SetTxData(TSB\_ESIO\_TypeDef \* ESIOx, uint32\_t dat);
- ◆ uint32\_t ESIO\_GetRxData(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ FunctionalState ESIO\_IsRegisterModifiable(TSB\_ESIO\_TypeDef \* ESIOx);

- ◆ ESIO\_StatusFlag ESIO\_GetStatus(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ ESIO\_ParityErrNum ESIO\_GetParityErrorFlag(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ void ESIO\_ClrAllParityErrFlag(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ uint32\_t ESIO\_GetHorizontalParityError(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_LINEx Line);
- ◆ void ESIO\_ClrHorizontalParityError(TSB\_ESIO\_TypeDef \* ESIOx, ESIO\_LINEx Line);
- ◆ ESIO\_VerticalParityErrFrame ESIO\_GetVerticalParityErrFrame(TSB\_ESIO\_TypeDef \* ESIOx);
- ◆ void ESIO\_ClrAllVerticalParityErrNum(TSB\_ESIO\_TypeDef \* ESIOx);

## 8.2.2 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) 初期化と共通設定:  
ESIO\_SWReset(), ESIO\_Enable(), ESIO\_Disable(), ESIO\_SelectMode(),  
ESIO\_SelectTransferMode(), ESIO\_SetTxPinInIdle(), ESIO\_Init(),  
ESIO\_SetTransferNum(), ESIO\_SetBaudRate()
- 2) FIFO、DMA の設定:  
ESIO\_SetFIFOLevelINT(), ESIO\_SetDMA(), ESIO\_InitFIFO()
- 3) 転送制御:  
ESIO\_SetTxData(), ESIO\_GetRxData(), ESIO\_SetTxRxCtr(), ESIO\_SetCS()
- 4) 状態リード:  
ESIO\_GetStatus(), ESIO\_GetParityErrorFlag(), ESIO\_ClrAllParityErrFlag(),  
ESIO\_GetHorizontalParityError(), ESIO\_ClrHorizontalParityError(),  
ESIO\_GetVerticalParityErrFrame(), ESIO\_ClrAllVerticalParityErrNum(),  
ESIO\_IsRegisterModifiable(), ESIO\_GetTxRxCtrl()
- 5) 割り込み制御:  
ESIO\_SetINT()

## 8.2.3 関数仕様

**補足:** 下記全 API において、パラメーター“TSB\_ESIO\_TypeDef \* ESIOx” は、下記いずれかの値になります。

TSB\_ESIO0, TSB\_ESIO1, TSB\_ESIO2

### 8.2.3.1 ESIO\_SWReset

ESIO のソフトウェアリセット

**関数のプロトタイプ宣言:**

```
void  
ESIO_SWReset(TSB_ESIO_TypeDef * ESIOx)
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

ESIO をソフトウェアリセットします。

**戻り値:**

なし

## 8.2.3.2 ESIO\_Enable

ESIO 動作の許可

**関数のプロトタイプ宣言:**

void

ESIO\_Enable(TSB\_ESIO\_TypeDef \* ESIOx)

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

ESIO 動作を許可します。

**戻り値:**

なし

**補足:**

本 API をコールしてから初期設定および通信を行ってください。

## 8.2.3.3 ESIO\_Disable

ESIO 動作の禁止

**関数のプロトタイプ宣言:**

void

ESIO\_Disable(TSB\_ESIO\_TypeDef \* ESIOx)

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

ESIO 動作を禁止します。

**戻り値:**

なし

## 8.2.3.4 ESIO\_SetTxRxCtrl

通信制御の設定

**関数のプロトタイプ宣言:**

void

ESIO\_SetTxRxCtrl (TSB\_ESIO\_TypeDef \* ESIOx,  
FunctionalState NewState)

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**NewState:** 以下より通信許可/禁止を選択します。

- **ENABLE:** 通信許可
- **DISABLE:** 通信停止

**機能:**

通信制御の設定を行います。

**戻り値:**

なし

## 8.2.3.5 ESIO\_GetTxRxCtrl

通信制御の設定状態取得

**関数のプロトタイプ宣言:**

FunctionalState

ESIO\_GetTxRxCtrl(TSB\_ESIO\_TypeDef \* ESIOx)

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

通信制御の設定状態を取得します。

**戻り値:**

通信状態の設定状態:

**ENABLE:** 通信許可

**DISABLE:** 通信禁止

## 8.2.3.6 ESIO\_SelectMode

ESIO モードの選択

**関数のプロトタイプ宣言:**

void

```
ESIO_SelectMode(TSB_ESIO_TypeDef * ESIOx,  
                ESIO_Mode Mode)
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**Mode:** 以下より SPI モード、または SIO モードを選択します。

- **ESIO\_MODE\_SPI:** SPI モード
- **ESIO\_MODE\_SIO:** SIO モード

**機能:**

ESIO モード(SIO または SPI)を選択します。

**戻り値:**

なし

## 8.2.3.7 ESIO\_SelectTransferMode

転送モードの選択

**関数のプロトタイプ宣言:**

void

```
ESIO_SelectTransferMode(TSB_ESIO_TypeDef * ESIOx,  
                        ESIO_TransferMode TrMode)
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**TrMode:** 以下より転送モードを選択します。

- **ESIO\_TRMODE\_TX:** 送信のみ
- **ESIO\_TRMODE\_RX:** 受信のみ
- **ESIO\_TRMODE\_TXRX:** 全二重通信

**機能:**

転送モードを選択します。

**戻り値:**

なし

## 8.2.3.8 ESIO\_SetCS

チップセレクト信号の選択

**関数のプロトタイプ宣言:**

```
void  
ESIO_SetCS(TSB_ESIO_TypeDef * ESIOx,  
            ESIO_CSx CSx)
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**CSx:** 以下よりチップセレクト信号を選択します。

- **ESIO\_CS0:** ESIOxCS0
- **ESIO\_CS1:** ESIOxCS1

**機能:**

チップセレクト信号を選択します。

**戻り値:**

なし

## 8.2.3.9 ESIO\_SetTxPinInIdle

ESIOxTXD のアイドル時の出力値固定機能の設定

**関数のプロトタイプ宣言:**

```
void  
ESIO_SetTxPinInIdle(TSB_ESIO_TypeDef * ESIOx,  
                    ESIO_TxPinInIdle PinMode);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**PinMode:** ESIOxTXD のアイドル時の出力値固定機能を選択します。

- **ESIO\_TXPIN\_FINALBIT:** 最終ビットを保持
- **ESIO\_TXPIN\_KEEPLow:** Low レベル出力を保持
- **ESIO\_TXPIN\_KEEPhigh:** High レベル出力を保持

**機能:**

ESIOxTXD のアイドル時の出力値固定機能を設定します。

**戻り値:**

なし

## 8.2.3.10 ESIO\_SetFIFOLevelINT

送受信 Fill レベルの設定

**関数のプロトタイプ宣言:**

void

ESIO\_SetFIFOLevelINT(TSB\_ESIO\_TypeDef \* ESIOx,  
uint8\_t TxRx, uint8\_t Level)

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**TxRx:** 以下より送信あるいは受信のいずれかを選択します。

- **ESIO\_TX:** 送信
- **ESIO\_RX:** 受信

**Level:** 0 から 8 の範囲で Fill レベルを設定します。(LINE の数とフレーム長に依存します)

**機能:**

送受信 Fill レベルを設定します。

**戻り値:**

なし

## 8.2.3.11 ESIO\_SetDMA

送受信 DMA 要求の許可/禁止

**関数のプロトタイプ宣言:**

void

ESIO\_SetDMA(TSB\_ESIO\_TypeDef \* ESIOx,  
uint8\_t TxRx, FunctionalState NewState)

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**TxRx:** 以下より送信または受信のいずれかを選択します。

- **ESIO\_TX:** 送信
- **ESIO\_RX:** 受信

**NewState:** 以下より DMA 要求の許可または/禁止のいずれかを選択します。

- **ENABLE:** 許可
- **DISABLE:** 禁止

**機能:**

送受信 DMA 要求を設定します。



戻り値:

なし

## 8.2.3.12 ESIO\_SetINT

割り込み要因の許可/禁止

関数のプロトタイプ宣言:

void

```
ESIO_SetINT(TSB_ESIO_TypeDef * ESIOx,  
            uint32_t IntSrc,  
            FunctionalState NewState)
```

引数:

**ESIOx**: ESIO チャンネルを選択します。

**IntSrc**: 以下よりいずれかの割り込み要因を選択します。

- **ESIO\_INT\_PERR**: パリティエラー割り込み
- **ESIO\_INT\_RXEND**: 受信完了割り込み
- **ESIO\_INT\_RXFIFO**: 受信 FIFO 割り込み
- **ESIO\_INT\_TXEND**: 送信完了制御割り込み
- **ESIO\_INT\_TXFIFO**: 送信 FIFO 割り込み
- **ESIO\_INT\_ALL**: 上記全割り込み要因

**NewState**: 以下より割り込み許可/禁止のいずれかを選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

割り込み要因の許可/禁止を設定します。

戻り値:

なし

## 8.2.3.13 ESIO\_InitFIFO

送受信バッファのクリア

関数のプロトタイプ宣言:

void

```
ESIO_InitFIFO(TSB_ESIO_TypeDef * ESIOx, uint8_t TxRx);
```

引数:

**ESIOx**: ESIO チャンネルを選択します。

**TxRx:** 以下より送信または受信のいずれかを選択します。

- **ESIO\_TX:** 送信
- **ESIO\_RX:** 受信

**機能:**

送受信バッファをクリアします。

**戻り値:**

なし

## 8.2.3.14 ESIO\_SetBaudRate

ボーレートジェネレーターの入力クロックおよび分周値"N"の設定

**関数のプロトタイプ宣言:**

```
void  
ESIO_SetBaudRate(TSB_ESIO_TypeDef * ESIOx,  
                  ESIO_BaudClock Clk,  
                  uint8_t Divider);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**Clk:** 以下よりボーレートジェネレーターの入力クロックを選択します。

- **ESIO\_PHIT0\_DIVIDE\_2** : T0/2
- **ESIO\_PHIT0\_DIVIDE\_4** : T0/4
- **ESIO\_PHIT0\_DIVIDE\_8** : T0/8
- **ESIO\_PHIT0\_DIVIDE\_16** : T0/16
- **ESIO\_PHIT0\_DIVIDE\_32** : T0/32
- **ESIO\_PHIT0\_DIVIDE\_64** : T0/64
- **ESIO\_PHIT0\_DIVIDE\_128** : T0/128
- **ESIO\_PHIT0\_DIVIDE\_256** : T0/256
- **ESIO\_PHIT0\_DIVIDE\_512** : T0/512
- **ESIO\_PHIT0\_DIVIDE\_1024** : T0/1024

**Divider:** 1～16 の範囲でボーレートジェネレータの分周値"N"を設定します。

**機能:**

ボーレートジェネレーターの入力クロックおよび分周値"N"を設定します。

**戻り値:**

なし

## 8.2.3.15 ESIO\_Init

ESIO チャンネルの初期化

**関数のプロトタイプ宣言:**

```
void  
ESIO_Init(TSB_ESIO_TypeDef * ESIOx,  
          ESIO_InitTypeDef * Init);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**Init:** ESIO 基本構成を含む構造体です。(詳細は"データ構造"を参照してください)

**機能:**

ESIO チャンネルの初期化を行います。

**戻り値:**

なし

**補足:**

まず ESIO\_SetTransferNum()をコールしてください。

## 8.2.3.16 ESIO\_SetTxData

送信 FIFO へのデータ書き込み

**関数のプロトタイプ宣言:**

```
void  
ESIO_SetTxData(TSB_ESIO_TypeDef * ESIOx,  
               uint32_t dat);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**dat:** 送信データ

**機能:**

送信 FIFO へのデータを書き込みます。

**戻り値:**

なし

## 8.2.3.17 ESIO\_GetRxData

受信 FIFO からのデータ読み出し

**関数のプロトタイプ宣言:**

```
uint32_t  
ESIO_GetRxData(TSB_ESIO_TypeDef * ESIOx);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

受信 FIFO からのデータを読み出します。

**戻り値:**

受信データ

## 8.2.3.18 ESIO\_IsRegisterModifiable

設定可能状態フラグの読み出し

**関数のプロトタイプ宣言:**

```
FunctionalState  
ESIO_IsRegisterModifiable(TSB_ESIO_TypeDef * ESIOx);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

設定可能状態フラグを読み出します。

**戻り値:**

設定可能状態フラグ:

- **ENABLE:** 設定可能状態
- **DISABLE:** 設定禁止状態

## 8.2.3.19 ESIO\_GetStatus

ステータスの取得

**関数のプロトタイプ宣言:**

```
ESIO_StatusFlag  
ESIO_GetStatus(TSB_ESIO_TypeDef * ESIOx);
```

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

ステータスを取得します。

**戻り値:**

ステータスを表す構造体"ESIO\_StatusFlag" (詳細は"データ構造"を参照してください)

## 8.2.3.20 ESIO\_GetParityErrorFlag

パリティエラーフラグの読み出し

**関数のプロトタイプ宣言:**

ESIO\_ParityErrNum

ESIO\_GetParityErrorFlag(TSB\_ESIO\_TypeDef \* ESIOx);

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

パリティエラーフラグを読み出します。

**戻り値:**

パリティエラーフラグを表す構造体"ESIO\_ParityErrNum" (詳細は"データ構造"を参照してください)

## 8.2.3.21 ESIO\_ClrAllParityErrFlag

全パリティエラーフラグのクリア

**関数のプロトタイプ宣言:**

void

ESIO\_ClrAllParityErrFlag(TSB\_ESIO\_TypeDef \* ESIOx);

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

すべてのパリティエラーフラグをクリアします。

戻り値:  
なし

## 8.2.3.22 ESIO\_GetHorizontalParityError

受信ラインの水平パリティエラーフラグの読み出し

関数のプロトタイプ宣言:

```
uint32_t  
ESIO_GetHorizontalParityError(TSB_ESIO_TypeDef * ESIOx,  
                             ESIO_LINEx Line);
```

引数:

**ESIOx:** ESIO チャンネルを選択します。

**Line:** 以下よりいずれかの受信ラインを選択します。

- **ESIO\_LINE0:** 受信ライン 0
- **ESIO\_LINE1:** 受信ライン 1
- **ESIO\_LINE2:** 受信ライン 2
- **ESIO\_LINE3:** 受信ライン 3

機能:

受信ラインの水平パリティエラーフラグを読み出します。

戻り値:

受信ラインの水平パリティエラーフラグ

## 8.2.3.23 ESIO\_ClrHorizontalParityError

受信ラインの水平パリティエラーフラグのクリア

関数のプロトタイプ宣言:

```
void  
ESIO_ClrHorizontalParityError(TSB_ESIO_TypeDef * ESIOx,  
                              ESIO_LINEx Line);
```

引数:

**ESIOx:** ESIO チャンネルを選択します。

**Line:** 以下よりいずれかの受信ラインを選択します。

- **ESIO\_LINE0:** 受信ライン 0
- **ESIO\_LINE1:** 受信ライン 1
- **ESIO\_LINE2:** 受信ライン 2
- **ESIO\_LINE3:** 受信ライン 3

**機能:**

受信ラインの水平パリティエラーフラグをクリアします。

**戻り値:**

なし

## 8.2.3.24 ESIO\_GetVerticalParityErrFrame

垂直パリティエラーフレーム状態の読み出し

**関数のプロトタイプ宣言:**

ESIO\_VerticalParityErrFrame

ESIO\_GetVerticalParityErrFrame(TSB\_ESIO\_TypeDef \* ESIOx);

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

垂直パリティエラーフレーム状態を読み出します。

**戻り値:**

垂直パリティエラーの状態

## 8.2.3.25 ESIO\_ClrAllVerticalParityErrNum

垂直パリティエラーフラグのクリア

**関数のプロトタイプ宣言:**

void

ESIO\_ClrAllVerticalParityErrNum(TSB\_ESIO\_TypeDef \* ESIOx);

**引数:**

**ESIOx:** ESIO チャンネルを選択します。

**機能:**

垂直パリティエラーフラグをクリアします。

**戻り値:**

なし

## 8.2.4 データ構造

### 8.2.4.1 ESIO\_InitTypeDef

メンバ:

uint8\_t

**DataDirection**: 以下のいずれかの送信方向を選択します。

- **ESIO\_LSB\_FIRST**: LSB ファースト
- **ESIO\_MSB\_FIRST**: MSB ファースト

UInt8\_t

**FrameLength**: 8～32 のフレーム長を設定します。

UInt8\_t

**CycleBetweenFrames** バースト時フレーム間インターバルサイクルを 0～15 の間で設定します。

UInt8\_t

**NumberOfLINE**: ライン幅を設定します。

- **0**: 1 ライン
- **1**: 2 ライン
- **2**: 3 ライン
- **3**: 4 ライン

UInt8\_t

**ClockPolarity**: 以下のいずれかのシリアルクロックの極性を選択します。

- **ESIO\_CS\_ACTIVE\_LOW**: Low レベル
- **ESIO\_CS\_ACTIVE\_HIGH**: High レベル

uint8\_t

**ShortestIdleCycle**: 最少アイドル時間を 1～15 の間で設定します。

uint8\_t

**CS1ActiveLevel**: ESIOxCS1 の論理を選択します。

- **ESIO\_CS\_ACTIVE\_LOW**: 負論理
- **ESIO\_CS\_ACTIVE\_HIGH**: 正論理

uint8\_t

**CS0ActiveLevel**: ESIOxCS0 の論理を選択します。

- **ESIO\_CS\_ACTIVE\_LOW**: 負論理
- **ESIO\_CS\_ACTIVE\_HIGH**: 正論理

uint8\_t



**DelayCycleNum\_CS\_SCLK:** シリアルクロック遅延設定(CS →SCK)を 1～16 の間で設定します。

uint8\_t

**DelayCycleNum\_Negate\_CS:** CS ネゲート遅延設定(SCK→CS)を 1～16 の間で設定します。

FunctionalState

**HorizontalParityCheck:** 以下よりバースト転送時の水平パリティ機能の許可/禁止を選択します。シングル転送時は禁止を選択してください。

- **ENABLE:** 許可
- **DISABLE:** 禁止

uint8\_t

**HorizontalParity:** 以下よりバースト転送時の水平パリティモードを選択します。

- **ESIO\_PARITY\_ODD:** 奇数パリティ
- **ESIO\_PARITY\_EVEN:** 偶数パリティ

FunctionalState

**VerticalParityCheck:** 以下より垂直パリティ機能の許可/禁止を選択します。

- **ENABLE:** 許可
- **DISABLE:** 禁止

uint8\_t

**VerticalParity:** 以下より垂直パリティモードを選択します。

- **ESIO\_PARITY\_ODD:** 奇数パリティ
- **ESIO\_PARITY\_EVEN:** 偶数パリティ

## 8.2.4.2 ESIO\_StatusFlag

メンバ:

uint32\_t

**All:** ESIO ステータスフラグ

**Bit**

uint32\_t

**RxFIFOFillLevel:** 4                      受信 FIFO fill レベル

uint32\_t

**RxFIFOFull:** 1                      受信 FIFO フルフラグ

uint32\_t

**RxFIFO\_INT:** 1                      受信 FIFO 割り込みフラグ

uint32\_t

**RxCompleted:** 1                      受信完了フラグ

uint32\_t

<b>RxRUN</b> : 1 uint32_t	受信動作中フラグ
<b>Reserved1</b> : 8 uint32_t	Reserved
<b>TxFIFOFillLevel</b> : 4 uint32_t	送信 FIFO fill レベル
<b>TxFIFOEmpty</b> : 1 uint32_t	送信 FIFO エンプティフラグ
<b>TxFIFO_INT</b> : 1 uint32_t	送信 FIFO 割り込みフラグ
<b>TxCompleted</b> : 1 uint32_t	送信完了フラグ
<b>TxRUN</b> : 1 uint32_t	送信動作中フラグ
<b>Reserved2</b> : 7 uint32_t	Reserved
<b>ESIO_RegUsing</b> : 1	ESIO 設定可能状態フラグ

## 8.2.4.3 ESIO\_ParityErrNum

メンバ:

uint32\_t

**All**: ESIO パリティエラーフラグ

**Bit**

uint32_t	
<b>LINE0_ParityErrNum</b> : 2 uint32_t	受信ライン 0 垂直パリティエラーフラグ
<b>LINE1_ParityErrNum</b> : 2 uint32_t	受信ライン 1 垂直パリティエラーフラグ
<b>LINE2_ParityErrNum</b> : 2 uint32_t	受信ライン 2 垂直パリティエラーフラグ
<b>LINE3_ParityErrNum</b> : 2 uint32_t	受信ライン 3 垂直パリティエラーフラグ
<b>HorizontalParityErr</b> : 1 uint32_t	水平パリティエラーフラグ
<b>VerticalParityErr</b> : 1 uint32_t	垂直パリティエラーフラグ
<b>Reserved1</b> : 22	Reserved

## 8.2.4.4 ESIO\_VerticalParityErrFrame

メンバ:

uint32\_t

**All:** ESIO 垂直パリティエラーフレーム

**Bit**

uint32\_t

**LINE0\_VericalParityErrFrame:** 8 受信ライン 0 垂直パリティエラー番号保持ビット

uint32\_t

**LINE1\_VericalParityErrFrame:** 8 受信ライン 1 垂直パリティエラー番号保持ビット

uint32\_t

**LINE2\_VericalParityErrFrame:** 8 受信ライン 2 垂直パリティエラー番号保持ビット

uint32\_t

**LINE3\_VericalParityErrFrame:** 8 受信ライン 3 垂直パリティエラー番号保持ビット

## 9. EXB

### 9.1 概要

本デバイスは、外部にメモリや I/Oなどを接続するための外部バスインターフェース機能を内蔵しています。外部バスインターフェース回路 (EBIF)、チップセレクト(CS)ウェイトコントローラがこれに相当します。

チップセレクト、ウェイトコントローラは、任意の4 ブロックアドレス空間のマッピングアドレス指定と、この4 ブロックアドレス空間に対して、ウェイトおよびデータバス幅(8 ビットまたは16 ビット)を制御します。

外部バスインターフェース回路(EBIF)は、CS/内蔵ウェイトコントローラの設定にもとづき外部バスのタイミングを制御します。

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/TX04_Periph_Driver/src/tmpm440_exb.c,  
/Libraries/TX04_Periph_Driver/inc/tmpm440_exb.h
```

### 9.2 API 関数

#### 9.2.1 関数一覧

- ◆ void EXB\_SetBusMode(uint8\_t **BusMode**);
- ◆ void EXB\_SetBusCycleExtension(uint8\_t **Cycle**);
- ◆ void EXB\_SetEndianType(uint8\_t **ChipSelect**, EXB\_EndianType **EndianType**);
- ◆ void EXB\_Enable(uint8\_t **ChipSelect**);
- ◆ void EXB\_Disable(uint8\_t **ChipSelect**);
- ◆ void EXB\_Init(uint8\_t **ChipSelect**, EXB\_InitTypeDef\* **InitStruct**);

#### 9.2.2 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) EXB バスモード、バスサイクルウェイト拡張、データバス幅、チップセクタを元にした外部バスサイクルの設定:  
EXB\_SetBusMode(), EXB\_SetBusCycleExtension(), EXB\_Init()
- 2) 許可/禁止制御:  
EXB\_Enable(), EXB\_Disable().

- 3) 外部メモリまたは周辺 I/O(ASIC 等)のエンディアン制御:  
EXB\_SetEndianType()

## 9.2.3 関数仕様

### 9.2.3.1 EXB\_SetBusMode

EXB 外部バスモードの設定

関数のプロトタイプ宣言:

void

EXB\_SetBusMode(uint8\_t **BusMode**)

引数:

**BusMode**: 以下から EXB 外部バスモードを選択します。

- EXB\_BUS\_MULTIPLEX: マルチプレクスバスモード

機能:

外部バスモードを設定します。**BusMode** に **BusMode** に EXB\_BUS\_MULTIPLEX を設定した場合、バスモードはマルチプレクスモードになります。

戻り値:

なし

### 9.2.3.2 EXB\_SetBusCycleExtension

バスサイクルウェイト拡張の設定

関数のプロトタイプ宣言:

void

EXB\_SetBusCycleExtension(uint8\_t **Cycle**)

引数:

**Cycle**: バスサイクルウェイト拡張を指定します。

- EXB\_CYCLE\_NONE: 拡張なし
- EXB\_CYCLE\_DOUBLE: 2 倍
- EXB\_CYCLE\_QUADRUPLE: 4 倍

機能:

バスサイクルのセットアップ、ウェイト、リカバリサイクル機能を 2 倍、4 倍に設定します。

戻り値:

なし

## 9.2.3.3 EXB\_SetEndianType

外部メモリまたは周辺 I/O(ASIC 等)のエンディアン制御

関数のプロトタイプ宣言:

void

EXB\_SetEndianType(uint8\_t **ChipSelect**, EXB\_EndianType **EndianType**)

引数:

**ChipSelect**: 以下からチップセレクトを選択します。

➤ **EXB\_CS0**: CS0

➤ **EXB\_CS1**: CS1

**EndianType**: 以下からエンディアンタイプを選択します。

➤ **SAME\_ENDIAN\_AS\_CPU**: CPU と同じエンディアン

➤ **NOT\_SAME\_ENDIAN\_AS\_CPU**: CPU と異なるエンディアン

機能:

外部メモリまたは周辺 I/O(ASIC 等)のエンディアンを設定します。

戻り値:

なし

## 9.2.3.4 EXB\_Enable

チップセレクトの許可。

関数のプロトタイプ宣言:

void

EXB\_Enable(uint8\_t **ChipSelect**)

引数:

**ChipSelect**: チップセレクトを選択します。

➤ **EXB\_CS0**: CS0

➤ **EXB\_CS1**: CS1

機能:

チップセレクトを許可します。

戻り値:

なし

## 9.2.3.5 EXB\_Disable

チップセレクトの禁止

関数のプロトタイプ宣言:

void

EXB\_Disable(uint8\_t **ChipSelect**)

引数:

**ChipSelect**: チップセレクトを選択します。

➤ EXB\_CS0: CS0

➤ EXB\_CS1: CS1

機能:

チップセレクトを禁止します。

戻り値:

なし

## 9.2.3.6 EXB\_Init

チップセレクト設定の初期化

関数のプロトタイプ宣言:

void

EXB\_Init (uint8\_t **ChipSelect**,  
EXB\_InitTypeDef\* **InitStruct**)

引数:

**ChipSelect**: チップセレクトを選択します。

➤ EXB\_CS0: CS0

➤ EXB\_CS1: CS1

**InitStruct**: チップセレクト空間サイズ、スタートアドレス、データバス幅、外部バスサイクルを設定する構造体です。(詳細は、“データ構造”を参照してください)

機能:

チップセレクト設定を初期化します。

戻り値:

なし

## 9.2.4 データ構造

### 9.2.4.1 EXB\_InitTypeDef

メンバ:

uint8\_t

**AddrSpaceSize**: アドレス空間を設定します。

- **EXB\_16M\_BYTE**: アドレス空間 16Mbyte
- **EXB\_8M\_BYTE**: アドレス空間 8Mbyte
- **EXB\_4M\_BYTE**: アドレス空間 4Mbyte
- **EXB\_2M\_BYTE**: アドレス空間 2Mbyte
- **EXB\_1M\_BYTE**: アドレス空間 1Mbyte
- **EXB\_512K\_BYTE**: アドレス空間 512Kbyte
- **EXB\_256K\_BYTE**: アドレス空間 256Kbyte
- **EXB\_128K\_BYTE**: アドレス空間 128Kbyte
- **EXB\_64K\_BYTE**: アドレス空間 64Kbyte

uint8\_t

**StartAddr**: 開始アドレスを設定します。最大値は 0x1FF です。

uint8\_t

**BusWidth**: データバス幅を設定します。

- **EXB\_BUS\_WIDTH\_BIT\_8**: データバス幅 8bit,
- **EXB\_BUS\_WIDTH\_BIT\_16**: データバス幅 16bit.

EXB\_CyclesTypeDef

**Cycles**: 外部バス周期を設定します。

**InternalWait**, **ReadSetupCycle**, **WriteSetupCycle**, **ALEWaitCycle** (マルチプレクスバスマードのみ), **ReadRecoveryCycle**, **WriteRecoveryCycle**, **ChipSelectRecoveryCycle**. (詳細は “EXB\_CyclesTypeDef” を参照)

### 9.2.4.2 EXB\_CyclesType Def

メンバ:

uint8\_t

**InternalWait**: 内部ウェイト(自動挿入)を設定します。

- **EXB\_INTERNAL\_WAIT\_0**: 0 wait
- **EXB\_INTERNAL\_WAIT\_1**: 1 wait
- **EXB\_INTERNAL\_WAIT\_2**: 2 wait
- **EXB\_INTERNAL\_WAIT\_3**: 3 wait
- **EXB\_INTERNAL\_WAIT\_4**: 4 wait
- **EXB\_INTERNAL\_WAIT\_5**: 5 wait



- **EXB\_INTERNAL\_WAIT\_6:** 6 wait
- **EXB\_INTERNAL\_WAIT\_7:** 7 wait
- **EXB\_INTERNAL\_WAIT\_8:** 8 wait
- **EXB\_INTERNAL\_WAIT\_9:** 9 wait
- **EXB\_INTERNAL\_WAIT\_10:** 10 wait
- **EXB\_INTERNAL\_WAIT\_11:** 11 wait
- **EXB\_INTERNAL\_WAIT\_12:** 12 wait
- **EXB\_INTERNAL\_WAIT\_13:** 13 wait
- **EXB\_INTERNAL\_WAIT\_14:** 14 wait
- **EXB\_INTERNAL\_WAIT\_15:** 15 wait

uint8\_t

**ReadSetupCycle** : リード(RDn)セットアップサイクルを設定します。

- **EXB\_CYCLE\_0:** 0 cycle
- **EXB\_CYCLE\_1:** 1 cycle
- **EXB\_CYCLE\_2:** 2 cycle
- **EXB\_CYCLE\_4:** 4 cycle

uint8\_t

**WriteSetupCycle** : ライト(WRn)セットアップサイクルを設定します。

- **EXB\_CYCLE\_0:** 0 cycle
- **EXB\_CYCLE\_1:** 1 cycle
- **EXB\_CYCLE\_2:** 2 cycle
- **EXB\_CYCLE\_4:** 4 cycle

uint8\_t

**ALEWaitCycle**: ALE ウェイトサイクル(マルチプレクスバスモード時)を選択します。

- **EXB\_CYCLE\_0:** 0 cycle
- **EXB\_CYCLE\_1:** 1 cycle
- **EXB\_CYCLE\_2:** 2 cycle
- **EXB\_CYCLE\_4:** 4 cycle

uint8\_t

**ReadRecoveryCycle**: リード(RDn)リカバリサイクルを選択します。

- **EXB\_CYCLE\_0:** 0 cycle
- **EXB\_CYCLE\_1:** 1 cycle
- **EXB\_CYCLE\_2:** 2 cycle
- **EXB\_CYCLE\_3:** 3 cycle
- **EXB\_CYCLE\_4:** 4 cycle
- **EXB\_CYCLE\_5:** 5 cycle
- **EXB\_CYCLE\_6:** 6 cycle
- **EXB\_CYCLE\_8:** 8 cycle

uint8\_t

**WriteRecoveryCycle:** ライト(WRn)リカバリサイクルを選択します。

- **EXB\_CYCLE\_0:** 0 cycle
- **EXB\_CYCLE\_1:** 1 cycle
- **EXB\_CYCLE\_2:** 2 cycle
- **EXB\_CYCLE\_3:** 3 cycle
- **EXB\_CYCLE\_4:** 4 cycle
- **EXB\_CYCLE\_5:** 5 cycle
- **EXB\_CYCLE\_6:** 6 cycle
- **EXB\_CYCLE\_8:** 8 cycle

uint8\_t

**ChipSelectRecoveryCycle:** チップセレクト(CSxn)リカバリサイクルを選択します。

- **EXB\_CYCLE\_0:** 0 cycle
- **EXB\_CYCLE\_1:** 1 cycle
- **EXB\_CYCLE\_2:** 2 cycle
- **EXB\_CYCLE\_4:** 4 cycle

## 10. FC

### 10.1 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、TMPM440F10XBG が 1024Kbyte、TMPM440FEXBG が 768Kbyte 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\\TX04\_Periph\_Driver\\src\\tmpm440\_fc.c

\\Libraries\\TX04\_Periph\_Driver \\inc\\tmpm440\_fc.h

### 10.2 API 関数

#### 10.2.1 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)
- ◆ void FC\_SetCtrlReg(FunctionalState **NewState**)

#### 10.2.2 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) 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().
- 5) フラッシュバッファの制御:  
FC\_SetCtrlReg().

## 10.2.3 関数仕様

### 10.2.3.1 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

### 10.2.3.2 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

### 10.2.3.3 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

### 10.2.3.4 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。

TMPM440F10XBG:

➤ **FC\_BLOCK\_0 ~ FC\_BLOCK\_27**

TMPM440FEXBG:

➤ **FC\_BLOCK\_0 ~ FC\_BLOCK\_23**

**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

### 10.2.3.5 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。

TMPM440F10XBG:

➤ **FC\_BLOCK\_0 ~ FC\_BLOCK\_27**

TMPM440FEXBG:

➤ **FC\_BLOCK\_0 ~ FC\_BLOCK\_23**

**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)

### 10.2.3.6 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\_0**: ブロック 0 ~ 7
- **FC\_BLOCK\_GROUP\_1**: ブロック 8 ~ 13
- **FC\_BLOCK\_GROUP\_2**: ブロック 14 ~ 21
- **FC\_BLOCK\_GROUP\_3**: ブロック 22 ~ 27(TMPM440F10XBG);  
ブロック 22 ~ 23(TMPM440FEXBG)

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)

### 10.2.3.7 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 FC\_PAGE\_SIZE(512Byte)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)

## 10.2.3.8 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)

## 10.2.3.9 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)

## 10.2.3.10 FC\_SetCtrlReg

フラッシュバッファの無効/有効



**関数のプロトタイプ宣言:**

void

FC\_SetCtrlReg(FunctionalState **NewState**)

**引数:**

**NewState:** 以下からフラッシュバッファの無効/有効を選択します。

- **DISABLE:** 無効(同時にバッファをクリアします)
- **ENABLE:** 有効

**機能:**

書き込みまたは消去を行った後に無効→許可を行い、バッファクリアを行ってください。

**戻り値:**

なし

## 10.2.4 データ構造

なし

## 11. FUART

### 11.1 概要

TPM440 は非同期のシリアルチャネル (Full UART)とモデム制御を内蔵します。  
本製品は 2 チャネルの Full UART(FUART0 と FUART1)を内蔵します。

FUARTドライバAPI は、Full UART チャネルを構成する機能、たとえばボーレート、ビット長、パリティチェック、ストップビット、フロー制御、などの共通パラメータを提供します。また、データの送信/受信、エラーチェックなどのような転送を制御します。

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/TX04\_Periph\_Driver/src/tmpm440\_fuart.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_fuart.h

### 11.2 API 関数

#### 11.2.1 関数一覧

- ◆ void FUART\_Enable(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ void FUART\_Disable(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ uint32\_t FUART\_GetRxData(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ void FUART\_SetTxData(TSB\_FUART\_TypeDef \* **FUARTx**, uint32\_t **Data**)
- ◆ FUART\_Err FUART\_GetErrStatus(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ void FUART\_ClearErrStatus(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ WorkState FUART\_GetBusyState(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ FUART\_StorageStatus FUART\_GetStorageStatus(TSB\_FUART\_TypeDef \* **FUARTx**, UART\_Direction **Direction**)
- ◆ void FUART\_Init(TSB\_FUART\_TypeDef \* **FUARTx**, FUART\_InitTypeDef \* **InitStruct**)
- ◆ void FUART\_EnableFIFO(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ void FUART\_DisableFIFO(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ void FUART\_SetSendBreak(TSB\_FUART\_TypeDef \* **FUARTx**, FunctionalState **NewState**)
- ◆ void FUART\_SetINTFIFOLevel(TSB\_FUART\_TypeDef \* **FUARTx**, uint32\_t **RxLevel**, uint32\_t **TxLevel**)
- ◆ void FUART\_SetINTMask(TSB\_FUART\_TypeDef \* **FUARTx**, uint32\_t **IntMaskSrc**)
- ◆ FUART\_INTStatus FUART\_GetINTMask(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ FUART\_INTStatus FUART\_GetRawINTStatus(TSB\_FUART\_TypeDef \* **FUARTx**)
- ◆ FUART\_INTStatus FUART\_GetMaskedINTStatus(TSB\_FUART\_TypeDef \* **FUARTx**)

- ◆ void FUART\_ClearINT(TSB\_FUART\_TypeDef \* **FUARTx**, FUART\_INTStatus **INTStatus**)
- ◆ void FUART\_SetDMAOnErr(TSB\_FUART\_TypeDef \* **FUARTx**, FunctionalState **NewState**)
- ◆ void FUART\_SetFIFODMA(TSB\_FUART\_TypeDef \* **FUARTx**, FUART\_Direction **Direction**, FunctionalState **NewState**)
- ◆ FUART\_AllModemStatus FUART\_GetModemStatus(TSB\_FUART\_TypeDef \* **FUARTx**)

## 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) Full UART 構成と初期化、共通動作:  
FUART\_Enable(), FUART\_Disable, FUART\_Init(), FUART\_GetRxData(),  
FUART\_SetTxData(), FUART\_GetErrStatus(), FUART\_ClearErrStatus(),  
FUART\_GetBusyState(), FUART\_GetStorageStatus(), FUART\_SetSendBreak()
- 2) FIFO と DMA の設定.  
FUART\_EnableFIFO(), FUART\_DisableFIFO(), FUART\_SetINTFIFOLevel(),  
FUART\_SetFIFODMA, FUART\_SetDMAOnErr()
- 3) 割り込み設定、割り込みステータスとクリア  
FUART\_SetINTMask(), FUART\_GetINTMask(), FUART\_GetRawINTStatus(),  
FUART\_GetMaskedINTStatus, FUART\_ClearINT()
- 4) モデム制御  
FUART\_GetModemStatus()

## 11.2.3 関数仕様

**補足:** 下記の全 APIにおいて、パラメータ “TSB\_FUART\_TypeDef\* **FUARTx**” は、**FUART0** もしくは **FUART1** となります。

### 11.2.3.1 FUART\_Enable

Full UART チャンネルの有効化

**関数のプロトタイプ宣言:**

void  
FUART\_Enable(TSB\_FUART\_TypeDef \* **FUARTx**)

**引数:**

**FUARTx:** Full UART チャンネルを指定します。

**機能:**

Full UART チャンネルを有効にします。

**戻り値:**

なし

## 11.2.3.2 FUART\_Disable

Full UART チャンネルの無効化

**関数のプロトタイプ宣言:**

void

FUART\_Disable(TSB\_FUART\_TypeDef \* **FUARTx**)

**引数:**

**FUARTx**: Full UART チャンネルを指定します。

**機能:**

Full UART チャンネルを無効にします。

**戻り値:**

なし

## 11.2.3.3 FUART\_GetRxData

受信データの取得

**関数のプロトタイプ宣言:**

uint32\_t

FUART\_GetRxData(TSB\_FUART\_TypeDef \* **FUARTx**)

**引数:**

**FUARTx**: Full UART チャンネルを指定します。

**機能:**

受信データを取得します。

本 API は、**FUART\_GetStorageStatus(FUARTx, FUART\_RX)**の戻り値が

**FUART\_STORAGE\_NORMAL** あるいは **FUART\_STORAGE\_FULL** の場合に使用してください。

**戻り値:**

受信データ

## 11.2.3.4 FUART\_SetTxData

送信データの設定

関数のプロトタイプ宣言:

void

FUART\_SetTxData(TSB\_FUART\_TypeDef \* **FUARTx**,  
uint32\_t **Data**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

**Data**: 送信データポインタです。データサイズは 0x00 - 0xFF です。

機能:

送信用にデータを設定し、**FUARTx** で選択された Full UART チャンネル経由で送信を開始します。

戻り値:

なし

## 11.2.3.5 FUART\_GetErrStatus

受信エラーステータスの取得

関数のプロトタイプ宣言:

FUART\_Err

FUART\_GetErrStatus(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

本 API は、データ転送後にエラーステータスを取得します。そのため本 API は、**FUART\_GetRxData(FUARTx)**の後に実行してください。ただし、このリードシーケンスはエラーステータス情報を取得した直後のみ実行可能です。

戻り値:

**FUART\_NO\_ERR**: エラーはありません

**FUART\_OVERRUN**: オーバーランエラー

**FUART\_PARITY\_ERR**: パリティエラー

**FUART\_FRAMING\_ERR**: フレミングエラー

**FUART\_BREAK\_ERR**: ブレークエラー

**FUART\_ERRS**: 2 つ以上のエラー

## 11.2.3.6 FUART\_ClearErrStatus

受信エラーステータスのクリア

関数のプロトタイプ宣言:

void

FUART\_ClearErrStatus(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

フレーミングエラー、パリティエラー、ブレークエラー、オーバーランエラーの各エラーがクリアされます。

戻り値:

なし

## 11.2.3.7 FUART\_GetBusyState

データ送信状態の取得

関数のプロトタイプ宣言:

WorkState

FUART\_GetBusyState(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

データ送信中であるか停止中であるか状態を取得します。

戻り値:

データ送信状態:

**BUSY**: データ送信中。

**DONE**: データ送信が停止中

## 11.2.3.8 FUART\_GetStorageStatus

送受信 FIFO または送受信保持レジスタの取得

関数のプロトタイプ宣言:

FUART\_StorageStatus  
FUART\_GetStorageStatus(TSB\_FUART\_TypeDef \* **FUARTx**,  
FUART\_Direction **Direction**)

**引数:**

**FUARTx**: Full UART チャンネルを指定します。

**Direction**: 送信または受信のどちらかを選択します。

- **FUART\_RX**: 受信 FIFO または受信保持レジスタ
- **FUART\_TX**: 送信 FIFO または送信保持レジスタ

**機能:**

FIFO が許可されている場合、送受信 FIFO のステータスを取得します。

FIFO が 禁止されている場合、送受信保持レジスタのステータスを取得します。

**戻り値:**

**FUART\_STORAGE\_EMPTY**: FIFO または保持レジスタが empty 状態

**FUART\_STORAGE\_NORMAL**: FIFO または保持レジスタが正常状態

**FUART\_STORAGE\_INVALID**: FIFO または保持レジスタが無効状態

**FUART\_STORAGE\_FULL**: FIFO または保持レジスタが full 状態

## 11.2.3.9 FUART\_Init

Full UART チャンネルの設定

**関数のプロトタイプ宣言:**

void

FUART\_Init(TSB\_FUART\_TypeDef \* **FUARTx**,  
FUART\_InitTypeDef \* **InitStruct**)

**引数:**

**FUARTx**: Full UART チャンネルを指定します。

**InitStruct**: ボーレート、ワード長、ストップビット、パリティ、転送モード、フロー制御の設定値を格納します。(詳細は“データ構造説明”を参照)

**機能:**

ボーレート、ワード長、ストップビット、パリティ、転送モード、フロー制御の設定を行います。

Full UART 回路を有効にする前に本 API を実行してください。

**戻り値:**

なし

## 11.2.3.10 FUART\_EnableFIFO

送受信 FIFO の有効化

関数のプロトタイプ宣言:

void

FUART\_EnableFIFO(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

送受信 FIFO を許可します。

戻り値:

なし

## 11.2.3.11 FUART\_DisableFIFO

送受信 FIFO の無効化

関数のプロトタイプ宣言:

FUART\_DisableFIFO(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

送受信 FIFO を禁止し、モードをキャラクタモードに変更します。

戻り値:

なし

## 11.2.3.12 FUART\_SetSendBreak

ブレーク付き送信の選択

関数のプロトタイプ宣言:

void

FUART\_SetSendBreak(TSB\_FUART\_TypeDef \* **FUARTx**,  
FunctionalState **NewState**)

引数:



**FUARTx:** Full UART チャンネルを指定します。

**NewState:** ブレーク付き送信の許可/禁止を選択します。

- **ENABLE:** ブレーク送信する。
- **DISABLE:** ブレーク送信しない。

**機能:**

ブレーク付き送信の許可/禁止を選択します。ブレーク状態を生成するには、最低 1 つ以上のフレームを送信中に、本 API にてイネーブルにしてください。ブレーク状態が生成された場合でも、送信 FIFO には影響を与えません。

**戻り値:**

なし

### 11.2.3.13 FUART\_SetINTFIFOLevel

送受信割り込み FIFO レベルの選択

**関数のプロトタイプ宣言:**

void

```
FUART_SetINTFIFOLevel(TSB_FUART_TypeDef * FUARTx,  
                      uint32_t RxLevel,  
                      uint32_t TxLevel)
```

**引数:**

**FUARTx:** Full UART チャンネルを指定します。

**RxLevel:** 受信割り込み FIFO レベルを選択します。(受信 FIFO は 32 段です)

- **FUART\_RX\_FIFO\_LEVEL\_4:** 受信 FIFO が 4 バイト以上
- **FUART\_RX\_FIFO\_LEVEL\_8:** 受信 FIFO が 8 バイト以上
- **FUART\_RX\_FIFO\_LEVEL\_16:** 受信 FIFO が 16 バイト以上
- **FUART\_RX\_FIFO\_LEVEL\_24:** 受信 FIFO が 24 バイト以上
- **FUART\_RX\_FIFO\_LEVEL\_28:** 受信 FIFO が 28 バイト以上

**TxLevel:** 送信割り込み FIFO レベルを選択します。(送信 FIFO は 32 段です)

- **FUART\_TX\_FIFO\_LEVEL\_4:** 送信 FIFO が 4 バイト以上
- **FUART\_TX\_FIFO\_LEVEL\_8:** 送信 FIFO が 8 バイト以上
- **FUART\_TX\_FIFO\_LEVEL\_16:** 送信 FIFO が 16 バイト以上
- **FUART\_TX\_FIFO\_LEVEL\_24:** 送信 FIFO が 24 バイト以上
- **FUART\_TX\_FIFO\_LEVEL\_28:** 送信 FIFO が 28 バイト以上

**機能:**

UARTTXINTR および UARTRXINTR が発生する FIFO レベルを定義します。このレベルを超えると割り込みが発生します。

**戻り値:**

なし

## 11.2.3.14 FUART\_SetINTMask

割り込み発生要因の設定

**関数のプロトタイプ宣言:**

void

```
FUART_SetINTMask(TSB_FUART_TypeDef * FUARTx,  
                 uint32_t IntMaskSrc)
```

**引数:**

**FUARTx**: Full UART チャンネルを指定します。

**IntMaskSrc**: 割り込み発生要因を選択します。

- **FUART\_NONE\_INT\_MASK**: すべての割り込みを禁止します。
- **FUART\_RIN\_MODEM\_INT\_MASK**: RIN モデム割り込みを許可します。
- **FUART\_CTS\_MODEM\_INT\_MASK**: CTS モデム割り込みを許可します。
- **FUART\_DCD\_MODEM\_INT\_MASK**: DCD モデム割り込みを許可します。
- **FUART\_DSR\_MODEM\_INT\_MASK**: DSR モデム割り込みを許可します。
- **FUART\_RX\_FIFO\_INT\_MASK**: 受信割り込みを許可します。
- **FUART\_TX\_FIFO\_INT\_MASK**: 送信割り込みを許可します。
- **FUART\_RX\_TIMEOUT\_INT\_MASK**: 受信タイムアウト割り込みを許可します。
- **FUART\_FRAMING\_ERR\_INT\_MASK**: フレーミングエラー割り込みを許可します。
- **FUART\_PARITY\_ERR\_INT\_MASK**: パリティエラー割り込みを許可します。
- **FUART\_BREAK\_ERR\_INT\_MASK**: ブレークエラー割り込みを許可します。
- **FUART\_OVERRUN\_ERR\_INT\_MASK**: オーバーランエラー割り込みを許可します。
- **FUART\_ALL\_INT\_MASK**: すべての割り込みを許可します。

**機能:**

要因毎に割り込み発生時の許可/禁止を設定します。選択されていない要因の割り込みは禁止されます。

**戻り値:**

なし

## 11.2.3.15 FUART\_GetINTMask

割り込み発生要因の取得

**関数のプロトタイプ宣言:**

```
FUART_INTStatus
```

FUART\_GetINTMask(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

割り込み発生許可/禁止状態を要因毎に取得します。

戻り値:

**FUART\_INTStatus**: 割り込み発生要因が格納された変数です。  
(詳細は“データ構成説明”を参照)

### 11.2.3.16 FUART\_GetRawINTStatus

割り込み許可/禁止設定前の割り込みステータスの取得

関数のプロトタイプ宣言:

FUART\_INTStatus

FUART\_GetRawINTStatus(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

割り込み許可/禁止設定前の割り込みステータスを取得します。

戻り値:

**FUART\_INTStatus**: 割り込みステータスが格納された変数です。(詳細は“データ構成説明”を参照)

### 11.2.3.17 FUART\_GetMaskedINTStatus

割り込み許可/禁止設定後の割り込みステータスの取得

関数のプロトタイプ宣言:

FUART\_INTStatus

FUART\_GetMaskedINTStatus(TSB\_FUART\_TypeDef \* **FUARTx**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

機能:

割り込み許可/禁止設定後の割り込みステータスを取得します。

戻り値:

**FUART\_INTStatus**: 割り込みステータスが格納された変数です。(詳細は“データ構成説明”を参照)

## 11.2.3.18 FUART\_ClearINT

割り込み要因のクリア

関数のプロトタイプ宣言:

void

FUART\_ClearINT(TSB\_FUART\_TypeDef \* **FUARTx**,  
FUART\_INTStatus **INTStatus**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

**INTStatus**: クリア対象の割り込み要因を格納してください。(詳細は“データ構成説明”を参照)

機能:

割り込み要因をクリアします。

戻り値:

なし

## 11.2.3.19 FUART\_SetDMAOnErr

DMA オンエラーの許可/禁止選択

関数のプロトタイプ宣言:

void

FUART\_SetDMAOnErr(TSB\_FUART\_TypeDef \* **FUARTx**,  
FunctionalState **NewState**)

引数:

**FUARTx**: Full UART チャンネルを指定します。

**NewState**: DMA オンエラーの許可/禁止を選択します。

- **ENABLE**: 許可。
- **DISABLE**: 禁止。

機能:

DMA オンエラーの許可/禁止を設定します。許可が選択されると、データ受信中にエラーが発生したときに DMA 受信要求と UARTxRXDMASREQ と UARTxRXDMABREQ が禁止されます。

戻り値:

なし

## 11.2.3.20 FUART\_SetFIFODMA

送受信 DMA の許可/禁止選択

関数のプロトタイプ宣言:

void

```
FUART_SetFIFODMA(TSB_FUART_TypeDef * FUARTx,  
                  FUART_Direction Direction,  
                  FunctionalState NewState)
```

引数:

**FUARTx**: Full UART チャンネルを指定します。

**Direction**: 送信または受信のどちらかを選択します。

➤ **FUART\_RX**: 受信 FIFO または受信保持レジスタ

➤ **FUART\_TX**: 送信 FIFO または送信保持レジスタ

**NewState**: 送信 DMA または受信 DMA の許可/禁止を選択します。

➤ **ENABLE**: 許可。

➤ **DISABLE**: 禁止。

機能:

送信 DMA または受信 DMA の許可/禁止を選択します。

DMAC を用いた送信/受信 FIFO のデータ転送の場合、バス幅を 8bit に設定してください。

戻り値:

なし

## 11.2.3.21 FUART\_GetModemStatus

モデム状態の取得

関数のプロトタイプ宣言:

FUART\_AllModemStatus

```
FUART_GetModemStatus(TSB_FUART_TypeDef * FUARTx)
```

引数:

**FUARTx**: Full UART チャンネルを指定します。

**機能:**

CTS, DSR, DCD, RIN, DTR, RTS の各モデム状態を取得します。

**戻り値:**

**FUART\_AllModemStatus**: 各モデム状態を格納した変数です。(詳細は“データ構成説明”を参照)

## 11.2.4 データ構造

### 11.2.4.1 FUART\_InitTypeDef

**メンバ:**

uint32\_t

**BaudRate**: ボーレートを設定します。0(bps)は設定できません。また、2950000(bps)より小さい値を設定してください。

uint32\_t

**DataBits**: フレームで送受信されたデータビットの数を設定します。

- **UART\_DATA\_BITS\_5**: 5bit
- **UART\_DATA\_BITS\_6**: 6bit
- **UART\_DATA\_BITS\_7**: 7bit
- **UART\_DATA\_BITS\_8**: 8bit

uint32\_t

**StopBits**: 送信ストップビット長を設定します。

- **UART\_STOP\_BITS\_1**: 1bit
- **UART\_STOP\_BITS\_2**: 2bit

uint32\_t

**Parity**: パリティ状態を設定します。

- **UART\_NO\_PARITY**: パリティの送信およびチェックなし
- **UART\_0\_PARITY**: パリティビットとして"0"を送信または受信
- **UART\_1\_PARITY**: パリティビットとして"1"を送信または受信
- **UART\_EVEN\_PARITY**: パリティビットとして偶数パリティを送信または受信
- **UART\_ODD\_PARITY**: パリティビットとして奇数パリティを送信または受信

uint32\_t

**Mode**: 受信、送信あるいは両方の許可/禁止を設定します。

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

uint32\_t

**FlowCtrl:** ハードウェアフロー制御を設定します。

- **UART\_NONE\_FLOW\_CTRL:** フロー制御なし
- **UART\_CTS\_FLOW\_CTRL:** CTS フロー制御許可
- **UART\_RTS\_FLOW\_CTRL:** RTS フロー制御許可
- **UART\_CTS\_FLOW\_CTRL | UART\_RTS\_FLOW\_CTRL:** CTS/RTS フロー制御許可

## 11.2.4.2 FUART\_INTStatus

メンバ:

uint32\_t

**All:** Full UART 割り込みステータス、または割り込み制御

**Bit**

uint32\_t

**Reserved:** 1 未使用

uint32\_t

**CTS:** 1 CTS モデム割り込み

uint32\_t

**Reserved:** 1 未使用

uint32\_t

**Reserved:** 1 未使用

uint32\_t

**RxFIFO:** 1 受信 FIFO 割り込み

uint32\_t

**TxFIFO:** 1 送信 FIFO 割り込み

uint32\_t

**RxTimeout:** 1 受信タイムアウト割り込み

uint32\_t

**FramingErr:** 1 フレーミングエラー割り込み

uint32\_t

**ParityErr:** 1 パリティエラー割り込み

uint32\_t

**BreakErr:** 1 ブレークエラー割り込み

uint32\_t

**OverrunErr:** 1 オーバーランエラー割り込み

uint32\_t

**Reserved:** 21 未使用

## 11.2.4.3 FUART\_AllModemStatus

メンバ:

**Bit**

uint32\_t

**Reserved:** 30 未使用



## 12. GPIO

### 12.1 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/TX04\_Periph\_Driver/src/ tmpm440\_gpio.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_gpio.h

### 12.2 API 関数

#### 12.2.1 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\_SetOpenDrain2 (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**);
- ◆ void GPIO\_WriteDataByte(GPIO\_Port **GPIO\_x**, uint8\_t **Bit\_x**, uint8\_t **Value**);
- ◆ void GPIO\_ToggleDataByte(GPIO\_Port **GPIO\_x**, uint8\_t **Bit\_x**);

## 12.2.2 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(),  
GPIO\_WriteDataByte(), GPIO\_ToggleDataByte()
- 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()

## 12.2.3 関数仕様

### 12.2.3.1 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W

- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAA** : GPIO port AA
- **GPIO\_PAB** : GPIO port AB
- **GPIO\_PAC** : GPIO port AC
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**機能:**

DATA レジスタを読み込みます。

**戻り値:**

DATA レジスタの値です。

## 12.2.3.2 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R

- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAA** : GPIO port AA
- **GPIO\_PAB** : GPIO port AB
- **GPIO\_PAC** : GPIO port AC
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

### 12.2.3.3 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

#### 12.2.3.4 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

- **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

## 12.2.3.5 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAA** : GPIO port AA
- **GPIO\_PAB** : GPIO port AB

- **GPIO\_PAC** : GPIO port AC
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

### 12.2.3.6 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

## 12.2.3.7 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAA** : GPIO port AA
- **GPIO\_PAB** : GPIO port AB
- **GPIO\_PAC** : GPIO port AC
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

### 12.2.3.8 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U

- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

### 12.2.3.9 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAA** : GPIO port AA
- **GPIO\_PAB** : GPIO port AB
- **GPIO\_PAC** : GPIO port AC
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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、入力許可。

**NewState** が **DISABLE** の時、入力禁止。

**戻り値:**

なし

## 12.2.3.10 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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W

- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAA** : GPIO port AA
- **GPIO\_PAB** : GPIO port AB
- **GPIO\_PAC** : GPIO port AC
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAH** : GPIO port AH
- **GPIO\_PAJ** : GPIO port AJ

**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

### 12.2.3.11 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\_PG**: GPIO port G

**Bit\_x**: GPIO 端子を選択します。

- **GPIO\_BIT\_2**: GPIO pin 2
- **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、プルダウン許可。

**NewState** が **DISABLE** の時、プルダウン禁止。

**戻り値:**

なし

## 12.2.3.12 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\_PE** : GPIO port E
- **GPIO\_PH** : GPIO port H
- **GPIO\_PK** : GPIO port K
- **GPIO\_PM** : GPIO port M
- **GPIO\_PR** : GPIO port R
- **GPIO\_PW** : GPIO port W
- **GPIO\_PAJ** : GPIO port AJ

**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\_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、オープンドレイン許可。

**NewState** が **DISABLE** の時、CMOS 許可。

**戻り値:**

なし

### 12.2.3.13 GPIO\_SetOpenDrain2

CMOS/オープンドレインポートの設定(Open Drain Register 2 使用)

**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\_PH** : GPIO port H

**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\_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、オープンドレイン許可。

**NewState** が **DISABLE** の時、CMOS 許可。

**戻り値:**

なし

## 12.2.3.14 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\_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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V

- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAJ** : GPIO port AJ.

**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

**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

## 12.2.3.15 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\_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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAJ** : GPIO port AJ

**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

**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

## 12.2.3.16 GPIO\_WriteDataByte

ビットの有効化

関数のプロトタイプ宣言:

void

GPIO\_WriteDataByte(GPIO\_Port **GPIO\_x**, uint8\_t **Bit\_x**, uint8\_t **Value**)

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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y
- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAJ** : GPIO port AJ

**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

**Value:** バイトデータを設定します。セットされたビットを有効にします。

**機能:**

GPIO DATA レジスタの指定されたビットを設定します。

**戻り値:**

なし

### 12.2.3.17 GPIO\_ToggleDataByte

ビットの反転

**関数のプロトタイプ宣言:**

void

GPIO\_ToggleDataByte(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\_PJ** : GPIO port J
- **GPIO\_PK** : GPIO port K
- **GPIO\_PL** : GPIO port L
- **GPIO\_PM** : GPIO port M
- **GPIO\_PN** : GPIO port N
- **GPIO\_PP** : GPIO port P
- **GPIO\_PR** : GPIO port R
- **GPIO\_PT** : GPIO port T
- **GPIO\_PU** : GPIO port U
- **GPIO\_PV** : GPIO port V
- **GPIO\_PW** : GPIO port W
- **GPIO\_PY** : GPIO port Y

- **GPIO\_PAD** : GPIO port AD
- **GPIO\_PAE** : GPIO port AE
- **GPIO\_PAF** : GPIO port AF
- **GPIO\_PAG** : GPIO port AG
- **GPIO\_PAJ** : GPIO port AJ

**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 DATA レジスタの指定されたビットを反転します。

**戻り値:**

なし

## 12.2.4 データ構造

### 12.2.4.1 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

**OpenDrain2**    オープンドレインポート/CMOS ポートの設定(Open Drain Register2 使用):

- **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



## 13. KSCAN

### 13.1 概要

KSCAN ドライバ API は、キースキャン入出力、キースキャン動作、キースキャン割り込みなどを設定します。

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/TX04\_Periph\_Driver/src/ tmpm440\_kscan.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_kscan.h

### 13.2 API 関数

#### 13.2.1 関数一覧

- ◆ void KSCAN\_Enable(void)
- ◆ void KSCAN\_Disable(void)
- ◆ void KSCAN\_SetSCLK(uint8\_t ksclk)
- ◆ void KSCAN\_SetInputCtrlMask(uint8\_t InputCH, FunctionalState NewState)
- ◆ void KSCAN\_SetOutputCtrl(uint8\_t OutputCH, FunctionalState NewState)
- ◆ void KSCAN\_SetStrobeOutput(uint8\_t cycle)
- ◆ void KSCAN\_Start(void)
- ◆ void KSCAN\_Stop(void)
- ◆ uint8\_t KSCAN\_ReadStart(void)
- ◆ void KSCAN\_KSBRInitial(void)
- ◆ void KSCAN\_SWReset(void)
- ◆ void KSCAN\_SetConsecutiveMatches(uint8\_t match)
- ◆ void KSCAN\_SetChatCancelTime(uint8\_t timeN)
- ◆ void KSCAN\_SetStrobeWidth(uint8\_t StrobeWidth)
- ◆ uint8\_t KSCAN\_ReadBufChecked(uint8\_t output)
- ◆ void KSCAN\_MaskReadBuf(uint32\_t MaskCH, uint32\_t MaskBit, FunctionalState NewState)
- ◆ void KSCAN\_SetINTReq(FunctionalState NewState)

#### 13.2.2 関数の種類

上記機能は 4 つのペアに分けることができます。

- 1) KSCAN 機能の有効/無効:

KSCAN\_Enable (), KSCAN\_Disable (),KSCAN\_Start(),KSCAN\_Stop(),

- KSCAN\_ReadStart()
- 2) KSCAN 機能を設定:  
KSCAN\_SetSCLK(), KSCAN\_SetInputCtrl(), KSCAN\_SetOutputCtrl(),  
KSCAN\_SetConsecutiveMatches(),KSCAN\_SetChatCancelTime(),  
KSCAN\_SetStrobeWidth(),KSCAN\_ReadBufChecked(),  
KSCAN\_MaskReadBuf().KSCAN\_SetStrobeOutput()
- 3) KSCAN リセット:  
KSCAN\_KSBRInitial(), KSCAN\_SWReset()
- 4) KSCAN 用割り込み  
KSCAN\_SetINTReq ()

## 13.2.3 関数仕様

### 13.2.3.1 KSCAN\_Enable

KSCAN 機能の許可

**関数のプロトタイプ宣言:**

void KSCAN\_Enable(void)

**引数:**

なし

**機能:**

KSCAN 機能を許可します。

**戻り値:**

なし

### 13.2.3.2 KSCAN\_Disable

KSCAN 機能の禁止

**関数のプロトタイプ宣言:**

void KSCAN\_Disable(void)

**引数:**

なし

**機能:**

KSCAN 機能を禁止します。

**戻り値:**

なし

### 13.2.3.3 KSCAN\_SetSCLK

KSCAN 動作クロック(kscclk)の選択

関数のプロトタイプ宣言:

Void:

KSCAN\_SetSCLK(uint8\_t **kscclk**)

引数:

**kscclk**: 以下から KSCAN 動作クロックを選択します。

- **KSCAN\_KSCL\_FS** : fs
- **KSCAN\_KSCL\_TBOUT**: TBxOUT

機能:

KSCAN 動作クロックを選択します。

戻り値:

なし

### 13.2.3.4 KSCAN\_SetInputCtrl

KSCAN 入力制御の設定

関数のプロトタイプ宣言:

Void

KSCAN\_SetInputCtrl (uint8\_t **InputCH**, FunctionalState **NewState**)

引数:

**InputCH**: 以下のいずれのチャンネルを選択します。有効ビットの組み合わせが可能です。

- **KSCAN\_CH\_0**: KSIN0
- **KSCAN\_CH\_1**: KSIN1
- **KSCAN\_CH\_2**: KSIN2
- **KSCAN\_CH\_3**: KSIN3
- **KSCAN\_CH\_4**: KSIN4
- **KSCAN\_CH\_5**: KSIN5
- **KSCAN\_CH\_6**: KSIN6
- **KSCAN\_CH\_7**: KSIN7
- **KSCAN\_CH\_ALL**: 全ての KSIN

**NewState** : KSCAN 入力をマスクする/マスクしないを選択します。

- **ENABLE:** マスクする。
- **DISABLE:** マスクしない。

**機能:**

KSCAN 入力制御を設定します。

**戻り値:**

なし

### 13.2.3.5 KSCAN\_SetOutputCtrl

KSCAN 出力の Low 信号出力設定

**関数のプロトタイプ宣言:**

void

KSCAN\_SetOutputCtrl(uint8\_t **OutputCH**, FunctionalState **NewState**)

**引数:**

**OutputCH:** 以下のいずれかのチャンネルを選択します。有効ビットの組み合わせが可能です。

- **KSCAN\_CH\_0:** KSOUT0
- **KSCAN\_CH\_1:** KSOUT1
- **KSCAN\_CH\_2:** KSOUT2
- **KSCAN\_CH\_3:** KSOUT3
- **KSCAN\_CH\_4:** KSOUT4
- **KSCAN\_CH\_5:** KSOUT5
- **KSCAN\_CH\_6:** KSOUT6
- **KSCAN\_CH\_7:** KSOUT7
- **KSCAN\_CH\_ALL:** すべての KSOUT

**NewState:** Low 信号出力をする/しないを選択します。

- **ENABLE:** Low 信号出力する
- **DISABLE:** Low 信号出力しない

**機能:**

KSCAN 出力の Low 信号出力を設定します。

**戻り値:**

なし

## 13.2.3.6 KSCAN\_SetStrobeOutput

KSCAN ストローブ信号出力の設定

**関数のプロトタイプ宣言:**

void

KSCAN\_SetStrobeOutput(uint8\_t *cycle*)

**引数:**

**cycle:** ストローブ信号の出力サイクルと出力する/しないを選択します。  
データは 1 ～255 で設定します。

**機能:**

KSCAN ストローブ信号出力を設定します。

**戻り値:**

なし

## 13.2.3.7 KSCAN\_Start

キーストローブの出力/カウンタの開始

**関数のプロトタイプ宣言:**

void

KSCAN\_Start(void)

**引数:**

なし

**機能:**

キーストローブの出力/カウンタを開始します。

**戻り値:**

なし

## 13.2.3.8 KSCAN\_Stop

キーストローブの出力/カウンタの停止

**関数のプロトタイプ宣言:**

void

KSCAN\_Stope(void)

**引数:**

なし

**機能:**

キーストロープの出力/カウンタを停止します。

**戻り値:**

なし

## 13.2.3.9 KSCAN\_ReadStart

キースキャン開始状態の読み出し

**関数のプロトタイプ宣言:**

void

KSCAN\_ReadStart(void)

**引数:**

なし

**機能:**

キースキャン開始状態を読み出します。

**戻り値:**

キースキャン開始状態: (0: キースキャン停止中、1: キースキャン動作中)

## 13.2.3.10 KSCAN\_KSBRInitial

バッファレジスタ(KSBR1~0)の初期化

**関数のプロトタイプ宣言:**

void

KSCAN\_KSBRInitial(void)

**引数:**

なし

**機能:**

バッファレジスタ(KSBR1~0)を初期化します。

**戻り値:**

なし

## 13.2.3.11 KSCAN\_SWReset

ソフトウェアリセット

**関数のプロトタイプ宣言:**

void  
KSCAN\_SWReset(void)

**引数:**

なし

**機能:**

KSCAN 機能のリセットを行います。

**戻り値:**

なし

## 13.2.3.12 KSCAN\_SetConsecutiveMatches

バッファレジスタの更新を行うためのキー判定回数選択

**関数のプロトタイプ宣言:**

void  
KSCAN\_SetConsecutiveMatches(uint8\_t *match*)

**引数:**

**Match:** 連続一致回数を選択します。

- **KSCAN\_MATCH\_2C:** 2 回連続一致
- **KSCAN\_MATCH\_4C:** 4 回連続一致

**機能:**

バッファレジスタの更新を行うためのキー判定回数を選択します。

**戻り値:**

なし

## 13.2.3.13 KSCAN\_SetChatCancelTime

チャタリング除去時間の設定

**関数のプロトタイプ宣言:**

void  
KSCAN\_SetChatCancelTime(uint8\_t *timeN*)

**引数:**

**timeN:** チャタリング除去時間を選択します。(n: 0~255)  
チャタリング除去時間は、 $16/k_{sclk} \times n$  で決まります。

**機能:**

チャタリング除去時間を設定します。

**戻り値:**

なし

### 13.2.3.14 KSCAN\_SetStrobeWidth

ストロブ幅の設定

**関数のプロトタイプ宣言:**

void  
KSCAN\_SetStrobeWidth(uint8\_t **StrobeWidth**)

**引数:**

**StrobeWidth:** ストロブ幅を 0~5 の範囲で選択します。  
0:  $4/k_{sclk}$   
1:  $8/k_{sclk}$   
2:  $16/k_{sclk}$   
3:  $32/k_{sclk}$   
4:  $64/k_{sclk}$   
5:  $128/k_{sclk}$

**機能:**

ストロブ幅を設定します。

**戻り値:**

なし

### 13.2.3.15 KSCAN\_ReadBufChecked

バッファレジスタの読み出し

**関数のプロトタイプ宣言:**

void  
KSCAN\_ReadBufChecked(uint8\_t **output**)



**引数:**

**output:** KSCAN 出カラインを 0~7 の間で選択します。

**機能:**

バッファレジスタを読み出します。

**戻り値:**

なし

## 13.2.3.16 KSCAN\_MaskReadBuf

バッファレジスタのマスク

**関数のプロトタイプ宣言:**

void

KSCAN\_MaskReadBuf(uint32\_t **MaskCH**, uint32\_t **MaskBit**, FunctionalState **NewState**)

**引数:**

**MaskCH:** マスク対象の KSCAN 出カラインを 0~7 の間で選択します。

**MaskBit:** **MaskCH** に対する入力カラムのマスク対象のビットを 1~255 の間で設定します。

**NewState:** マスクする/しないを選択します。(DISABLE: マスクしない、ENABLE: マスクする)

**機能:**

バッファレジスタをマスクします。

**戻り値:**

なし

## 13.2.3.17 KSCAN\_SetINTReq

KSCAN 割り込み要求の許可/禁止

**関数のプロトタイプ宣言:**

Void

KSCAN\_SetINTReq(FunctionalState **NewState**)

**引数:**

**NewState :** KSCAN 割り込み要求を行うかどうかを選択します。

- **ENABLE :** 許可(割り込みをマスクしない)
- **DISABLE:** 禁止(割り込みをマスクする)

機能:

KSCAN 割り込み要求を行うかどうかを設定します。

戻り値:

なし

## 13.2.4 データ構造

なし

## 14. KWUP

### 14.1 概要

KWUPドライバ API は、KWUP 入力、KWUP 動作、KWUP 割り込みなどを設定します。

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/TX04\_Periph\_Driver/src/ tmpm440\_kwup.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_kwup.h

### 14.2 API 関数

#### 14.2.1 関数一覧

- ◆ void KWUP\_SetConfig(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**, KWUP\_SettingTypeDef \* **Settings**);
- ◆ KWUP\_PortStatus KWUP\_GetPortStatus(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**);
- ◆ void KWUP\_SetPullUpConfig(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**, KWUP\_PullUpCycles **T1**, KWUP\_PullUpCycles **T2**);
- ◆ KWUP\_INTStatus KWUP\_GetINTStatus(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**);

#### 14.2.2 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) KWUP の設定:

KWUP\_SetConfig (), KWUP\_SetPullUpConfig ()

2) KWUP 入力の状態および割り込み状態の取得:

KWUP\_GetPortStatus () KWUP\_GetINTStatus()

#### 14.2.3 関数仕様

##### 14.2.3.1 KWUP\_SetConfig

KWUP の初期設定

関数のプロトタイプ宣言:

void

KWUP\_SetConfig(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**, KWUP\_SettingTypeDef \* **Settings**);

**引数:**

**TSB\_KWUPx:** 以下から KWUP ユニットを選択します。

➤ **TSB\_KWUPA:** ユニット A

➤ **TSB\_KWUPB:** ユニット B

**Settings:** KWUP の初期化を行う構造体です。詳細は"データ構造"を参照してください。

**機能:**

KWUP の初期設定を行います。

**戻り値:** なし

## 14.2.3.2 KWUP\_GetPortStatus

KWUP 入力の端子状態の取得

**関数のプロトタイプ宣言:**

KWUP\_PortStatus

KWUP\_GetPortStatus(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**);

**引数:**

**TSB\_KWUPx:** 以下から KWUP ユニットを選択します。

➤ **TSB\_KWUPA:** ユニット A

➤ **TSB\_KWUPB:** ユニット B

**機能:**

KWUP 入力端子状態を取得します。

**戻り値:**

なし

## 14.2.3.3 KWUP\_SetPullUpConfig

ダイナミックプルアップ期間とダイナミックプルアップ周期の設定

**関数のプロトタイプ宣言:**

void

KWUP\_SetPullUpConfig(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**,  
KWUP\_PullUpCycles **T1**, KWUP\_PullUpCycles **T2**)

**引数:**

**TSB\_KWUPx:** 以下から KWUP ユニットを選択します。

➤ **TSB\_KWUPA:** ユニット A

➤ **TSB\_KWUPB**: ユニット B

**T1**: 以下からダイナミックプルアップ期間を選択します。

➤ **KWUP\_CYCLES\_2\_FS**: 2/fs

➤ **KWUP\_CYCLES\_4\_FS**: 4/fs

➤ **KWUP\_CYCLES\_8\_FS**: 8/fs

➤ **KWUP\_CYCLES\_16\_FS**: 16/fs

**T2**: 以下からダイナミックプルアップ周期を選択します。

➤ **KWUP\_CYCLES\_256\_FS**: 256/fs,

➤ **KWUP\_CYCLES\_512\_FS**: 512/fs,

➤ **KWUP\_CYCLES\_1024\_FS**: 1024/fs,

➤ **KWUP\_CYCLES\_2048\_FS**: 2048/fs,

**機能:**

ダイナミックプルアップ期間とダイナミックプルアップ周期を設定します。

**戻り値:**

なし

#### 14.2.3.4 KWUP\_GetINTStatus

KWUP 割り込み発生時に検出された KWUP 入力状態の取得

**関数のプロトタイプ宣言:**

KWUP\_INTStatus

KWUP\_GetINTStatus(TSB\_KWUP\_TypeDef \* **TSB\_KWUPx**);

**引数:**

**TSB\_KWUPx**: 以下から KWUP ユニットを選択します。

➤ **TSB\_KWUPA**: ユニット A

➤ **TSB\_KWUPB**: ユニット B

**機能:**

KWUP 割り込み発生時に検出された KWUP 入力状態を取得します。

**戻り値:**

KWUP 割り込み発生時に検出された KWUP 入力状態: “0” なし、“1” あり

**Key0**(Bit 0): KEYINT0

**Key1**(Bit 1): KEYINT1

**Key2**(Bit 2): KEYINT2

**Key3**(Bit 3): KEYINT3

**Key4**(Bit 4): KEYINT4

**Key5**(Bit 5): KEYINT5

**Key6**(Bit 6): KEYINT6

**Key7**(Bit 7): KEYINT7  
**Key8**(Bit 8): KEYINT8  
**Key9**(Bit 9): KEYINT9  
**Key10**(Bit 10): KEYINT10  
**Key11**(Bit 11): KEYINT11  
**Key12**(Bit 12): KEYINT12  
**Key13**(Bit 13): KEYINT13  
**Key14**(Bit 14): KEYINT14  
**Key15**(Bit 15): KEYINT15  
**Key16**(Bit 16): KEYINT16  
**Key17**(Bit 17): KEYINT17  
**Key18**(Bit 18): KEYINT18  
**Key19**(Bit 19): KEYINT19  
**Key20**(Bit 20): KEYINT20  
**Key21**(Bit 21): KEYINT21  
**Key22**(Bit 22): KEYINT22  
**Key23**(Bit 23): KEYINT23  
**Key24**(Bit 24): KEYINT24  
**Key25**(Bit 25): KEYINT25  
**Key26**(Bit 26): KEYINT26  
**Key27**(Bit 27): KEYINT27  
**Key28**(Bit 28): KEYINT28  
**Key29**(Bit 29): KEYINT29  
**Key30**(Bit 30): KEYINT30  
**Key31**(Bit 31): KEYINT31

## 14.2.4 データ構造

### 14.2.4.1 KWUP\_SettingTypeDef

メンバ:

KWUP\_Input

**KeyN:** KWUP 入力

- **KWUP\_INPUT\_0** :KWUP0
- **KWUP\_INPUT\_1** :KWUP1
- **KWUP\_INPUT\_2** :KWUP2
- **KWUP\_INPUT\_3** :KWUP3
- **KWUP\_INPUT\_4** :KWUP4
- **KWUP\_INPUT\_5** :KWUP5
- **KWUP\_INPUT\_6** :KWUP6
- **KWUP\_INPUT\_7** :KWUP7
- **KWUP\_INPUT\_8** :KWUP8

- KWUP\_INPUT\_9 :KWUP9
- KWUP\_INPUT\_10 :KWUP10
- KWUP\_INPUT\_11 :KWUP11
- KWUP\_INPUT\_12 :KWUP12
- KWUP\_INPUT\_13 :KWUP13
- KWUP\_INPUT\_14 :KWUP14
- KWUP\_INPUT\_15 :KWUP15
- KWUP\_INPUT\_16 :KWUP16
- KWUP\_INPUT\_17 :KWUP17
- KWUP\_INPUT\_18 :KWUP18
- KWUP\_INPUT\_19 :KWUP19
- KWUP\_INPUT\_20 :KWUP20
- KWUP\_INPUT\_21 :KWUP21
- KWUP\_INPUT\_22 :KWUP22
- KWUP\_INPUT\_23 :KWUP23
- KWUP\_INPUT\_24 :KWUP24
- KWUP\_INPUT\_25 :KWUP25
- KWUP\_INPUT\_26 :KWUP26
- KWUP\_INPUT\_27 :KWUP27
- KWUP\_INPUT\_28 :KWUP28
- KWUP\_INPUT\_29 :KWUP29
- KWUP\_INPUT\_30 :KWUP30
- KWUP\_INPUT\_31 :KWUP31

KWUP\_PullUpCtrl

**PullUpCtrl:** 以下からスタティックプルアップ、またはダイナミックプルアップを選択します。

- KWUP\_PUP\_CTRL\_BY\_STATIC : スタティックプルアップ
- KWUP\_PUP\_CTRL\_BY\_DYNAMIC : ダイナミックプルアップ

KWUP\_ActiveState

**ActiveState:** 以下から KWUP 入力を検出するアクティブ状態を選択します。

- KWUP\_ACTIVE\_BY\_L\_LEVEL : “Low”レベル
- KWUP\_ACTIVE\_BY\_H\_LEVEL : “High”レベル
- KWUP\_ACTIVE\_BY\_RISING\_EDGE : 立ち上がりエッジ
- KWUP\_ACTIVE\_BY\_FALLING\_EDGE: 立下りエッジ
- KWUP\_ACTIVE\_BY\_BOTH\_EDGES: 両エッジ

FunctionalState

**INTNewState:** 以下から KWUP 割り込み要求を選択します。

- DISABLE: 禁止
- ENABLE: 許可

## 15. PHC

### 15.1 概要

PHCドライバAPIは、PHCチャネルの設定、モードの設定、ノイズフィルタの設定、割り込み要求の設定、ステータスリード、カウンタリードなどが含まれます。

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/TX04\_Periph\_Driver/src/tmpm440\_phc.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_phc.h

### 15.2 API 関数

#### 15.2.1 関数一覧

- ◆ void PHC\_Enable(TSB\_PHC\_TypeDef \* **PHCx**);
- ◆ void PHC\_Disable(TSB\_PHC\_TypeDef \* **PHCx**);
- ◆ void PHC\_SetRunState(TSB\_PHC\_TypeDef \* **PHCx**, uint32\_t **Cmd**);
- ◆ void PHC\_Init(TSB\_PHC\_TypeDef \* **PHCx**, PHC\_InitTypeDef \* **InitStruct**);
- ◆ PHC\_INTFactor PHC\_GetINTFactor(TSB\_PHC\_TypeDef \* **PHCx**);
- ◆ void PHC\_ClearINTFactor(TSB\_PHC\_TypeDef \* **PHCx**, uint32\_t **ClearINT**);
- ◆ void PHC\_EnableInterrupt(TSB\_PHC\_TypeDef \* **PHCx**, uint32\_t **EnableINT**);
- ◆ void PHC\_DisableInterrupt(TSB\_PHC\_TypeDef \* **PHCx**, uint32\_t **DisableINT**);
- ◆ uint16\_t PHC\_GetPulseCntValue(TSB\_PHC\_TypeDef \* **PHCx**);
- ◆ void PHC\_ClearPulseCntValue(TSB\_PHC\_TypeDef \* **PHCx**);
- ◆ uint16\_t PHC\_GetCompareValue(TSB\_PHC\_TypeDef \* **PHCx**, uint8\_t **CmpReg**);
- ◆ void PHC\_SetCompareValue(TSB\_PHC\_TypeDef \* **PHCx**, uint8\_t **CmpReg**, uint16\_t **CmpValue**);
- ◆ void PHC\_SetDMAReq(TSB\_PHC\_TypeDef \* **PHCx**, FunctionalState **NewState**, uint8\_t **DMAReq**);

#### 15.2.2 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) 各 PHC チャネルの機能構成と制御:  
PHC\_Enable (), PHC\_Disable (), PHC\_Init(), PHC\_SetRunState()
- 2) 各 PHC チャネルのステータスリード:  
PHC\_GetINTFactor(), PHC\_GetPulseCntValue(), PHC\_GetCompareValue()



3) その他:

PHC\_ClearINTFactor(), PHC\_EnableInterrupt(), PHC\_DisableInterrupt(),  
PHC\_ClearPulseCntValue(), PHC\_SetCompareValue(), PHC\_SetDMAReq()

## 15.2.3 関数仕様

**補足:** 下記すべての API において、パラメーター “TSB\_PHC\_TypeDef \* **PHCx**” は、次のいずれかの値となります。

TSB\_PCH0, TSB\_PCH1

### 15.2.3.1 PHC\_Enable

PHCNT 動作の許可

**関数のプロトタイプ宣言:**

void

PHC\_Enable(TSB\_PHC\_TypeDef\* **PHCx**)

**引数:**

**PHCx:** PHCNT チャンネルを選択します。

**機能:**

PHCNT 動作を許可します。

**戻り値:**

なし

### 15.2.3.2 PHC\_Disable

PHCNT 動作の禁止

**関数のプロトタイプ宣言:**

void

PHC\_Disable(TSB\_PHC\_TypeDef\* **PHCx**)

**引数:**

**PHCx:** PHCNT チャンネルを選択します。

**機能:**

PHCNT 動作を禁止します。

**戻り値:**

なし

## 15.2.3.3 PHC\_SetRunState

アップダウンカウンタ PHCNT のカウント動作制御

関数のプロトタイプ宣言:

```
void  
PHC_SetRunState(TSB_PHC_TypeDef * PHCx,  
                uint32_t Cmd);
```

引数:

**PHCx**: PHCNT チャンネルを選択します。

**Cmd**: 以下からカウント動作を選択します。

- **PHC\_RUN**: 動作
- **PHC\_STOP**: 停止&クリア

機能:

**Cmd** が **PHC\_RUN** の場合、アップダウンカウンタのカウント動作を開始します。**Cmd** が **PHC\_STOP** の場合、アップダウンカウンタのカウント動作を停止し、内部カウンタをクリアします。

戻り値:

なし

## 15.2.3.4 PHC\_Init

PHCCNT の初期化

関数のプロトタイプ宣言:

```
void  
PHC_Init(TSB_PHC_TypeDef * PHCx,  
         PHC_InitTypeDef * InitStruct);
```

引数:

**PHCx**: PHCNT チャンネルを選択します。

**InitStruct**: カウントモード、ノイズフィルター制御、カウンタークリアなどの PHCNT 基本構成を含む構造体です。(詳細は“データ構成説明”を参照)

機能:

PHCNT を初期化します。

戻り値:

なし

## 15.2.3.5 PHC\_GetINTFactor

割り込み要因の取得

**関数のプロトタイプ宣言:**

PHC\_INTFactor

PHC\_GetINTFactor(TSB\_PHC\_TypeDef\* *PHCx*)

**引数:**

*PHCx*: PHCNT チャンネルを選択します。

**機能:**

PHCNT 割り込み要因を取得します。

**戻り値:**

PHCNT の割り込み要因: 以下は各ビットの意味です。

**Compare0** (Bit0): コンペア 0 一致割り込み

**Compare1** (Bit1): コンペア 1 一致割り込み

**Overflow** (Bit2): カウンタオーバーフロー

**Underflow** (Bit3): カウンタアンダーフロー

## 15.2.3.6 PHC\_ClearINTFactor

割り込み要因のクリア

**関数のプロトタイプ宣言:**

void

PHC\_ClearINTFactor(TSB\_PHC\_TypeDef \* *PHCx*,  
uint32\_t *ClearINT*)

**引数:**

*PHCx*: PHCNT チャンネルを選択します。

*ClearINT*: 以下からクリアする PHCNT 割り込み要因を選択します。有効ビットの組み合わせが可能です。

- **PHC\_FLG\_CMP0**: コンペア 0 一致割り込み
- **PHC\_FLG\_CMP1**: コンペア 1 一致割り込み
- **PHC\_FLG\_OVERFLOW**: カウンタオーバーフロー
- **PHC\_FLG\_UNDERFLOW**: カウンタアンダーフロー
- **PHC\_FLG\_ALL**: すべての割り込み要因

**機能:**

PHCNT 割り込みをクリアします。

戻り値:

なし

補足:

各割り込み要因は自動的にクリアされません。使用前に初期化してください。

## 15.2.3.7 PHC\_EnableInterrupt

PHCNT 割り込みの許可

関数のプロトタイプ宣言:

void

```
PHC_EnableInterrupt(TSB_PHC_TypeDef * PHCx,  
                    uint32_t EnableINT);
```

引数:

**PHCx**: PHCNT チャンネルを選択します。

**EnableINT**: 許可する割り込み要因を選択します。本引数は組み合わせ指定が可能です。

- **PHC\_CR\_INT\_COMP0**: INTPHTx0 割り込み (x: 0 ~ 3)
- **PHC\_CR\_INT\_COMP1**: INTPHTx1 割り込み (x: 0 ~ 3)
- **PHC\_CR\_INT\_COMP0\_AND\_1**: INTPHTx0 および INTPHTx1 の両割り込み(x: 0 ~ 3)
- **PHC\_CR\_INT\_EVERY**: INTPHEVERYx 割り込み (x: 0 ~ 3)
- **PHC\_CR\_INT\_ALL**: 上記全ての割り込み

機能:

PHCNT 割り込みを許可します。

戻り値:

なし

## 15.2.3.8 PHC\_DisableInterrupt

PHCNT 割り込みの禁止

関数のプロトタイプ宣言:

void

```
PHC_DisableInterrupt(TSB_PHC_TypeDef * PHCx,  
                     uint32_t DisableINT);
```

引数:

**PHCx**: PHCNT チャンネルを選択します。

**DisableINT:** 禁止する割り込み要因を選択します。本引数は組み合わせ指定が可能です。

- **PHC\_CR\_INT\_COMP0:** INTPHTx0 割り込み (x: 0 ~ 3)
- **PHC\_CR\_INT\_COMP1:** INTPHTx1 割り込み (x: 0 ~ 3)
- **PHC\_CR\_INT\_COMP0\_AND\_1:** INTPHTx0 および INTPHTx1 の両割り込み(x: 0 ~ 3)
- **PHC\_CR\_INT\_EVERY:** INTPHEVRYx 割り込み (x: 0 ~ 3)
- **PHC\_CR\_INT\_ALL:** 上記全ての割り込み

**機能:**

PHCNT 割り込みを禁止します。

**戻り値:**

なし

### 15.2.3.9 PHC\_GetPulseCntValue

アップダウンカウンタ値の読み出し

**関数のプロトタイプ宣言:**

uint16\_t

PHC\_GetPulseCntValue(TSB\_PHC\_TypeDef \* **PHCx**);

**引数:**

**PHCx:** PHCNT チャンネルを選択します。

**機能:**

アップダウンカウンタ値を読み出します。

**戻り値:**

アップダウンカウンタ値

**補足:**

PHCxCNT は MCU の動作クロックと非同期でカウントアップダウンします。そのため読み出すタイミングによっては正しい値を読み出すことができません。PHCxCNT を読み出すには、PHCxCNT を 2 回読み出し、読み出した値が一致するか確認するか、INTPHEVRYx 割り込みサービスルーチンの中で、次のカウントアップダウンまでに PHCxCNT を読み出してください。

### 15.2.3.10 PHC\_ClearPulseCntValue

アップダウンカウンタ値のクリア

**関数のプロトタイプ宣言:**

void

PHC\_ClearPulseCntValue(TSB\_PHC\_TypeDef \* **PHCx**);

**引数:**

**PHCx**: PHCNT チャンネルを選択します。

**機能:**

アップタウンカウンタ値をクリアします。

**戻り値:**

なし

**補足:**

本 API をコールするとカウンタ値は **0x7FFF** となります。

## 15.2.3.11 PHC\_GetCompareValue

コンペア値の取得

**関数のプロトタイプ宣言:**

uint16\_t

PHC\_GetCompareValue(TSB\_PHC\_TypeDef \* **PHCx**,  
uint8\_t **CmpReg**)

**引数:**

**PHCx**: PHCNT チャンネルを選択します。

**CmpReg**: コンペアレジスタを選択します。

- **PHC\_COMP\_0**: コンペアレジスタ 0
- **PHC\_COMP\_1**: コンペアレジスタ 1

**機能:**

**CmpReg** が **PHC\_COMP\_0** の場合、コンペアレジスタ 0 の値を取得します。

**CmpReg** が **PHC\_COMP\_1** の場合、コンペアレジスタ 1 の値を取得します。

**戻り値:**

コンペア値

## 15.2.3.12 PHC\_SetCompareValue

コンペア値の設定

## 関数のプロトタイプ宣言:

```
void  
PHC_SetCompareValue(TSB_PHC_TypeDef * PHCx,  
                    uint8_t CmpReg,  
                    uint16_t CmpValue);
```

## 引数:

**PHCx**: PHCNT チャンネルを選択します。

**CmpReg**: コンペアレジスタを選択します。

➤ **PHC\_COMP\_0**: コンペアレジスタ 0

➤ **PHC\_COMP\_1**: コンペアレジスタ 1

**CmpValue**: コンペアレジスタへ設定する値です。

## 機能:

**CmpReg** が **PHC\_COMP\_0** の場合、コンペアレジスタ 0 へ値を設定します。

**CmpReg** が **PHC\_COMP\_1** の場合、コンペアレジスタ 1 へ値を設定します。

## 戻り値:

なし

### 15.2.3.13 PHC\_SetDMAReq

DMA 要求の許可/禁止

## 関数のプロトタイプ宣言:

```
void  
PHC_SetDMAReq(TSB_PHC_TypeDef * PHCx,  
              FunctionalState NewState,  
              uint8_t DMAReq);
```

## 引数:

**PHCx**: PHCNT チャンネルを選択します。

**NewState**: DMA 要求の許可/禁止を選択します。

➤ **ENABLE**: 許可

➤ **DISABLE**: 禁止

**DMAReq**: 以下を設定します。

➤ **PHC\_DMA\_REQ\_CAPTURE\_2**

## 機能:

DMA 要求の許可/禁止を選択します。

## 戻り値:

なし

補足:

割り込み要求を禁止している場合、DMA 要求を許可しても DMA 要求は発生されません。

## 15.2.4 データ構造

### 15.2.4.1 PHC\_InitTypeDef

メンバ:

uint32\_t

**Mode:** 動作モードを選択します。以下より設定します。

- **PHC\_CR\_MODE\_NORMAL:** 通常モード
- **PHC\_CR\_MODE\_4TIMES:** 4 通倍モード
- **PHC\_CR\_MODE\_2TIMES\_IN0:** 2 通倍モード(PHCxIN0)
- **PHC\_CR\_MODE\_2TIMES\_IN1:** 2 通倍モード(PHCxIN1)

uint32\_t

**NoiseFilterCtrl:** ノイズフィルタの許可/禁止を選択します。

- **PHC\_CR\_NOISEFILTER\_ON:** ON
- **PHC\_CR\_NOISEFILTER\_OFF:** OFF

uint32\_t

**CountClearCtrl:** カウンタ値を初期値にする/しないを選択します。

- **PHC\_COUNT\_CONTINUE:** Don't care
- **PHC\_COUNT\_CLR:** クリア

### 15.2.4.2 PHC\_INTFactor

メンバ:

uint32\_t

**All:** PHCxCNT 割り込み要因

**Bit**

uint32\_t

**Compare0:** 1      コンペア 0 一致割り込み

uint32\_t

**Compare1:** 1      コンペア 1 一致割り込み

uint32\_t

**Overflow:** 1      カウンタオーバーフロー

uint32\_t

**UnderFlow:** 1      カウンタアンダーフロー

uint32\_t



*Reserverd* : 28 未使用

## 16. RTC

### 16.1 概要

本 RTC ドライバは、年、うるう年、月、日、曜日、時間、分、秒、時間モードなどを格納する RTC クロック、アラームの設定を行う関数セットです。

本ドライバ 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/TX04\_Periph\_Driver/src/tmpm440\_rtc.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_rtc.h

### 16.2 API 関数

#### 16.2.1 関数一覧

- ◆ void RTC\_SetSec(uint8\_t **Sec**);
- ◆ uint8\_t RTC\_GetSec(void);
- ◆ void RTC\_SetMin(RTC\_FuncMode **NewMode**, uint8\_t **Min**);
- ◆ uint8\_t RTC\_GetMin(RTC\_FuncMode **NewMode**);
- ◆ uint8\_t RTC\_GetAMPM(RTC\_FuncMode **NewMode**);
- ◆ void RTC\_SetHour24(RTC\_FuncMode **NewMode**, uint8\_t **Hour**);
- ◆ void RTC\_SetHour12(RTC\_FuncMode **NewMode**, uint8\_t **Hour**, uint8\_t **AmPm**);
- ◆ uint8\_t RTC\_GetHour(RTC\_FuncMode **NewMode**);
- ◆ void RTC\_SetDay(RTC\_FuncMode **NewMode**, uint8\_t **Day**);
- ◆ uint8\_t RTC\_GetDay(RTC\_FuncMode **NewMode**);
- ◆ void RTC\_SetDate(RTC\_FuncMode **NewMode**, uint8\_t **Date**);
- ◆ uint8\_t RTC\_GetDate(RTC\_FuncMode **NewMode**);
- ◆ void RTC\_SetMonth(uint8\_t **Month**);
- ◆ uint8\_t RTC\_GetMonth(void);
- ◆ void RTC\_SetYear(uint8\_t **Year**);
- ◆ uint8\_t RTC\_GetYear(void);
- ◆ void RTC\_SetHourMode(uint8\_t **HourMode**);
- ◆ uint8\_t RTC\_GetHourMode(void);
- ◆ void RTC\_SetLeapYear(uint8\_t **LeapYear**);
- ◆ uint8\_t RTC\_GetLeapYear(void);
- ◆ void RTC\_SetTimeAdjustReq(void);
- ◆ RTC\_ReqState RTC\_GetTimeAdjustReq(void);
- ◆ void RTC\_EnableClock(void);
- ◆ void RTC\_DisableClock(void);
- ◆ void RTC\_EnableAlarm(void);

- ◆ void RTC\_DisableAlarm(void);
- ◆ void RTC\_SetRTCINT(FunctionalState **NewState**);
- ◆ void RTC\_SetAlarmOutput(uint8\_t **Output**);
- ◆ void RTC\_ResetClockSec(void);
- ◆ RTC\_ReqState RTC\_GetResetClockSecReq(void);
- ◆ void RTC\_ResetAlarm(void);
- ◆ void RTC\_SetDateValue(RTC\_DateTypeDef \* **DateStruct**);
- ◆ void RTC\_GetDateValue(RTC\_DateTypeDef \* **DateStruct**);
- ◆ void RTC\_SetTimeValue(RTC\_TimeTypeDef \* **TimeStruct**);
- ◆ void RTC\_GetTimeValue(RTC\_TimeTypeDef \* **TimeStruct**);
- ◆ void RTC\_SetClockValue(RTC\_DateTypeDef \* **DateStruct**, RTC\_TimeTypeDef \* **TimeStruct**);
- ◆ void RTC\_GetClockValue(RTC\_DateTypeDef \* **DateStruct**, RTC\_TimeTypeDef \* **TimeStruct**);
- ◆ void RTC\_SetAlarmValue(RTC\_AlarmTypeDef \* **AlarmStruct**);
- ◆ void RTC\_GetAlarmValue(RTC\_AlarmTypeDef \* **AlarmStruct**);

## 16.2.2 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) RTC 機能の年月日の設定:  
RTC\_SetDay(), RTC\_GetDay(), RTC\_SetDate(), RTC\_GetDate(), RTC\_SetMonth(),  
RTC\_GetMonth(), RTC\_SetYear(), RTC\_GetYear(), RTC\_SetLeapYear(),  
RTC\_GetLeapYear(), RTC\_SetDateValue(), RTC\_GetDateValue()
- 2) RTC 機能の時間の設定:  
RTC\_SetSec(), RTC\_GetSec(), RTC\_SetMin(), RTC\_GetMin(), RTC\_SetHour24(),  
RTC\_SetHour12(), RTC\_GetHour(), RTC\_SetHourMode(), RTC\_GetHourMode,  
RTC\_GetAMPM(), RTC\_SetTimeValue(), RTC\_GetTimeValue()
- 3) RTC(clock)の設定:  
RTC\_EnableClock(), RTC\_DisableClock(), RTC\_SetTimeAdjustReq(),  
RTC\_GetTimeAdjustReq(), RTC\_ResetClockSec(), RTC\_GetResetClockSec(),  
RTC\_SetClockValue(), RTC\_GetClockValue()
- 4) RTC(alarm)の設定:  
RTC\_EnableAlarm(), RTC\_DisableAlarm(), RTC\_ResetAlarm(), RTC\_SetAlarmValue(),  
RTC\_GetAlarmValue()
- 5) その他:  
RTC\_SetAlarmOutput(), RTC\_SetRTCINT()

## 16.2.3 関数仕様

### 16.2.3.1 RTC\_SetSec

時計の秒桁設定

**関数のプロトタイプ宣言:**

```
void  
RTC_SetSec(uint8_t Sec);
```

**引数:**

**Sec**:最大 59 までの秒桁設定の値。

**機能:**

時計の秒桁値を設定します。RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数の呼び出し後、RTC1Hz 割り込みを待つ必要があります。

**戻り値:**

なし

## 16.2.3.2 RTC\_GetSec

時計の秒桁設定

**関数のプロトタイプ宣言:**

```
uint8_t  
RTC_GetSec(void);
```

**引数:**

なし。

**機能:**

時計の秒桁の値を返します。

**戻り値:**

時計の秒桁:  
0 ~ 59

## 16.2.3.3 RTC\_SetMin

時計/アラームの分桁設定

**関数のプロトタイプ宣言:**

```
void  
RTC_SetMin(RTC_FuncMode NewMode,  
            uint8_t Min);
```

**引数:**

**NewMode:** RTC モードを選択します。

- **RTC\_CLOCK\_MODE:** 時計機能
- **RTC\_ALARM\_MODE:** アラーム機能

**Min:** 最大 59 までの分析を設定します。

**機能:**

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計の分析を設定します。

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラームの分析を設定します。

RTC レジスタは、INTRTC のタイミングに同期して書き換えられます。この関数を呼び出した後に、1HZ 割り込みが発生するのを待つ必要があります。

**戻り値:**

なし

## 16.2.3.4 RTC\_GetMin

時計/アラームの分析読み込み

**関数のプロトタイプ宣言:**

```
uint8_t  
RTC_GetMin(RTC_FuncMode NewMode);
```

**引数:**

**NewMode:** RTC モードを選択します。

- **RTC\_CLOCK\_MODE:** 時計機能
- **RTC\_ALARM\_MODE:** アラーム機能

**機能:**

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計の分析の値を返します。

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラームの分析の値を返します。

**戻り値:**

分析:

0 ~ 59

## 16.2.3.5 RTC\_GetAMPM

12 時間モードの AM/PM 読み込み

**関数のプロトタイプ宣言:**

```
uint8_t  
RTC_GetAMPM(RTC_FuncMode NewMode);
```

**引数:**

**NewMode:** RTC モードを選択します。

- **RTC\_CLOCK\_MODE:** 時計機能
- **RTC\_ALARM\_MODE:** アラーム機能

**機能:**

時計/アラームの AM/PM を返します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計の AM/PM を返します。

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラームの AM/PM を返します。

**戻り値:**

時計モード:

**RTC\_AM\_MODE:** AM

**RTC\_PM\_MODE:** PM

## 16.2.3.6 RTC\_SetHour24

24 時間モードの時計/アラーム時桁設定。

**関数のプロトタイプ宣言:**

void

```
RTC_SetHour24(RTC_FuncMode NewMode,  
              uint8_t Hour);
```

**引数:**

**NewMode:** RTC モードを選択します。

- **RTC\_CLOCK\_MODE:** 時計機能
- **RTC\_ALARM\_MODE:** アラーム機能

**Hour:** 最大 23 までの時桁を設定します。

**機能:**

24 時間モードの時計/アラームの時桁を設定します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計機能の時桁を設定し、

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラームの時桁を設定します。

RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数の実行後、1Hz 割り込みが発生するのを待つ必要があります。

\*12 時間モードから 24 時間モードに変更する場合、本関数 **RTC\_SetHour24()** によって HOURR レジスタを再設定してください。

**戻り値:**

なし

## 16.2.3.7 RTC\_SetHour12

12 時間モードの時計/アラーム時桁設定

関数のプロトタイプ宣言:

void

```
RTC_SetHour12(RTC_FuncMode NewMode,  
              uint8_t Hour,  
              uint8_t AmPm);
```

引数:

**NewMode**: RTC モードを選択します。

- **RTC\_CLOCK\_MODE**: 時計機能
- **RTC\_ALARM\_MODE**: アラーム機能

**Hour**: 最大 11 までの時桁を設定します。

**AmPm**: 以下から時間モードを選択します。

- **RTC\_AM\_MODE**: 12H モードの AM モード
- **RTC\_PM\_MODE**: 12H モードの PM モード

機能:

12 時間モードの時計/アラームの時桁を設定します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計機能の時桁を設定し、

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラーム機能の時桁を設定します。

RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数の実行後、1Hz 割り込みが発生するのを待つ必要があります。

\*24 時間モードから 12 時間モードに変更する場合、本関数 **RTC\_SetHour12()** によって HOURS レジスタを再度設定してください。

戻り値:

なし

## 16.2.3.8 RTC\_GetHour

時計/アラームの時桁読み込み。

関数のプロトタイプ宣言:

uint8\_t

```
RTC_GetHour(RTC_FuncMode NewMode);
```

引数:

**NewMode**: RTC モードを選択します。

- **RTC\_CLOCK\_MODE**: 時計機能

➤ **RTC\_ALARM\_MODE:** アラーム機能

**機能:**

時計/アラームの時桁を返します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計機能の時桁の値を返し、

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラーム機能の時桁の値を返します。

**戻り値:**

24 時間モードでの時桁:

0 ~ 23

12H 時間モードでの時桁:

0 ~ 11

## 16.2.3.9 RTC\_SetDay

時計/アラームの曜日設定

**関数のプロトタイプ宣言:**

void

```
RTC_SetDay(RTC_FuncMode NewMode,  
           uint8_t Day);
```

**引数:**

**NewMode:** RTC モードを選択します。

- **RTC\_CLOCK\_MODE:** 時計機能
- **RTC\_ALARM\_MODE:** アラーム機能

**Day:** 曜日を選択します。

- **RTC\_SUN:** 日曜日
- **RTC\_MON:** 月曜日
- **RTC\_TUE:** 火曜日
- **RTC\_WED:** 水曜日
- **RTC\_THU:** 木曜日
- **RTC\_FRI:** 金曜日
- **RTC\_SAT:** 土曜日

**機能:**

時計/アラームの曜日を設定します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計機能の曜日を設定します。

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラーム機能の曜日を設定します。

RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数の実行後、1Hz 割り込みが発生するのを待つ必要があります。

**戻り値:**



なし

## 16.2.3.10 RTC\_GetDay

時計/アラームの曜日の読み込み

関数のプロトタイプ宣言:

```
uint8_t  
RTC_GetDay(RTC_FuncMode NewMode);
```

引数:

**NewMode**: RTC モードを選択します。

- **RTC\_CLOCK\_MODE**: 時計機能
- **RTC\_ALARM\_MODE**: アラーム機能

機能:

時計/アラームの曜日を返します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計機能の曜日を返し、  
**NewMode** が **RTC\_ALARM\_MODE** の場合、アラーム機能の曜日を返します。

戻り値:

曜日の値:

0 ~ 6

## 16.2.3.11 RTC\_SetDate

時計/アラームの日桁設定

関数のプロトタイプ宣言:

```
void  
RTC_SetDate(RTC_FuncMode NewMode,  
             uint8_t Date);
```

引数:

**NewMode**: RTC モードを選択します。

- **RTC\_CLOCK\_MODE**: 時計機能
- **RTC\_ALARM\_MODE**: アラーム機能

**Date**: 1 から 31 の日桁を設定します。

機能:

時計/アラームの日桁を設定します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合は、時計機能の日桁を設定し、  
**NewMode** が **RTC\_ALARM\_MODE** の場合は、アラーム機能の日桁を設定します。  
RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数を呼び出した後に、1Hz 割り込みが発生するのを待つ必要があります。

戻り値:

なし

## 16.2.3.12 RTC\_GetDate

時計/アラームの日桁読み込み

関数のプロトタイプ宣言:

uint8\_t

RTC\_GetDate(RTC\_FuncMode **NewMode**);

引数:

**NewMode**: RTC モードを選択します。

- **RTC\_CLOCK\_MODE**: 時計機能
- **RTC\_ALARM\_MODE**: アラーム機能

機能:

時計/アラームの日桁を返します。

**NewMode** が **RTC\_CLOCK\_MODE** の場合、時計機能の日桁の値を返し、

**NewMode** が **RTC\_ALARM\_MODE** の場合、アラーム機能の日桁の値を返します。

戻り値:

日桁:

1 ~ 31

## 16.2.3.13 RTC\_SetMonth

時計の月桁設定

関数のプロトタイプ宣言:

void

RTC\_SetMonth(uint8\_t **Month**);

引数:

**Month**: 1 から 12 の月桁を設定します。

機能:

時計の月桁を設定します。

RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数の実行後、1Hz 割り込みが発生するのを待つ必要があります。

**戻り値:**

なし

## 16.2.3.14 RTC\_GetMonth

時計の月桁読み込み

**関数のプロトタイプ宣言:**

```
uint8_t  
RTC_GetMonth(void);
```

**引数:**

なし。

**機能:**

時計の月桁の値を返します。

**戻り値:**

月桁:

1 ~ 12

## 16.2.3.15 RTC\_SetYear

時計の年桁設定

**関数のプロトタイプ宣言:**

```
void  
RTC_SetYear(uint8_t Year);
```

**引数:**

**Year:** 最大 99 までの年の値

**機能:**

時計の年桁を設定します。

RTC レジスタは、INTRTC のタイミングに同期して書換えられます。この関数の実行後、1Hz 割り込みが発生するのを待つ必要があります。

**戻り値:**

なし

## 16.2.3.16 RTC\_GetYear

時計の年桁の読み込み

関数のプロトタイプ宣言:

```
uint8_t  
RTC_GetYear(void);
```

引数:

なし。

機能:

時計の年桁の値を返します。

戻り値:

年桁:

0 ~ 99

## 16.2.3.17 RTC\_SetHourMode

24 時間時計/12 時間時計の選択

関数のプロトタイプ宣言:

```
void  
RTC_SetHourMode(uint8_t HourMode);
```

引数:

*HourMode*: 時間モードを選択します。

- **RTC\_12\_HOUR\_MODE**: 12 時間時計
- **RTC\_24\_HOUR\_MODE**: 24 時間時計

機能:

24 時間時計/12 時間時計を選択します。

*HourMode* が **RTC\_24\_HOUR\_MODE** の時、12 時間時計を選択し、

*HourMode* が **RTC\_12\_HOUR\_MODE** の時、24 時間時計を選択します。

補足:

本APIを実行する前に **RTC\_DisableClock()** を実行し、時計を停止してください。

(詳細は**RTC\_DisableClock()**を参照してください)

戻り値:

なし

## 16.2.3.18 RTC\_GetHourMode

時計モードの読み込み

関数のプロトタイプ宣言:

uint8\_t

RTC\_GetHourMode(void);

引数:

なし。

機能:

時計モードを読み込みます。

戻り値:

時計モード

RTC\_24\_HOUR\_MODE: 24 時間時計

RTC\_12\_HOUR\_MODE: 12 時間時計

## 16.2.3.19 RTC\_SetLeapYear

うるう年の設定

関数のプロトタイプ宣言:

void

RTC\_SetLeapYear(uint8\_t *LeapYear*);

引数:

*LeapYear*: 以下からうるう年を選択します。

- RTC\_LEAP\_YEAR\_0: 現在の年(今年)がうるう年
- RTC\_LEAP\_YEAR\_1: 現在がうるう年から 1 年目
- RTC\_LEAP\_YEAR\_2: 現在がうるう年から 2 年目
- RTC\_LEAP\_YEAR\_3: 現在がうるう年から 3 年目

機能:

うるう年を設定します。

*LeapYear* が RTC\_LEAP\_YEAR\_0 の場合、現在の年(今年)がうるう年で、

*LeapYear* が RTC\_LEAP\_YEAR\_1 の場合、現在がうるう年から 1 年目で、

*LeapYear* が RTC\_LEAP\_YEAR\_2 の場合、現在がうるう年から 2 年目で、

*LeapYear* が `RTC_LEAP_YEAR_3` の場合、現在がうるう年から 3 年目になります。

戻り値:

なし

## 16.2.3.20 RTC\_GetLeapYear

うるう年の読み込み

関数のプロトタイプ宣言:

`uint8_t`

`RTC_GetLeapYear(void);`

引数:

なし。

機能:

うるう年の状態を返します。

戻り値:

うるう年の状態を表す値

## 16.2.3.21 RTC\_SetTimeAdjustReq

+/- 30 秒の補正

関数のプロトタイプ宣言:

`void`

`RTC_SetTimeAdjustReq(void);`

引数:

なし。

機能:

秒の補正をします。要求は秒カウンタのカウントアップ時にサンプリングされ、秒が 0~29 秒の場合、秒桁のみ "0" になります。また、30~59 秒のときは分を桁上げて秒を"0"にします。

戻り値:

なし

## 16.2.3.22 RTC\_GetTimeAdjustReq

ADJUST 要求状態の読み込み

**関数のプロトタイプ宣言:**

RTC\_ReqState

RTC\_GetTimeAdjustReq(void);

**引数:**

なし。

**機能:**

ADJUST 要求状態を読み込みます。**RTC\_SetTimeAdjustReq()** の実行後に、この関数を実行し、繰り返して要求をしないようにします。

**戻り値:**

ADJUST 要求状態:

**RTC\_NO\_REQ** : ADJUST 要求なし

**RTC\_REQ**: ADJUST 要求あり

## 16.2.3.23 RTC\_EnableClock

時計機能の許可

**関数のプロトタイプ宣言:**

void

RTC\_EnableClock(void);

**引数:**

なし。

**機能:**

時計機能を有効にします。

**戻り値:**

なし

## 16.2.3.24 RTC\_DisableClock

時計機能の禁止

**関数のプロトタイプ宣言:**

void

RTC\_DisableClock(void);

**引数:**

なし。

**機能:**

時計機能を禁止にします。

**戻り値:**

なし

## 16.2.3.25 RTC\_EnableAlarm

アラーム機能の許可

**関数のプロトタイプ宣言:**

void

RTC\_EnableAlarm(void);

**引数:**

なし。

**機能:**

アラーム機能を許可します。

**戻り値:**

なし

## 16.2.3.26 RTC\_DisableAlarm

アラーム機能の禁止

**関数のプロトタイプ宣言:**

void

RTC\_DisableAlarm(void);

**引数:**

なし。

**機能:**

アラーム機能を禁止します。



戻り値:  
なし

## 16.2.3.27 RTC\_SetRTCINT

INTRTC 割り込みの許可/禁止

関数のプロトタイプ宣言:

```
void  
RTC_SetRTCINT(FunctionalState NewState);
```

引数:

**NewState**: 以下から *INTRTC* の有効/無効を選択します。

- **ENABLE**: *INTRTC* 割り込み有効
- **DISABLE**: *INTRTC* 割り込み無効

機能:

**NewState** が **ENABLE** の場合、*RTCINT* を有効にし、**NewState** が **DISABLE** の場合、*RTCINT* を無効にします。

戻り値:  
なし

## 16.2.3.28 RTC\_SetAlarmOutput

ALARM 端子の出力設定

関数のプロトタイプ宣言:

```
void  
RTC_SetAlarmOutput(uint8_t Output);
```

引数:

**Output**: 以下から、アラーム端子の出力を選択します。

- **RTC\_LOW\_LEVEL**: “0” パルス
- **RTC\_PULSE\_1\_HZ**: 1Hz 周期の “0” パルス
- **RTC\_PULSE\_16\_HZ**: 16Hz 周期の “0” パルス
- **RTC\_PULSE\_2\_HZ**: 2Hz 周期の “0” パルス
- **RTC\_PULSE\_4\_HZ**: 4Hz 周期の “0” パルス
- **RTC\_PULSE\_8\_HZ**: 8Hz 周期の “0” パルス

機能:

アラーム端子の出力を設定します。

**Output** が **RTC\_LOW\_LEVEL** の場合、時計に同期してアラーム端子の出力は “0” になり、**Output** が **RTC\_PULSE\_n\*\_HZ** の場合、アラーム端子の出力は  $n \times \text{Hz}$  周期の “0” パルスになります。(n\* は次のいずれかの値: 1, 2, 4, 8, 16)

**戻り値:**

なし

### 16.2.3.29 RTC\_ResetClockSec

時計秒カウンタのリセット

**関数のプロトタイプ宣言:**

void

RTC\_ResetClockSec(void);

**引数:**

なし。

**機能:**

時計秒カウンタをリセットします。

**戻り値:**

なし

### 16.2.3.30 RTC\_GetResetClockSecReq

時計秒カウンタのリセット要求状態の読み込み

**関数のプロトタイプ宣言:**

RTC\_ReqState

RTC\_GetResetClockSecReq(void);

**引数:**

なし。

**機能:**

時計秒カウンタのリセット要求状態を読み込みます。リセット要求は、低速クロックを使用してサンプリングします。クロックが安定するために、**RTC\_ResetClockSec()** の実行後に本関数を実行してください。

**戻り値:**

リセット要求状態:

- RTC\_NO\_REQ: リセット要求なし
- RTC\_REQ: リセット要求あり

## 16.2.3.31 RTC\_ResetAlarm

アラームリセット要求

**関数のプロトタイプ宣言:**

```
void  
RTC_ResetAlarm(void);
```

**引数:**

なし

**機能:**

アラームリセットを行います。  
アラームレジスタ(分、時、日、週)を初期化します。  
初期化後は、00 分、00 時、01 日、日曜日になります。

**戻り値:**

なし

## 16.2.3.32 RTC\_SetDateValue

時計の日付設定

**関数のプロトタイプ宣言:**

```
void  
RTC_SetDateValue(RTC_DateTypeDef * DateStruct);
```

**引数:**

**DateStruct:** うるう年、年、月、曜日、日を格納する構造体です。(詳細は「データ構造」を参照してください)

**機能:**

時計の日付(うるう年、年、月、曜日、日)を読み込みます。  
**RTC\_SetLeapYear()**, **RTC\_SetYear()**, **RTC\_SetMonth()**, **RTC\_SetDate()**,  
**RTC\_Setday()**を実行します。

**戻り値:**

なし

## 16.2.3.33 RTC\_GetDateValue

時計の日付の読み込み

**関数のプロトタイプ宣言:**

void

RTC\_GetDateValue(RTC\_DateTypeDef \* **DateStruct**);

**引数:**

**DateStruct:** うるう年、年、月、曜日、日を格納する含む構造体。(詳細は「データ構造」を参照)

**機能:**

時計のうるう年、年、月、曜日、日を読み込みます。

RTC\_GetLeapYear(), RTC\_GetYear(), RTC\_GetMonth(), RTC\_GetDate(),  
RTC\_Getday()を実行します。

**戻り値:**

なし

## 16.2.3.34 RTC\_SetTimeValue

時計の時刻設定

**関数のプロトタイプ宣言:**

void

RTC\_SetTimeValue(RTC\_TimeTypeDef \* **TimeStruct**);

**引数:**

**TimeStruct:** 時間モード、時間、12 時間モードの AM/PM モード、分、秒を格納する構造体。(詳細は「データ構造」を参照)

**機能:**

時間モード、時間、12 時間モードの AM/PM モード、分、秒を設定します。

RTC\_SetHourMode(), RTC\_SetHour12(), RTC\_SetHour24(), RTC\_SetMin(),  
RTC\_SetSec() を実行します。

**戻り値:**

なし

## 16.2.3.35 RTC\_GetTimeValue

時計の時刻の読み込み

**関数のプロトタイプ宣言:**

void

RTC\_GetTimeValue(RTC\_TimeTypeDef \* **TimeStruct**);

**引数:**

**TimeStruct:** 時間モード、時間、12 時間モードの AM/PM モード、分、秒を格納する構造体。(詳細は「データ構造」を参照)

**機能:**

時刻(時間モード、時間、12 時間モードの AM/PM モード、分、秒)を読み込みます。

**RTC\_GetHourMode()**, **RTC\_GetHour()**, **RTC\_GetAMPM()**, **RTC\_GetMin()**, **RTC\_GetSec()** が実行されます。

**戻り値:**

なし

## 16.2.3.36 RTC\_SetClockValue

時計の日時設定

**関数のプロトタイプ宣言:**

void

RTC\_SetClockValue(RTC\_DateTypeDef \* **DateStruct**,  
RTC\_TimeTypeDef \* **TimeStruct**);

**引数:**

**DateStruct:** うるう年、年、月、曜日、日を格納する構造体。

**TimeStruct:** 時間モード、時間、12 時間モードの AM/PM モード、分、秒を格納する構造体。(詳細は「データ構造」を参照)

**機能:**

時計の日付(うるう年、年、月、曜日、日)、および、時刻(時間モード、時間、12 時間モードの AM/PM モード、分、秒)を設定します。

**RTC\_SetLeapYear()**, **RTC\_SetYear()**, **RTC\_SetMonth()**, **RTC\_SetDate()**,  
**RTC\_SetDay()**, **RTC\_SetHourMode()**, **RTC\_SetHour24()**, **RTC\_SetHour12()**,  
**RTC\_SetMin()**, **RTC\_SetSec()** を実行します。

**戻り値:**

なし

## 16.2.3.37 RTC\_GetClockValue

時計の日時の読み込み

関数のプロトタイプ宣言:

void

```
RTC_GetClockValue(RTC_DateTypeDef * DateStruct,  
                  RTC_TimeTypeDef * TimeStruct);
```

引数:

**DateStruct**: うるう年、年、月、曜日、日を格納する構造体。

**TimeStruct**: 時間モード、時間、12 時間モードの AM/PM モード、分、秒を格納する構造体。(詳細は「データ構造」を参照)

機能:

時計の日付(うるう年、年、月、曜日、日)、および、時刻(時間モード、時間、12 時間モードの AM/PM モード、分、秒)を設定します。

**RTC\_GetLeapYear()**, **RTC\_GetYear()**, **RTC\_GetMonth()**, **RTC\_GetDate()**,  
**RTC\_GetDay()**, **RTC\_GetHourMode()**, **RTC\_GetHour()**, **RTC\_GetAMPM()**,  
**RTC\_GetMin()**, **RTC\_GetSec()** を実行します。

戻り値:

なし

## 16.2.3.38 RTC\_SetAlarmValue

アラームの日時設定

関数のプロトタイプ宣言:

void

```
RTC_SetAlarmValue(RTC_AlarmTypeDef * AlarmStruct);
```

引数:

**AlarmStruct**: 日、曜日、時間、12 時間モードの AM/PM、秒を格納する構造体。(詳細は「データ構造」を参照してください)

機能:

アラームの日時(日、曜日、時間、12 時間モードの AM/PM モード、秒を含む) を設定します。RTC\_SetDate(), RTC\_SetDay(), RTC\_SetHour12(), RTC\_SetHour24() , RTC\_SetMin() が実行されます。

戻り値:

なし

## 16.2.3.39 RTC\_GetAlarmValue

アラームの日時の読み込み

関数のプロトタイプ宣言:

void

RTC\_GetAlarmValue(RTC\_AlarmTypeDef \* **AlarmStruct**);

引数:

**AlarmStruct**: 日、曜日、時間、12 時間モードの AM/PM、秒を格納する構造体。  
(詳細は「データ構造」を参照)

機能:

アラームの日時(日、曜日、時間、12 時間モードの AM/PM モード、秒を含む) を読み込みます。

RTC\_GetDate(), RTC\_GetDay(), RTC\_GetHour() , RTC\_GetAMPM(),  
RTC\_GetMin() を実行します。

戻り値:

なし

## 16.2.4 データ構造

### 16.2.4.1 RTC\_DateTypeDef

メンバ:

uint8\_t

**LeapYear**: うるう年を設定します:

- **RTC\_LEAP\_YEAR\_0**: 現在の年(今年)がうるう年
- **RTC\_LEAP\_YEAR\_1**: 現在がうるう年から 1 年目
- **RTC\_LEAP\_YEAR\_2**: 現在がうるう年から 2 年目
- **RTC\_LEAP\_YEAR\_3**: 現在がうるう年から 3 年目

uint8\_t

**Year**: 年桁の値(0~99)。

uint8\_t

**Month:** 月桁の値(1～12)。

uint8\_t

**Date:** 日桁の値(1～31)。

uint8\_t

**Day:** 週桁の値を以下。

- **RTC\_SUN:** 日曜日
- **RTC\_MON:** 月曜日
- **RTC\_TUE:** 火曜日
- **RTC\_WED:** 水曜日
- **RTC\_THU:** 木曜日
- **RTC\_FRI:** 金曜日
- **RTC\_SAT:** 土曜日

## 16.2.4.2 RTC\_TimeTypeDef

メンバ:

uint8\_t

**HourMode:** 24 時間時計、12 時間時計のモード選択の値:

- **RTC\_12\_HOUR\_MODE:** 12 時間モード
- **RTC\_24\_HOUR\_MODE:** 24 時間モード

uint8\_t

**Hour:** 時間桁の値。(24 時間モード:0～23、12 時間モード:0～11)

uint8\_t

**AMPM:** 12 時間モード時の AM/PM の値:

- **RTC\_AM\_MODE:** AM モード
- **RTC\_PM\_MODE:** PM モード
- **RTC\_AMPM\_INVALID:** 24 時間モード

uint8\_t

**Min:** 0～59 までの分桁の値。

uint8\_t

**Sec:** 0～59 までの秒桁の値。

## 16.2.4.3 RTC\_AlarmTypeDef

メンバ:

uint8\_t

**Date:** アラーム機能有効時の日桁の値(1～31)。



uint8\_t

**Day:** アラーム機能有効時の週桁の値。

- **RTC\_SUN:** 日曜日
- **RTC\_MON:** 月曜日
- **RTC\_TUE:** 火曜日
- **RTC\_WED:** 水曜日
- **RTC\_THU:** 木曜日
- **RTC\_FRI:** 金曜日
- **RTC\_SAT:** 土曜日

uint8\_t

➤ **Hour:** アラーム機能有効時の時間桁の値。

uint8\_t

**AmPm:** アラーム機能有効時の AM/PM 選択の値:

- **RTC\_AM\_MODE:** AM モード
- **RTC\_PM\_MODE:** PM モード
- **RTC\_AMPM\_INVALID:** 24 時間モード

uint8\_t

**Min:** アラーム機能有効時の分桁の値(0～59)。

## 17. SBI

### 17.1 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な 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/TX04\_Periph\_Driver/src/tmpm440\_sbi.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_sbi.h

### 17.2 API 関数

#### 17.2.1 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**);

## 17.2.2 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\_SetSendData(), SBI\_GetReceiveData()
- 3) ステータス確認:  
SBI\_GetI2CState()
- 4) その他:  
SBI\_SWReset(), SBI\_SetIdleMode(), SBI\_EnableI2CfreeDataMode()

## 17.2.3 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 **TSB\_SBI0** となります。

### 17.2.3.1 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

### 17.2.3.2 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

### 17.2.3.3 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

### 17.2.3.4 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

## 17.2.3.5 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

## 17.2.3.6 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

## 17.2.3.7 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

## 17.2.3.8 SBI\_GeneratI2CStart

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\_GeneratI2CStart(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

## 17.2.3.9 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

## 17.2.3.10 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

## 17.2.3.11 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

## 17.2.3.12 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

## 17.2.3.13 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

## 17.2.3.14 SBI\_SetI2CFreeDataMode

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

## 17.2.4 データ構造

### 17.2.4.1 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。

### 17.2.4.2 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

## 18. TMRB

### 18.1 概要

本デバイスは、20 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 ~ TMRB19)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。

- 外部トリガパルスからのワンショットパルス出力
- 周波数測定
- パルス幅測定

本デバイスは、16 ビットの多目的タイマ (MPT)を内蔵しており、MPT はタイマーモードで動作する場合、TMRB と同一の動作を行います。

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/TX04\_Periph\_Driver/src/tmpm440\_tmr.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_tmr.h

### 18.2 API 関数

#### 18.2.1 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);
◆ TMRB_INTMask TMRB_GetINTMask(TSB_TB_TypeDef * TBx);
◆ void TMRB_SetINTMask(TSB_TB_TypeDef * TBx, uint32_t INTMask);
◆ void TMRB_ChangeLeadingTiming(TSB_TB_TypeDef * TBx, uint32_t LeadingTiming);
◆ void TMRB_ChangeTrailingTiming(TSB_TB_TypeDef * TBx, uint32_t TrailingTiming);
◆ uint16_t TMRB_GetRegisterValue(TSB_TB_TypeDef * TBx, uint8_t Reg);
◆ uint16_t TMRB_GetUpCntValue(TSB_TB_TypeDef * TBx);
◆ uint16_t TMRB_GetCaptureValue(TSB_TB_TypeDef * TBx, uint8_t CapReg);
◆ void TMRB_ExecuteSWCapture(TSB_TB_TypeDef * TBx);
◆ void TMRB_SetIdleMode(TSB_TB_TypeDef * TBx, FunctionalState NewState);
◆ void TMRB_SetSyncMode(TSB_TB_TypeDef * TBx, FunctionalState NewState);
◆ void TMRB_SetDoubleBuf(TSB_TB_TypeDef * TBx, FunctionalState NewState);
◆ void TMRB_SetExtStartTrg(TSB_TB_TypeDef * TBx, FunctionalState NewState,
                           uint8_t TrgMode);
◆ void TMRB_SetClkInCoreHalt(TSB_TB_TypeDef * TBx, uint8_t ClkState);
◆ void TMRB_SetExtInput(TSB_TB_TypeDef * TBx);
◆ void TMRB_SetDMAReq(TSB_TB_TypeDef * TBx, FunctionalState NewState,
                      uint8_t DMAReq);

```

## 18.2.2 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\_GetIntMask(), TMRB\_GetRegisterValue,  
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()

## 18.2.3 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,
TSB_TB10,   TSB_TB11,   TSB_TB12,   TSB_TB13,   TSB_TB14,

```

TSB\_TB15, TSB\_TB16, TSB\_TB17, TSB\_TB18, TSB\_TB19

## 18.2.3.1 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

## 18.2.3.2 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

## 18.2.3.3 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

## 18.2.3.4 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、デューティ期間の初期設定を行います。

戻り値:

なし

## 18.2.3.5 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**: 以下から、TMRB チャンネルを指定します。

TSB\_TB7, TSB\_TB8, TSB\_TB9, TSB\_TB10, TSB\_TB11,  
TSB\_TB12, TSB\_TB13, TSB\_TB14, TSB\_TB15, TSB\_TB16,  
TSB\_TB17, TSB\_TB18, TSB\_TB19

**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。

## 戻り値:

なし

### 18.2.3.6 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

## 18.2.3.7 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
}
```

```
}
```

## 18.2.3.8 TMRB\_GetINTMask

割り込みマスク要因の取得

**関数のプロトタイプ宣言:**

TMRB\_INTMask

TMRB\_GetINTMask(TSB\_TB\_TypeDef\* **TBx**)

**引数:**

**TBx:** TMRB チャンネルを指定します。

**機能:**

割り込みマスク要因を取得します。

**戻り値:**

マスクする割り込み要因です。

- **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。

## 18.2.3.9 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

## 18.2.3.10 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 **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:**

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

## 18.2.3.11 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。

## 18.2.3.12 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

## 18.2.3.13 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:** 以下から TMRB チャンネルを指定します。

TSB\_TB7, TSB\_TB8, TSB\_TB9, TSB\_TB10, TSB\_TB11,  
TSB\_TB12, TSB\_TB13, TSB\_TB14, TSB\_TB15, TSB\_TB16,  
TSB\_TB17, TSB\_TB18, TSB\_TB19

**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

## 18.2.3.14 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:** 以下から TMRB チャンネルを指定します。

TSB\_TB7, TSB\_TB8, TSB\_TB9, TSB\_TB10, TSB\_TB11,  
TSB\_TB12, TSB\_TB13, TSB\_TB14, TSB\_TB15, TSB\_TB16,  
TSB\_TB17, TSB\_TB18, TSB\_TB19

**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

## 18.2.3.15 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

### 18.2.3.16 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**: 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, TSB\_TB8, TSB\_TB10,  
TSB\_TB11, TSB\_TB12, TSB\_TB13, TSB\_TB14, TSB\_TB15,  
TSB\_TB16, TSB\_TB17

**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:

TMRB00～TMRB03 を同期モードに設定すると、TMRB00 のスタートに同期して動作がスタートし、TMRB04～TMRB07 を同期モードに設定すると、TMRB04 のスタートに同期して動作がスタートし、TMRB10～TMRB13 を同期モードに設定すると、TMRB10 のスタート

に同期して動作がスタートし、TMRB14~TMRB17 を同期モードに設定すると、TMRB14 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、TMRB00, TMRB04, TMRB14, TMRB17 のカウントを開始する前に、**TMRB\_SetRunState()** によって TMRB00~TMRB03、TMRB04~TMRB07、TMRB10~TMRB13、TMRB14~TMRB17 をスタートしてください。

### 18.2.3.17 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

### 18.2.3.18 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

## 18.2.3.19 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



## 18.2.3.20 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)
```

**引数:**

**TBx:** TMRB チャンネルを選択します。

**機能:**

外部入力として TBxIN0/1 を設定します。

**戻り値:**

なし

## 18.2.3.21 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:**コンペアー致
- **TMRB\_DMA\_REQ\_CAPTURE\_1:** インプットキャプチャ 1
- **TMRB\_DMA\_REQ\_CAPTURE\_0:** インプットキャプチャ 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。

## 18.2.4 データ構造

### 18.2.4.1 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 (TMRB0~TMRB7 only)
- **TMRB\_CLK\_DIV\_128:** fperiph / 128 (TMRB0~TMRB7 only)
- **TMRB\_CLK\_DIV\_256:** fperiph / 256 (TMRB0~TMRB7 only)
- **TMRB\_CLK\_DIV\_512:** fperiph / 512 (TMRB0~TMRB7 only)

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。

### 18.2.4.2 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。

### 18.2.4.3 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 -

## 19. TMRC

### 19.1 概要

本デバイスは、32bit タイマ(TMRC)を 1 ユニット内蔵しています。1 ユニットには 32 ビットタイムベースタイマー(TCT)を 1 チャンネル、32 ビットインプットキャプチャ-レジスタ(TCCAP0 ~ 3)を 4 チャンネルの 32 ビットアウトプットコンペアレジスタ (TCCMP0 ~ 7)を 8 チャンネル内蔵しています。

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/TX04\_Periph\_Driver/src/tmpm440\_tmrc.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_tmrc.h

### 19.2 API 関数

#### 19.2.1 関数一覧

- ◆ void TMRC\_Enable(TSB\_TC\_TypeDef \* **TCx**);
- ◆ void TMRC\_Disable(TSB\_TC\_TypeDef \* **TCx**);
- ◆ void TMRC\_SetRunState(TSB\_TC\_TypeDef \* **TCx**, uint32\_t **Cmd**);
- ◆ void TMRC\_SetIdleMode(TSB\_TC\_TypeDef \* **TCx**, FunctionalState **NewState**);
- ◆ void TMRC\_ExecuteSWCapture(TSB\_TC\_TypeDef \* **TCx**);
- ◆ void TMRC\_SetSrcClk(TSB\_TC\_TypeDef \* **TCx**, uint32\_t **ClkDiv**);
- ◆ void TMRC\_SetNoiseFilter(TSB\_TC\_TypeDef \* **TCx**, TMRC\_NoiseFilter **NosFlt**);
- ◆ uint32\_t TMRC\_GetSWCaptureValue(TSB\_TC\_TypeDef \* **TCx**);
- ◆ uint32\_t TMRC\_GetReadCaptureValue(TSB\_TC\_TypeDef \* **TCx**);
- ◆ void TMRC\_CMPRegConfig(TSB\_TC\_TypeDef \* **TCx**, TMRC\_CMPConfigTypeDef \* **CMPConfigStruct**);
- ◆ void TMRC\_SetCMPRegValue(TSB\_TC\_TypeDef \* **TCx**, TMRC\_CMPReg **CMPRegNum**, uint32\_t **CmpRegVal**);
- ◆ void TMRC\_CAPRegConfig(TSB\_TC\_TypeDef \* **TCx**, TMRC\_CAPConfigTypeDef \* **CAPConfigStruct**);
- ◆ uint32\_t TMRC\_GetCntCaptureValue(TSB\_TC\_TypeDef \* **TCx**, TMRC\_CAPReg **CAPRegNum**);

#### 19.2.2 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) 共通機能の構成と制御:  
TMRC\_Enable(), TMRC\_Disable(), TMRC\_SetRunState(), TMRC\_CAPRegConfig(),  
TMRC\_CMPRegConfig ()
- 2) キャプチャ機能:  
TMRC\_ExecuteSWCapture()
- 3) ステータス表示:  
TMRC\_GetCntCaptureValue (), TMRC\_GetReadCaptureValue (),  
TMRC\_GetSWCaptureValue (), TMRC\_SetCMPRegValue()
- 4) その他:  
TMRC\_SetIdleMode(), TMRC\_SetNoiseFilter(), TMRC\_SetSrcClk()

## 19.2.3 関数仕様

補足: 引数に記載されている“TSB\_TC\_TypeDef\* **TCx**” は特に記述のない限り以下のようになります。

**TSB\_TC**

### 19.2.3.1 TMRC\_Enable

TMRC 機能の許可

**関数のプロトタイプ宣言:**

void

TMRC\_Enable(TSB\_TC\_TypeDef\* **TCx**)

**引数:**

**TCx:** TMRC チャンネルを指定します。

**機能:**

TMRC 機能を有効にします。

**戻り値:**

なし

### 19.2.3.2 TMRC\_Disable

TMRC 機能の禁止

**関数のプロトタイプ宣言:**

void

TMRC\_Disable(TSB\_TC\_TypeDef\* **TCx**)

**引数:**

**TCx:** TMRC チャンネルを指定します。

**機能:**

TMRC 機能を無効にします。

**戻り値:**

なし

### 19.2.3.3 TMRC\_SetRunState

カウンタ動作の設定

**関数のプロトタイプ宣言:**

```
void  
TMRC_SetRunState(TSB_TC_TypeDef* TCx,  
                  uint32_t Cmd)
```

**引数:**

**TCx:** TMRC チャンネルを指定します。

**Cmd:** カウンタ動作を選択します。

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

**機能:**

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

**Cmd** が **TMRC\_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

### 19.2.3.4 TMRC\_SetIdleMode

IDLE 時の動作設定

**関数のプロトタイプ宣言:**

```
void  
TMRC_SetIdleMode(TSB_TC_TypeDef* TCx,  
                  FunctionalState NewState)
```

**引数:**

**TCx:** TMRC 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 TMRC 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

### 19.2.3.5 TMRC\_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

TMRC\_ExecuteSWCapture(TSB\_TC\_TypeDef\* **TCx**)

**引数:**

**TCx:** TMRC チャンネルを指定します。

**機能:**

キャプチャレジスタ 0 (TCxCP0)にカウント値を取り込みます。

**戻り値:**

なし

### 19.2.3.6 TMRC\_SetSrcClk

TBT のソースクロック選択

**関数のプロトタイプ宣言:**

void

TMRC\_SetSrcClk (TSB\_TC\_TypeDef\* **TCx**,  
uint32\_t **ClkDiv**)

**引数:**

**TCx:** TMRC チャンネルを指定します。

**ClkDiv:** 以下から TCT のソースクロックを選択します。

- **TMRC\_CLK\_DIV\_4:**  $\phi T2$

- **TMRC\_CLK\_DIV\_8:**  $\phi T4$
- **TMRC\_CLK\_DIV\_16:**  $\phi T8$
- **TMRC\_CLK\_DIV\_32:**  $\phi T16$
- **TMRC\_CLK\_DIV\_64:**  $\phi T32$
- **TMRC\_CLK\_DIV\_128:**  $\phi T64$
- **TMRC\_CLK\_DIV\_256:**  $\phi T128$
- **TMRC\_CLK\_DIV\_512:**  $\phi T256$
- **TMRC\_CLK\_TCTBT\_IN:** TCTBTIN 端子入力

**機能:**

TBT ソースクロックを選択します。

**戻り値:**

なし

### 19.2.3.7 TMRC\_SetNoiseFilter

TCTBTIN 入力ノイズ除去の設定

**関数のプロトタイプ宣言:**

```
void  
TMRC_SetNoiseFilter (TSB_TC_TypeDef* TCx,  
                    TMRC_NoiseFilter NosFlt)
```

**引数:**

**TCx:** TMRC チャンネルを指定します。

**NosFlt:** TCTBTIN 入力ノイズ除去を選択します。

- **TMRC\_FILTER:** ノイズ除去あり(2/fsys 以上)
- **TMRC\_NOFILTER:** ノイズ除去なし(6/fsys 以上)

**機能:**

TCTBTIN 入力ノイズ除去を制御します。

**戻り値:**

なし

### 19.2.3.8 TMRC\_GetSWCaptureValue

キャプチャー値の取得

**関数のプロトタイプ宣言:**



void

TMRC\_GetSWCaptureValue (TSB\_TC\_TypeDef\* **TCx**)

引数:

**TCx**: TMRC チャンネルを指定します。

機能:

キャプチャー値を取得します。

戻り値:

なし

### 19.2.3.9 TMRC\_GetReadCaptureValue

リードキャプチャー値の取得

関数のプロトタイプ宣言:

void

TMRC\_GetReadCaptureValue (TSB\_TC\_TypeDef\* **TCx**)

引数:

**TCx**: TMRC チャンネルを指定します。

機能:

リードキャプチャー値を取得します。

戻り値:

なし

### 19.2.3.10 TMRC\_CMPRegConfig

コンペア条件の設定

関数のプロトタイプ宣言:

void

TMRC\_CMPRegConfig (TSB\_TC\_TypeDef\* **TCx**,  
TMRC\_CMPConfigTypeDef **CMPCConfigStruct**)

引数:

**TCx**: TMRC チャンネルを指定します。

**CMPConfigStruct:** コンペア条件の基本設定情報を含む構造体です。(詳細は“データ構造”を参照してください)

**機能:**

コンペア条件を設定します。

**戻り値:**

なし

## 19.2.3.11 TMRC\_SetCMPRegValue

カウンタ比較値の設定

**関数のプロトタイプ宣言:**

void

TMRC\_SetCMPRegValue (TSB\_TC\_TypeDef\* **TCx**,  
TMRC\_CMPReg **CMPRegNum**,  
uint32\_t **CMPRegVal**)

**引数:**

**TCx:** TMRC チャンネルを指定します。

**CMPRegNum:** 以下から比較設定するレジスタを選択します。

- **TMRC\_CMP\_0:** コンペアレジスタ 0
- **TMRC\_CMP\_1:** コンペアレジスタ 1
- **TMRC\_CMP\_2:** コンペアレジスタ 2
- **TMRC\_CMP\_3:** コンペアレジスタ 3
- **TMRC\_CMP\_4:** コンペアレジスタ 4
- **TMRC\_CMP\_5:** コンペアレジスタ 5
- **TMRC\_CMP\_6:** コンペアレジスタ 6
- **TMRC\_CMP\_7:** コンペアレジスタ 7

**CMPRegVal:** 比較する値を設定します。最大値は 0xFFFFFFFF です。

**機能:**

カウンタ比較値を設定します。

**戻り値:**

なし

## 19.2.3.12 TMRC\_CAPRegConfig

キャプチャ条件の設定

**関数のプロトタイプ宣言:**

```
void  
TMRC_CMPRegConfig (TSB_TC_TypeDef* TCx,  
                   TMRC_CAPConfigTypeDef CAPConfigStruct)
```

**引数:**

**TCx:** TMRC チャンネルを指定します。

**CAPConfigStruct:** キャプチャ条件の基本設定情報を含む構造体です。(詳細は“データ構造”を参照してください)

**機能:**

キャプチャ条件を設定します。

**戻り値:**

なし

### 19.2.3.13 TMRC\_GetCntCaptureValue

キャプチャー値の取得

**関数のプロトタイプ宣言:**

```
void  
TMRC_GetCntCaptureValue (TSB_TC_TypeDef* TCx,  
                         TMRC_CAPReg CAPRegNum)
```

**引数:**

**TCx:** TMRC チャンネルを指定します。

**CAPRegNum:**..以下から取得するキャプチャレジスタを選択します。

- **TMRC\_CAP\_0:** キャプチャレジスタ 0
- **TMRC\_CAP\_1:** キャプチャレジスタ 1
- **TMRC\_CAP\_2:** キャプチャレジスタ 2
- **TMRC\_CAP\_3:** キャプチャレジスタ 3

**機能:**

キャプチャ値を取得します。

**戻り値:**

なし

## 19.2.4 データ構造

## 19.2.4.1 TMRC\_CMPConfigTypeDef

メンバ:

uint32\_t

**TMRC\_CMPReg:** 以下からコンペアレジスタを選択します。

- **TMRC\_CMP\_0:** コンペアレジスタ 0
- **TMRC\_CMP\_1:** コンペアレジスタ 1
- **TMRC\_CMP\_2:** コンペアレジスタ 2
- **TMRC\_CMP\_3:** コンペアレジスタ 3
- **TMRC\_CMP\_4:** コンペアレジスタ 4
- **TMRC\_CMP\_5:** コンペアレジスタ 5
- **TMRC\_CMP\_6:** コンペアレジスタ 6
- **TMRC\_CMP\_7:** コンペアレジスタ 7

uint32\_t

**TMRC\_FFReverseTrg:** 以下から TCFF0 反転トリガを選択します。

- **TMRC\_FF\_REVERSE\_TRG\_DISABLE:** トリガディゼーブル
- **TMRC\_FF\_REVERSE\_TRG\_ENABLE:** トリガイネーブル

uint32\_t

**TMRC\_FlipFlopCtrl:** TCFF0 の制御

- **TMRC\_FLIPFLOP\_INVERT:** TCFF0 の値を反転(ソフト反転)します。
- **TMRC\_FLIPFLOP\_SET:** TCFF0 を"1"にセットします。
- **TMRC\_FLIPFLOP\_CLEAR:** TCFF0 に"0"をセットします。

uint32\_t

**TMRC\_CMPDBCtrl:** コンペアレジスタのダブルバッファを選択します。

- **TMRC\_CMP\_DB\_DISABLE:** 無効
- **TMRC\_CMP\_DB\_ENABLE:** 有効

uint32\_t

**ModeSetting:** コンペア一致検出を選択します。

- **TMRC\_CMP\_MATCH\_DISABLE:** 無効
- **TMRC\_CMP\_MATCH\_ENABLE:** 有効

## 19.2.4.2 TMRC\_CAPConfigTypeDef

メンバ:

uint32\_t

**TMRC\_CAPReg:** 以下からキャプチャレジスタを選択します。

- **TMRC\_CAP\_0:** キャプチャレジスタ 0
- **TMRC\_CAP\_1:** キャプチャレジスタ 1
- **TMRC\_CAP\_2:** キャプチャレジスタ 2
- **TMRC\_CAP\_3:** キャプチャレジスタ 3

uint32\_t

**TMRC\_NoiseFilter:** 以下から TCIN0 ~ 3 端子入のノイズ除去を選択します。

- **TMRC\_NOFILTER:** 2/fsys 以上
- **TMRC\_FILTER:** 4/fsys 以上

uint32\_t

**TMRC\_ValidEdgeSel:** TCIN0 ~ 3 入力の有効エッジを選択します。

- **TMRC\_NO\_CAPTURE:** キャプチャしない
- **TMRC\_IN\_RISING\_EDGE:** 立ち上がり
- **TMRC\_IN\_FALLING\_EDGE:** 立ち下り
- **TMRC\_IN\_BOTH\_EDGE:** 両エッジ

## 20. TMRD

### 20.1 概要

本デバイスは、高分解能 16 ビットタイマ出力を 1 ブロック内蔵しています。1 ブロックには 2 ユニットのタイマが内蔵されており、それぞれタイマ出力を 2 チャンネル内蔵しています。

TMRD は、2 つのタイマユニット(TMRD0、TMRD1)とこれらタイマユニットにクロックを供給する 2 つのクロック設定回路(プリスケアラ)から構成され、以下の機能を内蔵しています。

- 16 ビットインターバルタイマ
- 16 ビットプログラマブル矩形波出力 (PPG)

16 ビットインターバルタイマでは、以下の 2 つのモードを内蔵しています。

- タイマモード: TMRD0 と TMRD1 が独立して動作します。
- 連動タイマモード: TMRD0 と TMRD1 のタイマ動作が同時にスタートします。

16 ビットプログラマブルパルス矩形波出力では、以下の 2 つのモードを内蔵しています。

- PPG モード: TMRD0 と TMRD1 は独立して動作し、プログラムされた 2ch+2ch の矩形波を出力します。
- 連動 PPG モード: TMRD0 と TMRD1 は同時動作し、プログラムされた 3ch+1ch あるいは 4ch の矩形波を出力します。

本ドライバ API は、クロック動作、タイミング制御、PPG 出力制御、波形制御、DMA 要求の設定、割り込み要因、アップカウンタのクリアなどの TMRD 設定を行います。

全ドライバ API は、マクロ、データタイプ、構造、API を定義する以下のファイルで構成されています。  
/Libraries/TX04\_Periph\_Driver/src/tmpm440\_tmr.c  
/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_tmr.h

### 20.2 API 関数

#### 20.2.1 関数一覧

- ◆ void TMRD\_Enable(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**)
- ◆ void TMRD\_Disable(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**)
- ◆ void TMRD\_SetRunStateInHalt(TSB\_TD\_TypeDef \* **TDx**, uint8\_t **RunState**)
- ◆ void TMRD\_SetRunStateInIdle(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **RunState**)
- ◆ void TMRD\_SetMode(TSB\_TD\_TypeDef \* **TDx**, uint8\_t **Mode**)
- ◆ void TMRD\_SetClkDivision(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **ClkDiv**)

- ◆ void TMRD\_SetUpCntCtrl(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **UpCntCtrl**)
- ◆ void TMRD\_SetPPGInitLeadingEdge(TSB\_TD\_TypeDef \* **TDx**, uint8\_t **PPGChannel**,  
uint8\_t **WaveEdge**)
- ◆ void TMRD\_SetCMPRegWritePath(TSB\_TD\_TypeDef \* **TDx**,  
TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **WritePath**)
- ◆ void TMRD\_SetCMP0INTSrc(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **INTSrc**)
- ◆ void TMRD\_SetRunState(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **RunState**)
- ◆ void TMRD\_SetPhaseRelation(TSB\_TD\_TypeDef \* **TDx**, uint8\_t **PhaseRelation**)
- ◆ void TMRD\_EnableUpdateCMPReg(TSB\_TD\_TypeDef \* **TDx** ,  
TMRD\_UNIT\_Channel **CHx**)
- ◆ void TMRD\_SetDMAReq(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
FunctionalState **NewState**)
- ◆ void TMRD\_SetInitTiming(TSB\_TD\_TypeDef \* **TDx**, TMRD\_UNIT\_Channel **CHx**,  
TMRD\_TimingTypeDef \* **TimingStruct**)
- ◆ void TMRD\_ChangeTiming(TSB\_TD\_TypeDef \* **TDx**, uint8\_t **TimingType**,  
uint32\_t **Timing**)
- ◆ uint16\_t TMRD\_GetTiming(TSB\_TD\_TypeDef \* **TDx**, uint8\_t **TimingType**)
- ◆ void TMRD\_SetBitModulationCycle(TSB\_TD\_TypeDef \* **TDx**,  
TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **BitModCycle**)
- ◆ void TMRD\_SetBitModUpdateTiming(TSB\_TD\_TypeDef \* **TDx**,  
TMRD\_UNIT\_Channel **CHx**,  
FunctionalState **NewState**)

## 20.2.2 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) 各タイマの設定:  
TMRD\_Enable(), TMRD\_Disable(), TMRD\_SetUpCntCtrl (), TMRD\_SetMode(),  
TMRD\_SetRunState()
- 2) クロック制御:  
TMRD\_SetRunStateInHalt(), TMRD\_SetRunStateInIdle(), TMRD\_SetClkDivision()
- 3) PPG 出力や矩形波の制御:  
TMRD\_SetPPGInitLeadingEdge(), TMRD\_SetPhaseRelation(),  
TMRD\_SetBitModulationCycle(), TMRD\_SetBitModUpdateTiming()
- 4) コンペアレジスタやタイマレジスタの制御:  
TMRD\_SetCMPRegWritePath(), TMRD\_EnableUpdateCMPReg(), TMRD\_SetInitTiming(),  
TMRD\_ChangeTiming(), TMRD\_GetTiming()
- 5) 割り込み要因や DMA 要求の設定:  
TMRD\_SetCMP0INTSrc(), TMRD\_SetDMAReq()

## 20.2.3 関数仕様

補足: 引数に記載されている“TSB\_TD\_TypeDef \* **TDx**”は以下を指定してください。

**TSB\_TD.**

補足: 引数に記載されている“TMRD\_UNIT\_Channel **CHx**”は、特に断りのない限り、以下のいずれかを選択してください。

**TMRD\_UNIT\_CH\_0, TMRD\_UNIT\_CH\_1**

### 20.2.3.1 TMRD\_Enable

クロック供給の許可

関数のプロトタイプ宣言:

void

```
TMRD_Enable(TSB_TD_TypeDef * TDx,  
             TMRD_UNIT_Channel CHx)
```

引数:

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。

機能:

クロック供給を有効にします。

戻り値:

なし

### 20.2.3.2 TMRD\_Disable

クロック動作の禁止

関数のプロトタイプ宣言:

void

```
TMRD_Disable(TSB_TD_TypeDef * TDx,  
             TMRD_UNIT_Channel CHx)
```

引数:

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。



**機能:**

クロック供給を無効にします。

**戻り値:**

なし

### 20.2.3.3 TMRD\_SetRunStateInHalt

デバッグ中の動作設定(HALT 時のアップカウンタ)

**関数のプロトタイプ宣言:**

void

TMRD\_SetRunStateInHalt(TSB\_TD\_TypeDef \* **TDx**,  
uint8\_t **RunState**)

**引数:**

**TDx**: TMRD ユニットを選択します。

**RunState**: デバッグ中の動作設定(HALT 中のアップカウンタ)を選択します。

- **TMRD\_RUN**: 動作(アップカウンタは停止しません)
- **TMRD\_STOP**: 停止(アップカウンタのみ停止します)

**機能:**

デバッグ中の動作(HALT 中のアップカウンタ)を設定します。

**戻り値:**

なし

### 20.2.3.4 TMRD\_SetRunStateInIdle

IDLE 中の動作設定

**関数のプロトタイプ宣言:**

void

TMRD\_SetRunStateInIdle(TSB\_TD\_TypeDef \* **TDx**,  
TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **RunState**)

**引数:**

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。

**RunState:** IDLE 中の動作を選択します。

- **TMRD\_RUN:** 動作
- **TMRD\_STOP:** 停止

**機能:**

IDLE 中の動作を設定します。

**戻り値:**

なし

## 20.2.3.5 TMRD\_SetMode

動作モードの選択

**関数のプロトタイプ宣言:**

void

TMRD\_SetMode(TSB\_TD\_TypeDef \* **TDx**,  
uint8\_t **Mode**)

**引数:**

**TDx:** TMRD ユニットを選択します。

**Mode:** 以下から動作モードを選択します。

マクロ定義	TMRD0 の動作モード	TMRD1 の動作モード
<b>TMRD_MODE_BOTH_TMR</b>	タイマモード	タイマモード
<b>TMRD_MODE_0TMR_1PPG</b>	タイマモード	PPG モード
<b>TMRD_MODE_0PPG_1TMR</b>	PPG モード	タイマモード
<b>TMRD_MODE_BOTH_PPG</b>	PPG モード	PPG モード
<b>TMRD_MODE_INTERLOCK_TMR</b>	TMRD0 と TMRD1 を同時スタートさせるタイマモード	
<b>TMRD_MODE_INTERLOCK_PPG_2CH</b>	TMRD0 と TMRD1 が連動する PPG モード (TMRD0 の ch00 と TMRD1 の ch10 の更新タイミングが同期)	
<b>TMRD_MODE_INTERLOCK_PPG_3CH</b>	TMRD0 と TMRD1 が連動する PPG モード (TMRD0 の ch00 と TMRD1 の全 ch の更新タイミングが同期)	

**機能:**

TMRD0 と TMRD1 の動作モードを選択します。

**戻り値:**

なし

**補足:**

動作モードが **TMRD\_MODE\_INTERLOCK\_PPG\_2CH** または **TMRD\_MODE\_INTERLOCK\_PPG\_3CH** の場合、TMRDCLK0とTMRDCLK1は個別に設定できません。TMRDCLK1とTMRDCLK0は同じ周波数になります。

## 20.2.3.6 TMRD\_SetClkDivision

プリスケアラの設定

**関数のプロトタイプ宣言:**

void

```
TMRD_SetClkDivision(TSB_TD_TypeDef* TDx,  
                    TMRD_UNIT_Channel CHx,  
                    uint8_t ClkDiv)
```

**引数:**

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャネルを選択します。

**ClkDiv**: 以下からプリスケアラを選択します。

- **TMRD\_CLK\_DIV\_1**: ftmrd
- **TMRD\_CLK\_DIV\_2**: ftmrd/2
- **TMRD\_CLK\_DIV\_4**: ftmrd/4
- **TMRD\_CLK\_DIV\_8**: ftmrd/8
- **TMRD\_CLK\_DIV\_16**: ftmrd/16

**機能:**

TMRD0とTMRD1のプリスケアラ(TMRDCLK0/1)を設定します。

**戻り値:**

なし

**補足:**

連動 PPG モードの場合、TMRD1 のプリスケアラ選択は無効となり、TMRD0 で選択したプリスケアラクロックで動作します。したがって、この場合は TMRD0 のプリスケアラの設定のみを行ってください。

## 20.2.3.7 TMRD\_SetUpCntCtrl

CP00/CP10 一致時のアップカウンタ 0/1(UC0/UC1)動作設定

**関数のプロトタイプ宣言:**

Void

```
TMRD_SetUpCntCtrl(TSB_TD_TypeDef * TDx,  
                  TMRD_UNIT_Channel CHx,  
                  uint8_t UpCntCtrl)
```

**引数:**

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。

**UpCntCtrl**: CP00/CP10 一致時のアップカウンタ動作を選択します。

- **TMRD\_FREE\_RUN**: 一致検出にかかわらずフリーランカウンタとして動作
- **TMRD\_AUTO\_CLEAR**: 一致検出で"0"に初期化

**機能:**

CP00/CP10 一致時のアップカウンタ 0/1(UC0/UC1)動作を設定します。

**戻り値:**

なし

**補足:**

連動 PPC モードでは、**UpCntCtrl = TMRD\_FREE\_RUN** の設定は無効です。

## 20.2.3.8 TMRD\_SetPPGInitLeadingEdge

信号 a0/a1、b0/b1 の leading/trailing edge の初期設定

**関数のプロトタイプ宣言:**

void

```
TMRD_SetPPGInitLeadingEdge(TSB_TD_TypeDef * TDx,  
                            uint8_t PPGChannel,  
                            uint8_t WaveEdge)
```

**引数:**

**TDx**: TMRD ユニットを選択します。

**PPGChannel**: 以下から PPG 出力チャンネルを選択します。

- **TMRD\_PPG\_CHANNEL\_A0**: ch00 の PPG 出力信号 a0
- **TMRD\_PPG\_CHANNEL\_A1**: ch00 の PPG 出力信号 a1
- **TMRD\_PPG\_CHANNEL\_B0**: ch10 の PPG 出力信号 b0
- **TMRD\_PPG\_CHANNEL\_B1**: ch10 の PPG 出力信号 b1

**WaveEdge**: 以下から leading edge/trailing edge を選択します。

- **TMRD\_WAVE\_EDGE\_RISING**: Leading edge が立ち上がり、trailing edge が立下りです。
- **TMRD\_WAVE\_EDGE\_FALLING**: Leading edge が立ち上がり、trailing edge が立下りです。

**機能:**  
信号 a0/a1、b0/b1 の leading/trailing edge の初期設定を行います。

**戻り値:**  
なし

20.2.3.9 TMRD\_SetCMPRegWritePath

コンペアレジスタへのデータ書き込み経路設定

**関数のプロトタイプ宣言:**  
void  
TMRD\_SetCMPRegWritePath(TSB\_TD\_TypeDef\* **TDx**,  
TMRD\_UNIT\_Channel **CHx**,  
uint8\_t **WritePath**)

**引数:**  
**TDx**: TMRD ユニットを選択します。  
**CHx**: TMRD チャンネルを選択します。

**WritePath**: コンペアレジスタへのデータ書き込み経路を選択します。  
➤ **TMRD\_CMP\_WRITE\_DIRECT**: CPI の命令によるダイレクト書き込み  
➤ **TMRD\_CMP\_WRITE\_INDIRECT**: タイマレジスタ経由書き込み(更新フラグを設定してください)

**機能:**  
コンペアレジスタへのデータ書き込み経路を設定します。  
**WritePath** が **TMRD\_CMP\_WRITE\_DIRECT** の場合、タイマレジスタへの書き込みと同時に、同値が対応するコンペアレジスタに書き込まれます。  
**WritePath** が **TMRD\_CMP\_WRITE\_INDIRECT** の場合、TMRD\_EnableUpdateCMPReg()による更新フラグの設定が必要です。  
それぞれのモードの機能については以下を参照してください。

タイマモード時	TDnmMOD<TDCLE>="0"	アップカウンタのオーバーフロー時のコンペアレジスタ(TDmnCPx)の値が、タイマレジスタ(TDmnRGx)の値に更新されます。
	TDnmMOD<TDCLE>="1"	コンパレータ(CP00/CP10)の一致時に

		コンペアレジスタ(TDmnCPx)の値が、タイマレジスタ(TDmnRGx)の値に更新されます。
PPG モード	コンペアレジスタ (CP00/CP10) の一致時にコンペアレジスタ (TDmnCPx)の値が、タイマレジスタ(TDmnRGx)の値に更新されます。	
連動 PPG モード	TMRD0 の更新方法は、PPG モードと同じです。 TMRD1 の更新方法は、次の通りです。 コンパレータ 05(UC05)の一致時にコンペアレジスタ(TDmnCPx)の値が、タイマレジスタ(TDmnRGx)の値に更新されます。	

**戻り値:**

なし

**補足:**

連動 PPG モードの場合、TMRD1 のプリスケアラ選択は無効となり、TMRD0 で選択したプリスケアラクロックで動作します。したがって、この場合は TMRD0 のプリスケアラの設定のみを行ってください。

## 20.2.3.10 TMRD\_SetCMP0INTSrc

INTTDxCMP0 の割り込み要因設定

**関数のプロトタイプ宣言:**

```
void
TMRD_SetCMP0INTSrc(TSB_TD_TypeDef* TDx,
                    TMRD_UNIT_Channel CHx,
                    uint8_t INTSrc)
```

**引数:**

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。

**INTSrc**: INTTDxCMP0 割り込み要因を選択します。

- **TMRD\_INT\_NONE**: 割り込み要因なし
- **TMRD\_INT\_MATCH\_CYCLE**: CP00/CP10 の一致
- **TMRD\_INT\_MATCH\_PHASE**: CP05(TMRD0 のみ)の一致
- **TMRD\_INT\_UC\_OVERFLOW**: COUNTER0/1(UC0/UC1)のオーバーフロー

**機能:**

INTTDxCMP0 の割り込み要因を設定します。

**戻り値:**

なし

**補足:**

PPG モードでは、**TMRD\_INT\_UC\_OVERFLOW** は無効で割り込み要因となりません。  
連動 PPG モードでは、TMRD1 の **TMRD\_INT\_MATCH\_CYCLE** は無効で割り込み要因となりません。  
TMRD1 の **TMRD\_INT\_MATCH\_PHASE** は無効で割り込み要因はありません。

## 20.2.3.11 TMRD\_SetRunState

カウンタ動作の設定

**関数のプロトタイプ宣言:**

```
void  
TMRD_SetRunState(TSB_TD_TypeDef* TDx,  
                  TMRD_UNIT_Channel CHx,  
                  uint8_t RunState)
```

**引数:**

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。

**RunState**: カウンタ動作を選択します。

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

**機能:**

カウンタ動作を設定します。  
す。

**戻り値:**

なし

**補足:**

連動タイマモード及び連動 PPG モードの場合、設定は無効となり COUNTER(UC0)と連動して動作を開始します。

## 20.2.3.12 TMRD\_SetPhaseRelation

A 相出力に対する B 相出力の位相関係の設定

**関数のプロトタイプ宣言:**

void

TMRD\_SetPhaseRelation(TSB\_TD\_TypeDef\* **TDx**,  
uint8\_t **PhaseRelation**)

引数:

**TDx**: TMRD ユニットを選択します。

**PhaseRelation**: A 相出力に対する B 相出力の位相関係を選択します。

- **TMRD\_PHASE\_DELAY\_OR\_SAME**: 遅らせる、または同位相
- **TMRD\_PHASE\_FAST\_OR\_SAME**: 進める、または同位相

機能:

A 相出力に対する B 相出力の位相関係を設定します。

戻り値:

なし

補足:

連動 PPG モードのみ有効です。A 相及び B 相出力は、タイマモード、連動タイマモード、PPG モードで切り替えできません。

### 20.2.3.13 TMRD\_EnableUpdateCMPReg

更新イネーブルフラグの設定

関数のプロトタイプ宣言:

void

TMRD\_EnableUpdateCMPReg(TSB\_TD\_TypeDef\* **TDx**,  
uint8\_t **UpdateCHx**)

引数:

**TDx**: TMRD ユニットを選択します。

**UpdateCHx**: 以下から更新イネーブルフラグを選択します。有効ビットの組み合わせが可能です。

- **TMRD\_UPDATE\_CH\_00**: TD0 の CP0/CP1/ CP2/ CP5 の更新イネーブルフラグ
- **TMRD\_UPDATE\_CH\_01**: TD0 の CP3/CP4 の更新イネーブルフラグ
- **TMRD\_UPDATE\_CH\_10**: TD1 の CP0/CP1/CP2 の更新イネーブルフラグ
- **TMRD\_UPDATE\_CH\_11**: TD1 の CP3/CP4 の更新イネーブルフラグ

機能:

更新イネーブルフラグを設定します。



関連機能の TMRD\_SetCMPRegWritePath() も合わせてご確認ください。

戻り値:

なし

補足:

連動 PPG では TMRD0 の有効イネーブルフラグ設定時、TMRD1 の有効イネーブルフラグが同時に設定されますので、再度 TMRD1 の設定を行わないでください。

## 20.2.3.14 TMRD\_SetDMAReq

DMA 要求許可設定(INTTDxCMP0)

関数のプロトタイプ宣言:

void

```
TMRD_SetDMAReq(TSB_TD_TypeDef* TDx,  
                TMRD_UNIT_Channel CHx,  
                FunctionalState NewState)
```

引数:

**TDx**: TMRD ユニットを選択します。

**CHx**: TMRD チャンネルを選択します。

**NewState**: DMA 要求の許可/禁止を選択します。

- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:

DMA 要求の許可/禁止を設定します。

戻り値:

なし

## 20.2.3.15 TMRD\_SetInitTiming

タイミング設定の初期化

関数のプロトタイプ宣言:

void

```
TMRD_SetInitTiming(TSB_TD_TypeDef* TDx,  
                   TMRD_UNIT_Channel CHx,  
                   TMRD_TimingTypeDef* TimingStruct)
```

**引数:**

**TDx:** TMRD ユニットを選択します。

**CHx:** TMRD チャンネルを選択します。

**TimingStruct:** タイミング設定の構造体です。詳細は"データ構造"を参照してください。

**機能:**

周波数タイミング、周期タイミング、trailing timing、位相のシフト量などの設定を行います。

**戻り値:**

なし

**補足:**

**TimingStruct** 構造体の各パラメータの設定可能値については、"データ構造"を参照してください。

## 20.2.3.16 TMRD\_ChangeTiming

タイミング値の変更

**関数のプロトタイプ宣言:**

void

```
TMRD_ChangeTiming(TSB_TD_TypeDef* TDx,  
                  uint8_t TimingType,  
                  uint16_t Timing)
```

**引数:**

**TDx:** TMRD ユニットを選択します。

**TimingType:** 以下から変更するタイミングの種類を選択します。

- **TMRD\_TIMING\_TD0\_CYCLE:** TMRD0 の周期設定 (TD0RG0)
- **TMRD\_TIMING\_A0\_LEADING:** 信号 a0 の leading timing (TD0RG1)
- **TMRD\_TIMING\_A0\_TRAILING:** 信号 a0 の trailing timing (TD0RG2)
- **TMRD\_TIMING\_A1\_LEADING:** 信号 a1 の leading timing (TD0RG3)
- **TMRD\_TIMING\_A1\_TRAILING:** 信号 a1 の trailing timing (TD0RG4)
- **TMRD\_TIMING\_PHASE\_SHIFT:** 位相のシフト量(TD0RG5)
- **TMRD\_TIMING\_TD1\_CYCLE:** TMRD1 の周期タイミング(TD1RG0)
- **TMRD\_TIMING\_B0\_LEADING:** 信号 b0 の leading timing (TD1RG1)
- **TMRD\_TIMING\_B0\_TRAILING:** 信号 b0 の trailing timing (TD1RG2)
- **TMRD\_TIMING\_B1\_LEADING:** 信号 b1 の leading timing (TD1RG3)
- **TMRD\_TIMING\_B1\_TRAILING:** 信号 b1 の trailing timing (TD1RG4)

**Timing:** タイミング設定値を設定します。データの範囲は 0x02~0x10000 です。

**機能:**

タイミング設定を変更します。

一部のタイミング設定を変更する場合に有効な API です。

**戻り値:**

なし

**補足:**

**Timing** 値について、各タイミングの設定範囲については"データ構造"を参照してください。

## 20.2.3.17 TMRD\_GetTiming

タイミング値の取得

**関数のプロトタイプ宣言:**

uint16\_t

TMRD\_GetTiming(TSB\_TD\_TypeDef\* **TDx**,  
uint8\_t **TimingType**)

**引数:**

**TDx:** TMRD ユニットを選択します。

**TimingType:** 以下から変更するタイミングの種類を選択します。

- **TMRD\_TIMING\_TD0\_CYCLE:** TMRD0 の周期設定 (TD0RG0)
- **TMRD\_TIMING\_A0\_LEADING:** 信号 a0 の leading timing (TD0RG1)
- **TMRD\_TIMING\_A0\_TRAILING:** 信号 a0 の trailing timing (TD0RG2)
- **TMRD\_TIMING\_A1\_LEADING:** 信号 a1 の leading timing (TD0RG3)
- **TMRD\_TIMING\_A1\_TRAILING:** 信号 a1 の trailing timing (TD0RG4)
- **TMRD\_TIMING\_PHASE\_SHIFT:** 位相のシフト量(TD0RG5)
- **TMRD\_TIMING\_TD1\_CYCLE:** TMRD1 の周期タイミング(TD1RG0)
- **TMRD\_TIMING\_B0\_LEADING:** 信号 b0 の leading timing (TD1RG1)
- **TMRD\_TIMING\_B0\_TRAILING:** 信号 b0 の trailing timing (TD1RG2)
- **TMRD\_TIMING\_B1\_LEADING:** 信号 b1 の leading timing (TD1RG3)
- **TMRD\_TIMING\_B1\_TRAILING:** 信号 b1 の trailing timing (TD1RG4)

**機能:**

タイミング設定値を取得します。

**戻り値:**

タイミング設定値

## 20.2.3.18 TMRD\_SetBitModulationCycle

1bit モジュレーションの周期設定

関数のプロトタイプ宣言:

void

TMRD\_SetBitModulationCycle (TSB\_TD\_TypeDef\* ***TDx***,  
uint8\_t ***PPGChannel***,  
uint8\_t ***BitModCycle***)

引数:

***TDx***: TMRD ユニットを選択します。

***PPGChannel***: 以下から PPG 出力チャンネルを選択します。

- **TMRD\_PPG\_CHANNEL\_A0**: PPG 出力信号 a0
- **TMRD\_PPG\_CHANNEL\_A1**: PPG 出力信号 a1
- **TMRD\_PPG\_CHANNEL\_B0**: PPG 出力信号 b0
- **TMRD\_PPG\_CHANNEL\_B1**: PPG 出力信号 b1

***BitModCycle***: 以下から 1bit モジュレーションの周期を選択します。

- **TMRD\_1BITMOD\_CYCLE\_NONE**: 1bit モジュレーション機能無し
- **TMRD\_1BITMOD\_CYCLE\_2TIMES**: (CP00/CP10 で決定される周期) x 2
- **TMRD\_1BITMOD\_CYCLE\_4TIMES**: (CP00/CP10 で決定される周期) x 4
- **TMRD\_1BITMOD\_CYCLE\_8TIMES**: (CP00/CP10 で決定される周期) x 8
- **TMRD\_1BITMOD\_CYCLE\_16TIMES**: (CP00/CP10 で決定される周期) x 16

機能:

PPG モードおよび連動 PPG モードにおける出力信号の 1bit モジュレーションの周期を設定します。

戻り値:

なし

補足:

タイマーモードと連動モードでは、TMRD\_SetBitModulationCycle()関数は無視されます。

## 20.2.3.19 TMRD\_SetBitModUpdateTiming

1bit モジュレーションの更新タイミング設定

関数のプロトタイプ宣言:

void

TMRD\_SetBitModUpdateTiming (TSB\_TD\_TypeDef\* ***TDx***,  
uint8\_t ***PPGChannel***,

FunctionalState **NewState**)

引数:

**TDx**: TMRD ユニットを選択します。

**PPGChannel**: 以下から PPG 出力チャンネルを選択します。

- **TMRD\_PPG\_CHANNEL\_A0**: PPG 出力信号 a0
- **TMRD\_PPG\_CHANNEL\_A1**: PPG 出力信号 a1
- **TMRD\_PPG\_CHANNEL\_B0**: PPG 出力信号 b0
- **TMRD\_PPG\_CHANNEL\_B1**: PPG 出力信号 b1

**NewState**: 各 PPG チャンネルの更新タイミングを選択します。

- **DISABLE**: 1bit モジュレーション周期ごと
- **ENABLE**: CPxx の一致検出

機能:

1bit モジュレーションの更新タイミングを設定します。

戻り値:

なし

補足:

タイマーモードまたは連動モードにおいて、TMRD\_SetBitModUpdateTiming()関数は、無視されます。

## 20.2.4 データ構造

### 20.2.4.1 TMRD\_TimingTypeDef

メンバ:

Uint32\_t

**Cycle**: TMRD0/1 の周期 (RG0)

uint16\_t

**LeadingTiming0**: TMRD0/1 の信号 a0/b0 の leading timing (RG1)

uint16\_t

**TrailingTiming0**: TMRD0/1 の信号 a0/b0 の trailing timing (RG2)

uint8\_t

**BitModulationRate0**: TMRD0/1 の ch1 の 1bit モジュレーションの周期

uint16\_t

**LeadingTiming1**: TMRD0/1 の信号 a1/b1 の leading timing (RG3)

uint16\_t

**TrailingTiming1:** TMRD0/1 の信号 a1/b1 の trailing timing (RG4)

uint16\_t

**PhaseShiftTiming:** 位相のシフト量 (RG5, TMRD0 のみ)

uint8\_t

**BitModulationRate1:** TMRD0/1 の ch0 の 1bit モジュレーションの周期です。

補足:

モードごとの各パラメータは、下表を参照してください。

Timer Unit	Compare register	16-bit interval timer	
		<TDCLE> = "0"	<TDCLE> = "1"
TMRD0	TD0CP0	$0x0000 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$
	TD0CP1	$0x0000 \leq \text{CPRG1}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG1}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP2	$0x0000 \leq \text{CPRG2}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP3	$0x0000 \leq \text{CPRG3}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG3}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP4	$0x0000 \leq \text{CPRG4}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP5	$0x0000 \leq \text{CPRG5}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG5}[15:0] \leq \text{CPRG0}[15:0]$
TMRD1	TD1CP0	$0x0000 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$
	TD1CP1	$0x0000 \leq \text{CPRG1}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG1}[15:0] \leq \text{CPRG0}[15:0]$
	TD1CP2	$0x0000 \leq \text{CPRG2}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$
	TD1CP3	$0x0000 \leq \text{CPRG3}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG3}[15:0] \leq \text{CPRG0}[15:0]$
	TD1CP4	$0x0000 \leq \text{CPRG4}[15:0] \leq 0xFFFF$	$0x0000 \leq \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$

タイマモードにおけるコンペアレジスタの設定範囲

Timer Unit	Compare register	16-bit programmable pulse generation	
		PPG	Interlock PPG
TMRD0	TD0CP0	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$
	TD0CP1	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$
	TD0CP2	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP3	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$
	TD0CP4	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$
	TD0CP5	don't care	$0x0000 \leq \text{CPRG5}[15:0] < (\text{CPRG0}[15:0] + 2)$
TMRD1	TD1CP0	$0x0001 \leq \text{CPRG0}[15:0] \leq 0xFFFF$	don't care
	TD1CP1	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$	$0x0000 \leq \text{CPRG1}[15:0] < \text{CPRG2}[15:0]$
	TD1CP2	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG1}[15:0] < \text{CPRG2}[15:0] \leq \text{TD0CP0} < \text{CPRG0}[15:0]$
	TD1CP3	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$	$0x0000 \leq \text{CPRG3}[15:0] < \text{CPRG4}[15:0]$
	TD1CP4	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{CPRG0}[15:0]$	$\text{CPRG3}[15:0] < \text{CPRG4}[15:0] \leq \text{TD0CP0} < \text{CPRG0}[15:0]$

PPG モードにおけるコンペアレジスタの設定範囲

## 21. UART

### 21.1 概要

本デバイスのシリアル I/O チャンネルは、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/TX04\_Periph\_Driver/src/tmpm440\_uart.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_uart.h

### 21.2 API 関数

#### 21.2.1 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\_FIFOConfig(TSB\_SC\_TypeDef \* **UARTx**, FunctionalState **NewState**)
- ◆ void UART\_SetFIFOTransferMode(TSB\_SC\_TypeDef \* **UARTx**, uint32\_t **TransferMode**)
- ◆ void UART\_TRxAutoDisable(TSB\_SC\_TypeDef \* **UARTx**,  
UART\_TRxAutoDisable **TRxAutoDisable**);
- ◆ void UART\_RxFIFOINTCtrl(TSB\_SC\_TypeDef \* **UARTx**, FunctionalState **NewState**)
- ◆ void UART\_TxFIFOINTCtrl(TSB\_SC\_TypeDef \* **UARTx**, FunctionalState **NewState**)
- ◆ void UART\_RxFIFOByteSel(TSB\_SC\_TypeDef \* **UARTx**, uint32\_t **BytesUsed**)
- ◆ void UART\_RxFIFOFillLevel(TSB\_SC\_TypeDef \* **UARTx**,uint32\_t **RxFIFOLevel**)
- ◆ void UART\_RxFIFOINTSel(TSB\_SC\_TypeDef \* **UARTx**, uint32\_t **RxINTCondition**)



- ◆ void UART\_RxFIFOClear(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ void UART\_TxFIFOFillLevel(TSB\_SC\_TypeDef \* **UARTx**, uint32\_t **TxFIFOLevel**);
- ◆ void UART\_TxFIFOINTSel(TSB\_SC\_TypeDef \* **UARTx**, uint32\_t **TxINTCondition**);
- ◆ void UART\_TxFIFOClear(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ void UART\_TxBufferClear(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ uint32\_t UART\_GetRxFIFOFillLevelStatus(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ uint32\_t UART\_GetRxFIFOOverRunStatus(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ uint32\_t UART\_GetTxFIFOFillLevelStatus(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ uint32\_t UART\_GetTxFIFOUnderRunStatus(TSB\_SC\_TypeDef \* **UARTx**);
- ◆ void UART\_SetRxDMAReq (TSB\_SC\_TypeDef\* **UARTx**, FunctionalState **NewState**);
- ◆ void UART\_SetTxDMAReq (TSB\_SC\_TypeDef\* **UARTx**, FunctionalState **NewState**);
- ◆ void UART\_SetInputClock(TSB\_SC\_TypeDef \* **UARTx**, uint32\_t **clock**);
- ◆ void SIO\_SetInputClock(TSB\_SC\_TypeDef \* **SIOx**, uint32\_t **Clock**);
- ◆ void SIO\_Enable(TSB\_SC\_TypeDef \* **SIOx**);
- ◆ void SIO\_Disable(TSB\_SC\_TypeDef \* **SIOx**);
- ◆ void SIO\_Init(TSB\_SC\_TypeDef \* **SIOx**, uint32\_t **IOClkSel**, SIO\_InitTypeDef \* **InitStruct**);
- ◆ uint8\_t SIO\_GetRxData(TSB\_SC\_TypeDef \* **SIOx**);
- ◆ void SIO\_SetTxData(TSB\_SC\_TypeDef \* **SIOx**, uint8\_t **Data**);

## 21.2.2 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) 初期化と設定:  
UART\_Enable(), UART\_Disable(), UART\_SetInputClock(), UART\_Init(),  
UART\_DefaultConfig(), SIO\_Enable(), SIO\_Disable(), SIO\_SetInputClock(), 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\_RxFIFOINTCtrl(),  
UART\_TxFIFOINTCtrl(), UART\_RxFIFOByteSel(), UART\_RxFIFOFillLevel(),  
UART\_RxFIFOINTSel(), UART\_RxFIFOClear(), UART\_TxFIFOFillLevel(),  
UART\_TxFIFOINTSel(), UART\_TxFIFOClear(), UART\_TxBufferClear(),  
UART\_GetRxFIFOFillLevelStatus(), UART\_GetRxFIFOOverRunStatus(),  
UART\_GetTxFIFOFillLevelStatus(), UART\_GetTxFIFOUnderRunStatus()

## 21.2.3 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, UART2, UART3, UART4, UART5**

また、“TSB\_SC\_TypeDef\* **SIOx**” は、以下から選択してください。  
**SIO0, SIO1, SIO2, SIO3, SIO4, SIO5**

## 21.2.3.1 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

## 21.2.3.2 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

## 21.2.3.3 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

## 21.2.3.4 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

## 21.2.3.5 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

## 21.2.3.6 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 (シリアルチャンネル) 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。

## 21.2.3.7 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 (シリアルチャネル) 割り込み関数の中で実行してください。

**戻り値:**

なし

## 21.2.3.8 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

## 21.2.3.9 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

## 21.2.3.10 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

## 21.2.3.11 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

## 21.2.3.12 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

## 21.2.3.13 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

## 21.2.3.14 UART\_SetInputClock

入力クロックの設定

関数のプロトタイプ宣言:

void

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

引数:

**UARTx**: UART チャンネルを指定します。

**Clock**: 以下から、プリスケアラの入力クロックを選択します。

- **0** :  $\Phi T0/2$
- **1** :  $\Phi T0$



**機能:**

プリスケアラの入力クロックを選択します。

**戻り値:**

なし

## 21.2.3.15 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

## 21.2.3.16 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

## 21.2.3.17 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\_RXTXCNT\_NONE**: なし
- **UART\_RXTXCNT\_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

## 21.2.3.18 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

### 21.2.3.19 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

### 21.2.3.20 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

## 21.2.3.21 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

## 21.2.3.22 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

## 21.2.3.23 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

## 21.2.3.24 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

## 21.2.3.25 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

## 21.2.3.26 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

## 21.2.3.27 UART\_TxBufferClear

送信バッファクリア

**関数のプロトタイプ宣言:**

void

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

**引数:**

**UARTx:** UART チャンネルを指定します。

**機能:**

送信バッファをクリアします。

**戻り値:**

なし

## 21.2.3.28 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

### 21.2.3.29 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

### 21.2.3.30 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

### 21.2.3.31 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

### 21.2.3.32 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

## 21.2.3.33 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

## 21.2.3.34 SIO\_SetInputClock

入力クロックの設定

**関数のプロトタイプ宣言:**

void

SIO\_SetInputClock (TSB\_SC\_TypeDef \* SIOx,  
uint32\_t Clock)

**引数:**

**SIOx**: SIO チャンネルを指定します。

**Clock**: 以下から、プリスケアラの入力クロックを選択します。

- **SIO\_CLOCK\_T0\_HALF** :  $\Phi T0/2$
- **SIO\_CLOCK\_T0** :  $\Phi T0$

**機能:**

プリスケアラの入力クロックを選択します。

**戻り値:**

なし

## 21.2.3.35 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)

## 21.2.3.36 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

## 21.2.3.37 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:** ボ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:**

ボーレート、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

## 21.2.4 データ構造

### 21.2.4.1 UART\_InitTypeDef

**メンバ:**

uint32\_t

**BaudRate** :UART 通信ボーレートを 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。

## 21.2.4.2 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\_FULDPX: 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

## 22. WDT

### 22.1 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\\TX04\_Periph\_Driver\\src\\tmpm440\_wdt.c

\\Libraries\\TX04\_Periph\_Driver\\inc\\tmpm440\_wdt.h

### 22.2 API 関数

#### 22.2.1 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)

#### 22.2.2 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(), WDT\_SetOverflowOutput(), WDT\_Init(), WDT\_Enable(),  
WDT\_Disable(), WDT\_WriteClearCode()

- 2) IDLE モード時の開始・停止:

WDT\_SetIdleMode()

#### 22.2.3 関数仕様

##### 22.2.3.1 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

### 22.2.3.2 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



## 22.2.3.3 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 割り込み要求が発生します。
- **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

## 22.2.3.4 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

## 22.2.3.5 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

## 22.2.3.6 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

## 22.2.3.7 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

## 22.2.4 データ構造

### 22.2.4.1 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:  $2^{15}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_17:  $2^{17}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_19:  $2^{19}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_21:  $2^{21}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_23:  $2^{23}/\text{fsys}$
- WDT\_DETECT\_TIME\_EXP\_25:  $2^{25}/\text{fsys}$

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。

## 23. PSC

### 23.1 概要

本デバイスは、モータ等のサーボ制御やさまざまな機器のシーケンス制御を行う演算器として PSC(Programmable Servo/Sequence Controller) を 1 ユニット内蔵しています。

PSC ドライバは API は、ドライバ関数を使用して PSC の設定を行う機能を提供します。

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/TX04\_Periph\_Driver/src/tmpm440\_psc.c

/Libraries/TX04\_Periph\_Driver/inc/tmpm440\_psc.h

### 23.2 API 関数

#### 23.2.1 関数一覧

- ◆ void PSC\_SetAccumulator(uint32\_t **UA0**);
- ◆ uint32\_t PSC\_GetAccumulator(void);
- ◆ void PSC\_SetMultiPReg(uint32\_t **UM0**);
- ◆ uint32\_t PSC\_GetMultiPReg(void);
- ◆ void PSC\_SetShiftCountReg(uint32\_t **UM1**);
- ◆ uint32\_t PSC\_GetShiftCountReg(void);
- ◆ void PSC\_SetUpperLimitReg(uint32\_t **UL0**);
- ◆ uint32\_t PSC\_GetUpperLimitReg(void);
- ◆ void PSC\_SetLowerLimitReg(uint32\_t **UL1**);
- ◆ uint32\_t PSC\_GetLowerLimitReg(void);
- ◆ void PSC\_SetAddSubReg0(uint32\_t **UR0**);
- ◆ uint32\_t PSC\_GetAddSubReg0(void);
- ◆ void PSC\_SetAddSubReg1(uint32\_t **UR1**);
- ◆ uint32\_t PSC\_GetAddSubReg1(void);
- ◆ void PSC\_SetArithmeticParameter(PSC\_ArithmeticPARAM **Param**);
- ◆ PSC\_ArithmeticPARAM PSC\_GetArithmeticParameter(void);
- ◆ void PSC\_SetAddrPointer0(uint32\_t **AP0**);
- ◆ uint32\_t PSC\_GetAddrPointer0(void);
- ◆ void PSC\_SetAddrPointer1(uint32\_t **AP1**);
- ◆ uint32\_t PSC\_GetAddrPointer1(void);
- ◆ void PSC\_SetAddrPointer2(uint32\_t **AP2**);
- ◆ uint32\_t PSC\_GetAddrPointer2(void);
- ◆ void PSC\_SetAddrPointer3(uint32\_t **AP3**);
- ◆ uint32\_t PSC\_GetAddrPointer3(void);

- ◆ void PSC\_SetBreakPointer(uint32\_t **BR0**);
- ◆ uint32\_t PSC\_GetBreakPointer(void);
- ◆ void PSC\_SetProgramPointer(uint32\_t **PG0**);
- ◆ uint32\_t PSC\_GetProgramPointer(void);
- ◆ void PSC\_SetRepetProcVectPointer(uint32\_t **VG0**);
- ◆ uint32\_t PSC\_GetRepetProcVectPointer(void);
- ◆ void PSC\_SetBreakFUNC(FunctionalState **NewState**);
- ◆ void PSC\_SetStepFUNC(FunctionalState **NewState**);
- ◆ void PSC\_SetExecutionFUNC(FunctionalState **NewState**);
- ◆ void PSC\_SetArithmeticResult(PSC\_ArithmeticResult **ArithmeticResult**);
- ◆ PSC\_ArithmeticResult PSC\_GetArithmeticResult(void);
- ◆ void PSC\_SetTriggerEventFUNC(uint32\_t **Event**, FunctionalState **NewState**);
- ◆ FunctionalState PSC\_GetTriggerEventState(uint32\_t **Event**);
- ◆ void PSC\_SetTriggerEventEdge(uint32\_t **Event**, PSC\_TriggerEventEdge **Edge**);
- ◆ PSC\_TriggerEventEdge PSC\_GetTriggerEventEdge(uint32\_t **Event**);
- ◆ PSC\_OverRunFlag PSC\_GetOverRunFlag(void);
- ◆ PSC\_StartupFlag PSC\_GetStartupFlag(void);
- ◆ void PSC\_ClearEventOverrunFlag(uint32\_t **Event**);
- ◆ void PSC\_SetStartup(uint32\_t **Event**);
- ◆ void PSC\_SetOutputData(uint8\_t **Data**);
- ◆ void PSC\_SetOutputDataBit(uint8\_t **Bit\_x**, uint8\_t **BitValue**);
- ◆ void PSC\_SetDataOutputFUNC(uint8\_t **Bit\_x**, FunctionalState **NewState**);
- ◆ FunctionalState PSC\_GetDataOutputState(uint8\_t **Bit\_x**);
- ◆ uint8\_t PSC\_GetInputData(void);
- ◆ uint8\_t PSC\_GetInputDataBit(uint8\_t **Bit\_x**);
- ◆ PSC\_ExecutionState PSC\_GetExecutionState(void);

## 23.2.2 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) PSC レジスタの設定:

PSC\_SetAccumulator(), PSC\_SetMultiplReg(), PSC\_SetShiftCountReg(),  
PSC\_SetUpperLimitReg(), PSC\_SetLowerLimitReg(), PSC\_SetAddSubReg0(),  
PSC\_SetAddSubReg1(), PSC\_SetArithmeticParameter(), PSC\_SetAddrPointer0(),  
PSC\_SetAddrPointer1(), PSC\_SetAddrPointer2(), PSC\_SetAddrPointer3(),  
PSC\_SetBreakPointer(), PSC\_SetProgramPointer(), PSC\_SetRepetProcVectPointer(),  
PSC\_SetArithmeticResult()

### 2) PSC レジスタの取得

PSC\_GetAccumulator(), PSC\_GetMultiplReg(), PSC\_GetShiftCountReg(),  
PSC\_GetUpperLimitReg(), PSC\_GetLowerLimitReg(), PSC\_GetAddSubReg0(),  
PSC\_GetAddSubReg1(), PSC\_GetAddrPointer0(), PSC\_GetAddrPointer1(),

PSC\_GetAddrPointer2(), PSC\_GetAddrPointer3(), PSC\_GetBreakPointer(),  
PSC\_GetProgramPointer(), PSC\_GetRepetProcVectPointer(),  
PSC\_GetArithmeticParameter(), PSC\_GetArithmeticResult()

### 3) その他

PSC\_SetBreakFUNC(), PSC\_SetStepFUNC(), PSC\_SetExecutionFUNC(),  
PSC\_SetTriggerEventFUNC(), PSC\_GetTriggerEventState(),  
PSC\_SetTriggerEventEdge(), PSC\_GetTriggerEventEdge(), PSC\_GetOverRunFlag(),  
PSC\_GetStartupFlag(), PSC\_ClearEventOverrunFlag(), PSC\_SetStartup(),  
PSC\_SetOutputData(), PSC\_SetOutputDataBit(), PSC\_SetDataOutputFUNC(),  
PSC\_GetDataOutputState(), PSC\_GetInputData(),  
PSC\_GetInputDataBit(), PSC\_GetExecutionState().

## 23.2.3 関数仕様

### 23.2.3.1 PSC\_SetAccumulator

アキュムレータレジスタの設定

**関数のプロトタイプ宣言:**

void

PSC\_SetAccumulator(uint32\_t **UA0**)

**引数:**

**UA0:** アキュムレータレジスタ(UA0)に設定する符号を除いた値を設定します。

**機能:**

アキュムレータレジスタ(UA0)に符号を除いた値を設定します。

**戻り値:**

なし

### 23.2.3.2 PSC\_GetAccumulator

アキュムレータレジスタ(UA0)の取得

**関数のプロトタイプ宣言:**

uint32\_t

PSC\_GetAccumulator(void)

**引数:**

なし

**機能:**

アキュムレータレジスタ(UA0)から符号を除いた値を取得します。

戻り値:

アキュムレータレジスタ(UA0)から符号を除いた値

### 23.2.3.3 PSC\_SetMultiReg

乗数レジスタ(UM0)の設定

関数のプロトタイプ宣言:

void

PSC\_SetMultiReg(uint32\_t *UM0*)

引数:

*UM0*: 乗数レジスタ(UM0)に設定する符号を除いた値を設定します。

機能:

乗数レジスタ(UM0)に、符号を除いた値を設定します。

戻り値:

なし

### 23.2.3.4 PSC\_GetMultiReg

乗数レジスタ(UM0)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetMultiReg(void)

引数:

なし

機能:

乗数レジスタ(UM0)から符号を除いた値を取得します。

戻り値:

乗数レジスタ(UM0)から符号を除いた値

### 23.2.3.5 PSC\_SetShiftCountReg

シフト数レジスタ(UM1)の設定

関数のプロトタイプ宣言:

void  
PSC\_SetShiftCountReg(uint32\_t **UM1**)

**引数:**

**UM1:** シフト数レジスタ(UM1)に設定する符号を除いた値を設定します。

**機能:**

シフト数レジスタ(UM1)に符号を除いた値を設定します。

**戻り値:**

なし

### 23.2.3.6 PSC\_GetShiftCountReg

シフト数レジスタ(UM1)の取得

**関数のプロトタイプ宣言:**

uint32\_t  
PSC\_GetShiftCountReg(void)

**引数:**

なし

**機能:**

シフト数レジスタ(UM1)から符号を除いた値を取得します。

**戻り値:**

シフト数レジスタ(UM1)から符号を除いた値

### 23.2.3.7 PSC\_SetUpperLimitReg

リミット上限値レジスタ(UL0)の設定

**関数のプロトタイプ宣言:**

void  
PSC\_SetUpperLimitReg(uint32\_t **UL0**)

**引数:**

**UL0:** リミット上限値レジスタ(UL0)に設定する符号を除いた値を指定します。

**機能:**

リミット上限値レジスタ(UL0)に値を設定します。



戻り値:

なし

## 23.2.3.8 PSC\_GetUpperLimitReg

リミット上限値レジスタ(UL0)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetUpperLimitReg(void)

引数:

なし

機能:

リミット上限値レジスタ(UL0)の符号を除いた値を取得します。

戻り値:

リミット上限値レジスタ(UL0)から符号を除いた値

## 23.2.3.9 PSC\_SetLowerLimitReg

リミット下限値レジスタ(UL1)の設定

関数のプロトタイプ宣言:

void

PSC\_SetLowerLimitReg(uint32\_t **UL1**)

引数:

**UL1**: リミット加減値レジスタ(UL1)に設定する符号を除いた値を指定します。

機能:

リミット上限値レジスタ(UL1)に符号を除いた値を設定します。

戻り値:

なし

## 23.2.3.10 PSC\_GetLowerLimitReg

リミット下限値レジスタ(UL1)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetLowerLimitReg(void)

引数:

なし

機能:

リミット下限値レジスタ(UL1)から符号を除いた値を取得します。

戻り値:

リミット下限値レジスタ(UL1)から符号を除いた値

### 23.2.3.11 PSC\_SetAddSubReg0

加減算値レジスタ(UR0)の設定

関数のプロトタイプ宣言:

void

PSC\_SetAddSubReg0(uint32\_t *UR0*)

引数:

*UR0*: 加減算値レジスタ(UR0)に設定する符号を除いた値を指定します。

機能:

加減算値レジスタ(UR0)に符号を除いた値を設定します。

戻り値:

なし

### 23.2.3.12 PSC\_GetAddSubReg0

加減算値レジスタ(UR0)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetAddSubReg0(void)

引数:

なし

機能:

加減算値レジスタ(UR0)から符号を除いた値を取得します。

戻り値:

加減算値レジスタ(UR0)から符号を除いた値

### 23.2.3.13 PSC\_SetAddSubReg1

加減算値レジスタ(UR1)の設定

**関数のプロトタイプ宣言:**

```
void  
PSC_SetAddSubReg1(uint32_t UR1)
```

**引数:**

**UR1:** 加減算値レジスタ(UR1)に設定する符号を除いた値を指定します。

**機能:**

加減算値レジスタ(UR1)に符号を除いた値を指定します。

**戻り値:**

なし

### 23.2.3.14 PSC\_GetAddSubReg1

加減算値レジスタ(UR1)の取得

**関数のプロトタイプ宣言:**

```
uint32_t  
PSC_GetAddSubReg1 (void)
```

**引数:**

なし

**機能:**

加減算値レジスタ(UR1)から符号を除いた値を取得します。

**戻り値:**

加減算値レジスタ(UR1)から符号を除いた値

### 23.2.3.15 PSC\_SetArithmeticParameter

演算パラメータ符号レジスタの設定

**関数のプロトタイプ宣言:**

```
void  
PSC_SetArithmeticParameter (PSC_ArithmeticPARAM Param)
```

**引数:**

**Param:** 演算パラメータ符号レジスタに設定する共用体を指定します。(詳細は"データ構造"を参照)

**機能:**

演算パラメータ符号レジスタに値を設定します。

**戻り値:**

なし

### 23.2.3.16 PSC\_GetArithmeticParameter

演算パラメータ符号レジスタの取得

**関数のプロトタイプ宣言:**

PSC\_ArithmeticPARAM

PSC\_GetArithmeticParameter (void)

**引数:**

なし

**機能:**

演算パラメータ符号レジスタの値を取得します。

**戻り値:**

演算パラメータ符号レジスタの値を格納した共用体  
(詳細は"データ構造"を参照)。

### 23.2.3.17 PSC\_SetAddrPointer0

アドレスポインタ 0(AP0)の設定

**関数のプロトタイプ宣言:**

void

PSC\_SetAddrPointer0 (uint32\_t **AP0**)

**引数:**

**AP0:** アドレスポインタ 0(AP0)に設定する値を指定します。

**機能:**

アドレスポインタ 0(AP0)に値を設定します。

戻り値:

なし

## 23.2.3.18 PSC\_GetAddrPointer0

アドレスポインタ 0(AP0)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetAddrPointer0 (void)

引数:

なし

機能:

アドレスポインタ 0(AP0)から値を取得します。

戻り値:

アドレスポインタ 0(AP0)から取得した値

## 23.2.3.19 PSC\_SetAddrPointer1

アドレスポインタ 1(AP1)の設定

関数のプロトタイプ宣言:

void

PSC\_SetAddrPointer1 (uint32\_t **AP1**)

引数:

**AP1**: アドレスポインタ 1(AP1)に設定する値を指定します。

機能:

アドレスポインタ 1(AP1)に値を設定します。

戻り値:

なし

## 23.2.3.20 PSC\_GetAddrPointer1

アドレスポインタ 1(AP1)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetAddrPointer1 (void)

引数:

なし

機能:

アドレスポインタ 1(AP1)から値を取得します。

戻り値:

アドレスポインタ 1(AP1)から取得した値

## 23.2.3.21 PSC\_SetAddrPointer2

アドレスポインタ 2(AP2)の設定

関数のプロトタイプ宣言:

void

PSC\_SetAddrPointer2 (uint32\_t **AP2**)

引数:

**AP2**: アドレスポインタ 2(AP2)に設定する値を指定します。

機能:

アドレスポインタ 2(AP2)に値を設定します。

戻り値:

なし

## 23.2.3.22 PSC\_GetAddrPointer2

アドレスポインタ 2(AP2)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetAddrPointer2 (void)

引数:

なし

機能:

アドレスポインタ 2(AP2)から値を取得します。

戻り値:

アドレスポインタ 2(AP2)から取得した値

## 23.2.3.23 PSC\_SetAddrPointer3

アドレスポインタ 3(AP3)の設定

関数のプロトタイプ宣言:

void

PSC\_SetAddrPointer3 (uint32\_t **AP3**)

引数:

**AP3**: アドレスポインタ 3(AP3)に設定する値を指定します。

機能:

アドレスポインタ 3(AP3)に値を設定します。

戻り値:

なし

## 23.2.3.24 PSC\_GetAddrPointer3

アドレスポインタ 3(AP3)の取得

関数のプロトタイプ宣言:

uint32\_t

PSC\_GetAddrPointer3 (void)

引数:

なし

機能:

アドレスポインタ 3(AP3)の値を取得します。

戻り値:

アドレスポインタ 3(AP3)から取得した値

## 23.2.3.25 PSC\_SetBreakPointer

ブレークポインタ(BR0)の設定

関数のプロトタイプ宣言:

void

PSC\_SetBreakPointer (uint32\_t **BR0**)

**引数:**

**BR0:** ブレークポインタ(BR0)に設定するアドレス値を指定します。

**機能:**

ブレークポインタ(BR0)にアドレス値を設定します。

**戻り値:**

なし

### 23.2.3.26 PSC\_GetBreakPointer

ブレークポインタ(BR0)の取得

**関数のプロトタイプ宣言:**

uint32\_t

PSC\_GetBreakPointer (void)

**引数:**

なし

**機能:**

ブレークポインタ(BR0)のアドレス値を取得します。

**戻り値:**

ブレークポインタ(BR0)から取得したアドレス値

### 23.2.3.27 PSC\_SetProgramPointer

プログラムポインタ(PG0)の設定

**関数のプロトタイプ宣言:**

void

PSC\_SetProgramPointer (uint32\_t **PG0**)

**引数:**

**PG0:** プログラムポインタ(PG0)に設定する値を指定します。

**機能:**

プログラムポインタ(PG0)に値を設定します。

**戻り値:**

なし



## 23.2.3.28 PSC\_GetProgramPointer

プログラムポインタ(PG0)の取得

**関数のプロトタイプ宣言:**

uint32\_t

PSC\_GetProgramPointer (void)

**引数:**

なし

**機能:**

プログラムポインタ(PG0)から値を取得します。

**戻り値:**

プログラムポインタ(PG0)から取得した値

## 23.2.3.29 PSC\_SetRepetProcVectPointer

繰り返し処理ベクトルポインタ(VG0)の設定

**関数のプロトタイプ宣言:**

void

PSC\_SetRepetProcVectPointer (uint32\_t **VG0**)

**引数:**

**VG0:** 繰り返し処理ベクトルポインタ(VG0)に設定する値を指定します。

**機能:**

繰り返し処理ベクトルポインタ(VG0)に値を設定します。

**戻り値:**

なし

## 23.2.3.30 PSC\_GetRepetProcVectPointer

繰り返し処理ベクトルポインタ(VG0)の取得

**関数のプロトタイプ宣言:**

uint32\_t

PSC\_GetRepetProcVectPointer (void)

**引数:**

なし

**機能:**

繰り返し処理ベクトルポインタ(VG0)から値を取得します。

**戻り値:**

繰り返し処理ベクトルポインタ(VG0)から取得した値

## 23.2.3.31 PSC\_SetBreakFUNC

ブレーク機能イネーブル

**関数のプロトタイプ宣言:**

void

PSC\_SetBreakFUNC (FunctionalState **NewState**)

**引数:**

**NewState:** ブレーク機能の ON/OFF を選択します。

- **DISABLE:** ブレーク機能OFF
- **ENABLE:** ブレーク機能ON

**機能:**

ブレーク機能の ON/OFF を選択します。

**戻り値:**

なし

## 23.2.3.32 PSC\_SetStepFUNC

ステップ機能イネーブル

**関数のプロトタイプ宣言:**

void

PSC\_SetStepFUNC (FunctionalState **NewState**)

**引数:**

**NewState:** ステップ機能の ON/OFF を選択します。

- **DISABLE:** ステップ機能OFF
- **ENABLE:** ステップ機能ON

**機能:**

ステップ機能の ON/OFF を選択します。

戻り値:  
なし

## 23.2.3.33 PSC\_SetExecutionFUNC

プログラムの実行/停止

関数のプロトタイプ宣言:

void

PSC\_SetExecutionFUNC (FunctionalState **NewState**)

引数:

**NewState**: プログラムの実行/停止を選択します。

- **DISABLE**: プログラム停止
- **ENABLE**: プログラム実行

機能:

プログラムの実行/停止を選択します。

戻り値:  
なし

## 23.2.3.34 PSC\_SetArithmeticResult

演算結果フラグの設定

関数のプロトタイプ宣言:

void

PSC\_SetArithmeticResult (PSC\_ArithmeticResult **ArithmeticResult**)

引数:

**ArithmeticResult**: 演算結果フラグに設定する値を共用体で指定します。  
(詳細は"データ構造"を参照)。

機能:

演算結果フラグに値を設定します。

戻り値:  
なし

## 23.2.3.35 PSC\_GetArithmeticResult

演算結果フラグの取得

**関数のプロトタイプ宣言:**

PSC\_ArithmeticResult

PSC\_GetArithmeticResult (void)

**引数:**

なし

**機能:**

演算結果フラグから値を取得します。

**戻り値:**

演算結果フラグから取得した値を格納した共用体  
(詳細は”データ構造”を参照)).

## 23.2.3.36 PSC\_SetTriggerEventFUNC

PSC 起動要因の設定

**関数のプロトタイプ宣言:**

void

PSC\_SetTriggerEventFUNC (uint32\_t **Event**, FunctionalState **NewState**)

**引数:**

**Event:** PSC 起動要因番号を選択します。

- PSC\_TRIGGER\_EVENT\_0: 起動要因0
- PSC\_TRIGGER\_EVENT\_1: 起動要因1
- PSC\_TRIGGER\_EVENT\_2: 起動要因2
- PSC\_TRIGGER\_EVENT\_3: 起動要因3
- PSC\_TRIGGER\_EVENT\_4: 起動要因4
- PSC\_TRIGGER\_EVENT\_5: 起動要因5
- PSC\_TRIGGER\_EVENT\_6: 起動要因6
- PSC\_TRIGGER\_EVENT\_7: 起動要因7
- PSC\_TRIGGER\_EVENT\_8: 起動要因8
- PSC\_TRIGGER\_EVENT\_9: 起動要因9
- PSC\_TRIGGER\_EVENT\_10: 起動要因10
- PSC\_TRIGGER\_EVENT\_11: 起動要因11
- PSC\_TRIGGER\_EVENT\_12: 起動要因12
- PSC\_TRIGGER\_EVENT\_13: 起動要因13
- PSC\_TRIGGER\_EVENT\_14: 起動要因14
- PSC\_TRIGGER\_EVENT\_15: 起動要因15

- **PSC\_TRIGGER\_EVENT\_ALL**: すべての起動要因

**NewState**: 起動要因の許可/禁止を選択します。

- **ENABLE**: 実行イネーブル
- **DISABLE**: 実行ディゼーブル

**機能:**

PSC 起動要因の許可/禁止を設定します。

**戻り値:**

なし

### 23.2.3.37 PSC\_GetTriggerEventState

PSC 起動要因の状態取得

**関数のプロトタイプ宣言:**

FunctionalState

PSC\_GetTriggerEventState (uint32\_t **Event**)

**引数:**

**Event**: PSC 起動要因番号を選択します。

- **PSC\_TRIGGER\_EVENT\_0**: 起動要因0
- **PSC\_TRIGGER\_EVENT\_1**: 起動要因1
- **PSC\_TRIGGER\_EVENT\_2**: 起動要因2
- **PSC\_TRIGGER\_EVENT\_3**: 起動要因3
- **PSC\_TRIGGER\_EVENT\_4**: 起動要因4
- **PSC\_TRIGGER\_EVENT\_5**: 起動要因5
- **PSC\_TRIGGER\_EVENT\_6**: 起動要因6
- **PSC\_TRIGGER\_EVENT\_7**: 起動要因7
- **PSC\_TRIGGER\_EVENT\_8**: 起動要因8
- **PSC\_TRIGGER\_EVENT\_9**: 起動要因9
- **PSC\_TRIGGER\_EVENT\_10**: 起動要因10
- **PSC\_TRIGGER\_EVENT\_11**: 起動要因11
- **PSC\_TRIGGER\_EVENT\_12**: 起動要因12
- **PSC\_TRIGGER\_EVENT\_13**: 起動要因13
- **PSC\_TRIGGER\_EVENT\_14**: 起動要因14
- **PSC\_TRIGGER\_EVENT\_15**: 起動要因15

**機能:**

PSC 起動要因の状態を取得します。

**戻り値:**

起動要因の許可/禁止状態

ENABLE: 実行イネーブル

DISABLE: 実行ディセーブル

### 23.2.3.38 PSC\_SetTriggerEventEdge

起動要因 8～11 の外部割り込みのエッジ選択

関数のプロトタイプ宣言:

void

PSC\_SetTriggerEventEdge (uint32\_t **Event**, PSC\_TriggerEventEdge **Edge**)

引数:

**Event**: 起動要因 8～11 の外部割り込みのエッジを選択します。

- PSC\_TRIGGER\_EVENT\_8: 起動要因8
- PSC\_TRIGGER\_EVENT\_9: 起動要因9
- PSC\_TRIGGER\_EVENT\_10: 起動要因10
- PSC\_TRIGGER\_EVENT\_11: 起動要因11

**Edge**: 外部割り込みのエッジを選択します。

- PSC\_START\_TRIGGER\_FALLING: 立ち下りエッジ
- PSC\_START\_TRIGGER\_RISING: 立ち上りエッジ

機能:

起動要因 8～11 の外部割り込みのエッジを選択します。

戻り値:

なし

### 23.2.3.39 PSC\_GetTriggerEventEdge

起動要因 8～11 の外部割り込みのエッジ選択状態の取得

関数のプロトタイプ宣言:

PSC\_TriggerEventEdge

PSC\_GetTriggerEventEdge (uint32\_t **Event**)

引数:

**Event**: 起動要因 8～11 の外部割り込みのエッジを選択します。

- PSC\_TRIGGER\_EVENT\_8: 起動要因8
- PSC\_TRIGGER\_EVENT\_9: 起動要因9
- PSC\_TRIGGER\_EVENT\_10: 起動要因10
- PSC\_TRIGGER\_EVENT\_11: 起動要因11

機能:

起動要因 8～11 の外部割り込みのエッジ選択状態を取得します。

戻り値:

外部割り込みのエッジ選択状態:

PSC\_START\_TRIGGER\_FALLING: 立ち下りエッジ

PSC\_START\_TRIGGER\_RISING: 立ち上りエッジ

## 23.2.3.40 PSC\_GetOverRunFlag

起動要因 15～0 による起動要因オーバーラン発生フラグの取得

関数のプロトタイプ宣言:

PSC\_OverRunFlag

PSC\_GetOverRunFlag (void)

引数:

なし

機能:

起動要因 15～0 による起動要因オーバーラン発生フラグを取得します。

戻り値:

オーバーラン発生フラグを格納した共用体（詳細は”データ構造”を参照）

## 23.2.3.41 PSC\_GetStartupFlag

起動要因 15～0 による起動要因フラグの取得

関数のプロトタイプ宣言:

PSC\_StartupFlag

PSC\_GetStartupFlag (void)

引数:

なし

機能:

起動要因 15～0 による起動要因フラグを取得します。

戻り値:

起動要因 15～0 による起動要因フラグを格納した共用体（詳細は”データ構造”を参照）

## 23.2.3.42 PSC\_ClearEventOverrunFlag

要因番号 15～0 による起動要因発生フラグ/オーバーラン発生フラグのクリア

**関数のプロトタイプ宣言:**

void

PSC\_ClearEventOverrunFlag (uint32\_t *Event*)

**引数:**

*Event*: PSC 起動要因番号を選択します。

- PSC\_TRIGGER\_EVENT\_0: 起動要因0
- PSC\_TRIGGER\_EVENT\_1: 起動要因1
- PSC\_TRIGGER\_EVENT\_2: 起動要因2
- PSC\_TRIGGER\_EVENT\_3: 起動要因3
- PSC\_TRIGGER\_EVENT\_4: 起動要因4
- PSC\_TRIGGER\_EVENT\_5: 起動要因5
- PSC\_TRIGGER\_EVENT\_6: 起動要因6
- PSC\_TRIGGER\_EVENT\_7: 起動要因7
- PSC\_TRIGGER\_EVENT\_8: 起動要因8
- PSC\_TRIGGER\_EVENT\_9: 起動要因9
- PSC\_TRIGGER\_EVENT\_10: 起動要因10
- PSC\_TRIGGER\_EVENT\_11: 起動要因11
- PSC\_TRIGGER\_EVENT\_12: 起動要因12
- PSC\_TRIGGER\_EVENT\_13: 起動要因13
- PSC\_TRIGGER\_EVENT\_14: 起動要因14
- PSC\_TRIGGER\_EVENT\_15: 起動要因15
- PSC\_TRIGGER\_EVENT\_ALL: すべての起動要因

**機能:**

要因番号 15～0 による起動要因発生フラグ/オーバーラン発生フラグをクリアします。

**戻り値:**

なし

## 23.2.3.43 PSC\_SetStartup

起動要因 15～0 の起動要因のセット

**関数のプロトタイプ宣言:**

void

PSC\_SetStartup (uint32\_t *Event*)

**引数:**

*Event*: PSC 起動要因番号を選択します。



- PSC\_TRIGGER\_EVENT\_0: 起動要因0
- PSC\_TRIGGER\_EVENT\_1: 起動要因1
- PSC\_TRIGGER\_EVENT\_2: 起動要因2
- PSC\_TRIGGER\_EVENT\_3: 起動要因3
- PSC\_TRIGGER\_EVENT\_4: 起動要因4
- PSC\_TRIGGER\_EVENT\_5: 起動要因5
- PSC\_TRIGGER\_EVENT\_6: 起動要因6
- PSC\_TRIGGER\_EVENT\_7: 起動要因7
- PSC\_TRIGGER\_EVENT\_8: 起動要因8
- PSC\_TRIGGER\_EVENT\_9: 起動要因9
- PSC\_TRIGGER\_EVENT\_10: 起動要因10
- PSC\_TRIGGER\_EVENT\_11: 起動要因11
- PSC\_TRIGGER\_EVENT\_12: 起動要因12
- PSC\_TRIGGER\_EVENT\_13: 起動要因13
- PSC\_TRIGGER\_EVENT\_14: 起動要因14
- PSC\_TRIGGER\_EVENT\_15: 起動要因15

**機能:**

起動要因 15～0 の起動要因のセットを行います。

**戻り値:**

なし

## 23.2.3.44 PSC\_SetOutputData

PSC 用ポート出力データの設定

**関数のプロトタイプ宣言:**

void

PSC\_SetOutputData (uint8\_t **Data**)

**引数:**

**Data:** PSC 用ポートから出力するデータを設定します。

**機能:**

PSC 用ポートから出力するデータを設定します。

**戻り値:**

なし

## 23.2.3.45 PSC\_SetOutputDataBit

PSC 用ポート出力データビットの設定

## 関数のプロトタイプ宣言:

void

PSC\_SetOutputDataBit (uint8\_t **Bit\_x**, uint8\_t **BitValue**)

## 引数:

**Bit\_x**: PSC 用ポートからデータを出力するビットを選択します。

- **PSC\_BIT\_0**: PSC用ポートビット0
- **PSC\_BIT\_1**: PSC用ポートビット1
- **PSC\_BIT\_2**: PSC用ポートビット2
- **PSC\_BIT\_3**: PSC用ポートビット3
- **PSC\_BIT\_4**: PSC用ポートビット4
- **PSC\_BIT\_5**: PSC用ポートビット5
- **PSC\_BIT\_6**: PSC用ポートビット6
- **PSC\_BIT\_7**: PSC用ポートビット7
- **PSC\_BIT\_ALL**: すべてのPSC用ポートビット

**BitValue**: 出力するビットの値を選択します。

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

## 機能:

PSC 用ポート出力データビットを設定します。

## 戻り値:

なし

### 23.2.3.46 PSC\_SetDataOutputFUNC

PSC 用ポート出力の制御

## 関数のプロトタイプ宣言:

void

PSC\_SetDataOutputFUNC (uint8\_t **Bit\_x**, FunctionalState **NewState**)

## 引数:

**Bit\_x**: PSC 用ポートからのデータ出力を許可/禁止するビットを選択します。

- **PSC\_BIT\_0**: PSC用ポートビット0
- **PSC\_BIT\_1**: PSC用ポートビット1
- **PSC\_BIT\_2**: PSC用ポートビット2
- **PSC\_BIT\_3**: PSC用ポートビット3
- **PSC\_BIT\_4**: PSC用ポートビット4
- **PSC\_BIT\_5**: PSC用ポートビット5
- **PSC\_BIT\_6**: PSC用ポートビット6

- **PSC\_BIT\_7:** PSC用ポートビット7
  - **PSC\_BIT\_ALL:** すべてのPSC用ポートビット
- NewState:** データ出力の許可/禁止を選択します。
- **ENABLE:** 出力許可
  - **DISABLE:** 出力禁止

**機能:**

PSC 用ポート出力の制御を行います。

**戻り値:**

なし

### 23.2.3.47 PSC\_GetDataOutputState

PSC 用ポート出力制御状態の取得

**関数のプロトタイプ宣言:**

FunctionalState

PSC\_GetDataOutputState (uint8\_t **Bit\_x**)

**引数:**

**Bit\_x:** PSC 用ポートからのデータ出力制御状態を取得するビットを選択します。

- **PSC\_BIT\_0:** PSC用ポートビット0
- **PSC\_BIT\_1:** PSC用ポートビット1
- **PSC\_BIT\_2:** PSC用ポートビット2
- **PSC\_BIT\_3:** PSC用ポートビット3
- **PSC\_BIT\_4:** PSC用ポートビット4
- **PSC\_BIT\_5:** PSC用ポートビット5
- **PSC\_BIT\_6:** PSC用ポートビット6
- **PSC\_BIT\_7:** PSC用ポートビット7

**機能:**

PSC 用ポート出力制御状態を取得します。

**戻り値:**

データ出力の許可/禁止状態:

**ENABLE:** 出力許可

**DISABLE:** 出力禁止

### 23.2.3.48 PSC\_GetInputData

PSC 用入力ポートの取得

**関数のプロトタイプ宣言:**

uint8\_t

PSC\_GetInputData (void)

**引数:**

なし

**機能:**

PSC 用入力ポートを取得します。

**戻り値:**

取得した入力ポートの値

## 23.2.3.49 PSC\_GetInputDataBit

PSC 用ポート入力データビットの取得

**関数のプロトタイプ宣言:**

uint8\_t

PSC\_GetInputDataBit (uint8\_t **Bit\_x**)

**引数:**

**Bit\_x**: PSC 用ポートからデータを入力するビットを選択します。

- **PSC\_BIT\_0**: PSC用ポートビット0
- **PSC\_BIT\_1**: PSC用ポートビット1
- **PSC\_BIT\_2**: PSC用ポートビット2
- **PSC\_BIT\_3**: PSC用ポートビット3
- **PSC\_BIT\_4**: PSC用ポートビット4
- **PSC\_BIT\_5**: PSC用ポートビット5
- **PSC\_BIT\_6**: PSC用ポートビット6
- **PSC\_BIT\_7**: PSC用ポートビット7

**機能:**

PSC 用ポート入力データビットを取得します。

**戻り値:**

PSC 用ポート入力データビットの値:

**PSC\_BIT\_VALUE\_0**: 0

**PSC\_BIT\_VALUE\_1**: 1

## 23.2.3.50 PSC\_GetExecutionState

PSC 用プログラム実行状態の取得

**関数のプロトタイプ宣言:**

PSC\_ExecutionState

PSC\_GetExecutionState (void)

**引数:**

なし

**機能:**

PSC 用プログラム実行状態を取得します。

**戻り値:**

PSC 用プログラムの実行状態

**PSC\_PROGRAM\_STOP:** プログラム停止中**PSC\_PROGRAM\_EXECUTE:** プログラム実行中

## 23.2.4 データ構造

### 23.2.4.1 PSC\_ArithmeticPARAM

**メンバ:**

uint32\_t

**All** 演算用レジスタの符号ビット**ビットフィールド:**

uint32\_t

**Reserved0** (Bit 0 ~ Bit15) 未使用

uint32\_t

**SignUA0** (Bit 16) A0の符号ビット

uint32\_t

**SignUM0** (Bit 17) M0の符号ビット

uint32\_t

**SignUM1** (Bit 18) M1の符号ビット

uint32\_t

**SignUL0** (Bit 19) L0の符号ビット

uint32\_t

**SignUL1** (Bit 20) L1の符号ビット

uint32\_t

**SignUR0** (Bit 21) R0の符号ビット

uint32\_t

**SignUR1** (Bit 22) R1の符号ビット

uint32\_t

**Reserved1** (Bit 23 ~ Bit 31) 未使用

## 23.2.4.2 PSC\_ArithmeticResult

メンバ:

uint32\_t

**All** 演算結果フラグレジスタ

ビットフィールド:

uint32\_t

**Reserved0** (Bit 0 ~ Bit15) 未使用

uint32\_t

**OverFlow** (Bit 16) オーバーフローフラグ

uint32\_t

**UnderFlow** (Bit 17) アンダーフローフラグ

uint32\_t

**Reserved1** (Bit 18 ~ Bit 23) 未使用

uint32\_t

**Zero** (Bit 24) ゼロフラグ

uint32\_t

**Reserved1** (Bit 25 ~ Bit 31) 未使用

## 23.2.4.3 PSC\_OverRunFlag

メンバ:

uint32\_t

**All** オーバーランフラグ

ビットフィールド:

uint32\_t

**Reserved** (Bit 0 ~ Bit15) 未使用

uint32\_t

**INTOVRF0** (Bit 16) 起動要因0による起動要因オーバーラン発生フラグ

uint32\_t

**INTOVRF1** (Bit 17) 起動要因1による起動要因オーバーラン発生フラグ

uint32\_t

**INTOVRF2** (Bit 18) 起動要因0による起動要因オーバーラン発生フラグ

uint32\_t

**INTOVRF3** (Bit 19) 起動要因0による起動要因オーバーラン発生フラグ

uint32\_t

**INTOVRF4** (Bit 20) 起動要因0による起動要因オーバーラン発生フラグ

uint32\_t

**INTOVRF5** (Bit 21) 起動要因0による起動要因オーバーラン発生フラグ

uint32\_t

**INTOVRF6** (Bit 22) 起動要因0による起動要因オーバーラン発生フラグ

uint32\_t

<b>INTOVRF7</b>	(Bit 23)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF8</b>	(Bit 24)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF9</b>	(Bit 25)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF10</b>	(Bit 26)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF11</b>	(Bit 27)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF12</b>	(Bit 28)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF13</b>	(Bit 29)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF14</b>	(Bit 30)	起動要因0による起動要因オーバーラン発生フラグ
uint32_t		
<b>INTOVRF15</b>	(Bit 31)	起動要因0による起動要因オーバーラン発生フラグ

## 23.2.4.4 PSC\_StartupFlag

メンバ:

uint32\_t

**All** 起動要因発生フラグ

ビットフィールド:

uint32_t		
<b>INTFLG0</b>	(Bit 0)	起動要因0
uint32_t		
<b>INTFLG1</b>	(Bit 1)	起動要因1
uint32_t		
<b>INTFLG2</b>	(Bit 2)	起動要因2
uint32_t		
<b>INTFLG3</b>	(Bit 3)	起動要因3
uint32_t		
<b>INTFLG4</b>	(Bit 4)	起動要因4
uint32_t		
<b>INTFLG5</b>	(Bit 5)	起動要因5
uint32_t		
<b>INTFLG6</b>	(Bit 6)	起動要因6
uint32_t		
<b>INTFLG7</b>	(Bit 7)	起動要因7
uint32_t		
<b>INTFLG8</b>	(Bit 8)	起動要因8

uint32_t		
<b>INTFLG9</b>	(Bit 9)	起動要因9
uint32_t		
<b>INTFLG10</b>	(Bit 10)	起動要因10
uint32_t		
<b>INTFLG11</b>	(Bit 11)	起動要因11
uint32_t		
<b>INTFLG12</b>	(Bit 12)	起動要因12
uint32_t		
<b>INTFLG13</b>	(Bit 13)	起動要因13
uint32_t		
<b>INTFLG14</b>	(Bit 14)	起動要因14
uint32_t		
<b>INTFLG15</b>	(Bit 15)	起動要因15
uint32_t		
<b>Reserved</b>	(Bit 16 ~ Bit31)	未使用