TOSHIBA

# TX03 ペリフェラルドライバ
# ユーザガイド
# (TMPM370/372/373/374)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M370UG-01J

## 本製品取り扱い上のお願い

- ソフトウエ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

本製品は東芝 TX03 シリーズマイコン用ペリフェラルドライバセットです。TMPM37x ペリフェラルドライバは、東芝TX03 ペリフェラルドライバの重要な部分で、TMPM37x シリーズの MCU 用です。

TX03 ペリフェラルドライバでは、ユーザ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。

TMPM37x 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。
- MCU の全ペリフェラルドライバをカバーしています。

**補足**: TMPM37x は TMPM370, TMPM372, TMPM373, TMPM374 を指します。

# 2. TX03 ペリフェラルドライバの構成

**/Libraries**
TX03 CMSIS ファイルと TMPM37x ペリフェラルドライバが格納されています。

**/Libraries/ TX03_CMSIS**
当フォルダには TMPM37x CMSIS ファイル (デバイス・ペリフェラル・アクセス・レイヤー、およびコア・ペリフェラル・アクセス・レイヤー) が格納されています。

**/Libraries/TX03_Periph_Driver**
TMPM37x コア、ペリフェラルドライバの全てのソースコードが格納されています。

**/Libraries/TX03_Periph_Driver/inc**
TMPM37x ペリフェラルドライバのヘッダファイルが格納されています。

**/Libraries/TX03_Periph_Driver/src**
TMPM37x 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**
TMPM37x 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**
TMPM37x 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**
TMPM37x ペリフェラルドライバの使用例が格納されています。

**/Utilities/TMPM37x-SK**
TMPM37x 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、2 つの 12 ビット逐次変換方式アナログ/デジタルコンバータ(ADC) を内蔵しています。

ADコンバータユニットAは15本のアナログ入力を持っています。6本はモータ0の測定用に使用可能ですが、これら6本の内の3本はIC内部でオペアンプ/コンパレータの出力に接続されているので、外部から入力可能なADは12本です。
AD コンバータユニット B は 17 本のアナログ入力を持っています。6 本はモータ 0 の測定用、および 2 本はモータ 1 の測定用に使用可能ですが、これら 8 本のうち 4 本は IC 内部でオペアンプ/コンパレータの出力に接続されているので、外部から入力可能な AD は 13 本です。

22本の外部アナログ入力端子 (AINA0～AINA8, AINA9/AINB0, AINA10/AINB1, AINA11/AINB2, AINB3～AINB12)は、入出力専用ポートと兼用です。

機能と特徴:

1. PMD や TMRB からのトリガ信号に同期して任意のアナログ入力を変換することができます。
2. ソフトウェア機能、通常起動において任意のアナログ入力を変換することができます。
3. AD 変換値レジスタが 12 個あります。
4. PMD トリガと TMRB トリガのトリガ起動によるプログラム終了時に割り込みを発生できます。
5. ソフトウェア起動、通常起動によるプログラム終了時に割り込みを発生できます。
6. AD 監視機能があります。有効時に比較条件と一致した場合は割り込みを発生します。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37x_adc.c
/Libraries/TX03_Periph_Driver/inc/tmpm37x_adc.h

## 3.2 API 関数

### 3.2.1 関数リスト

- void ADC_SetClk(TSB_AD_TypeDef * ***ADx***, uint32_t ***Sample_HoldTime***, ADC_PRESCALER ***Prescaler_Output***);
- void ADC_Enable(TSB_AD_TypeDef * ***ADx***);
- void ADC_Disable(TSB_AD_TypeDef * ***ADx***);
- void ADC_Start(TSB_AD_TypeDef * ***ADx***, ADC_TrgType ***Trg***);
- void ADC_StopConstantTrg(TSB_AD_TypeDef * ***ADx***);
- WorkState ADC_GetConvertState(TSB_AD_TypeDef * ***ADx***, ADC_TrgType ***Trg***);
- void ADC_SetMonitor(TSB_AD_TypeDef * ***ADx***, ADC_MonitorTypeDef * ***Monitor***);
- void ADC_DisableMonitor(TSB_AD_TypeDef * ***ADx***, ADC_CMPCRx ***CMPCRx***);
- ADC_Result ADC_GetConvertResult(TSB_AD_TypeDef * ***ADx***, ADC_REGx ***ResultREGx***);
- void ADC_SelectPMDTrgProgNum(TSB_AD_TypeDef * ***ADx***, PMD_TRG_PROG_SELx ***SELx***, uint8_t ***MacroProgNum***);

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

## 3.2.2 関数の種類

上記関数は 3 種類に分けられます。

1) ADC 変換の設定
   ADC_SetClk(), ADC_SetMonitor(), ADC_DisableMonitor(), ADC_SelectPMDTrgProgNum(), ADC_SetPMDTrgProgINT(), ADC_SetPMDTrg(), ADC_SetTimerTrg(), ADC_SetSWTrg(), ADC_SetConstantTrg().
2) ADC の有効化/無効か、変換開始
   ADC_Enable(), ADC_Disable(), ADC_Start(), ADC_StopConstantTrg().
3) ADC 変換ステータス/結果読み出し
   ADC_GetConvertState(), ADC_GetConvertResult().

## 3.2.3 関数仕様

### 3.2.3.1 ADC_SetClk

AD 変換サンプルホールド時間とプリスケーラ出力の設定。

**関数のプロトタイプ宣言**:
void ADC_SetClk(TSB_AD_TypeDef * ***ADx***,
uint32_t ***Sample_HoldTime***,
ADC_PRESCALER ***Prescaler_Output***)

**引数**:
***ADx:*** 以下から ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC Unit B

***Sample_HoldTime:*** 以下から ADC サンプルホールド時間を選択します。
- **ADC_HOLD_FIX:** TSH<0:3>へ“1001b”を書き込みます。

***Prescaler_Output:*** 以下から ADC プリスケーラ出力(ADCLK)を選択します。
- **ADC_FC_DIVIDE_LEVEL_NONE:** fc
- **ADC_FC_DIVIDE_LEVEL_2:** fc / 2
- **ADC_FC_DIVIDE_LEVEL_4:** fc / 4
- **ADC_FC_DIVIDE_LEVEL_8:** fc / 8
- **ADC_FC_DIVIDE_LEVEL_16:** fc / 16

**機能:**
**ADC_HOLD_FIX** を設定した ***Sample_HoldTime*** で ADC サンプルホールド時間を、***Prescaler_Output*** でプリスケーラ出力を設定します。

**戻り値:**
なし

### 3.2.3.2 ADC_Enable

指定された ADC ユニットを有効にします。

**関数のプロトタイプ宣言:**
void ADC_Enable(TSB_AD_TypeDef * ***ADx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

**機能:**
指定された ADC ユニットを有効にします。

**戻り値:**
なし

### 3.2.3.3 ADC_Disable

指定された ADC ユニットを無効にします。

**関数のプロトタイプ宣言:**
void ADC_Disable(TSB_AD_TypeDef * ***ADx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

**機能:**
指定された ADC ユニットを無効にします。

**戻り値:**
なし

### 3.2.3.4 ADC_Start

ソフトウェア起動、または通常起動にて AD 変換を開始します。

**関数のプロトタイプ宣言:**
void ADC_Start(TSB_AD_TypeDef * ***ADx***,
ADC_TrgType ***Trg***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***Trg*: 起動方法を設定します。**

- **ADC_TRG_SW:** ソフトウェア起動
- **ADC_TRG_CONSTANT:** 通常起動

**機能:**
指定された起動方法にて ADC ユニットの AD 変換を開始します。

**戻り値:**
なし

### 3.2.3.5 ADC_StopConstantTrg

通常起動時 ADC を停止します。

**関数のプロトタイプ宣言:**
void ADC_StopConstantTrg(TSB_AD_TypeDef * ***ADx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

**機能:**
通常起動時 、本関数で ADC を停止します。

**戻り値:**
なし

### 3.2.3.6 ADC_GetConvertState

AD 変換完了フラグを確認します。

**関数のプロトタイプ宣言:**
WorkState ADC_GetConvertState(TSB_AD_TypeDef * ***ADx***,
ADC_TrgType ***Trg***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***Trg*:** 起動方法を選択します。
- **ADC_TRG_SW:** ソフトウェア起動
- **ADC_TRG_CONSTANT:** 通常起動
- **ADC_TRG_TIMER:** タイマからのトリガ信号
- **ADC_TRG_PMD:** PMD からのトリガ信号

**機能:**
指定された起動方法にて AD 変換ステートを確認します。

**戻り値:**
AD 変換ステート
**BUSY:** 変換中
**DONE:** 変換完了

### 3.2.3.7 ADC_SetMonitor

AD 監視機能を設定します。

**関数のプロトタイプ宣言:**
void ADC_SetMonitor(TSB_AD_TypeDef * ***ADx***,
ADC_MonitorTypeDef * ***Monitor***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***Monitor*****:** 構造体の詳細は以下です。
```
typedef struct {
   ADC_CMPCRx CMPCRx;
   ADC_REGx ResultREGx;
   uint32_t CmpTimes;
   ADC_CmpCondition Condition;
   uint32_t CmpValue;
} ADC_MonitorTypeDef
```
詳細は後述の“データ構造“を参照してください。

**機能:**
ADC_MonitorTypeDef * ***Monitor*** で AC 監視設定を行い、有効にします。

**戻り値:**
なし

### 3.2.3.8 ADC_DisableMonitor

AD 監視機能を無効にします。

**関数のプロトタイプ宣言:**
void ADC_DisableMonitor(TSB_AD_TypeDef * ***ADx***,
ADC_CMPCRx ***CMPCRx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***CMPCRx***: 比較制御レジスタを選択します。
- **ADC_CMPCR_0:** ADxCMPCR0
- **ADC_CMPCR_1:** ADxCMPCR1

**機能:**
***CMPCRx*** で無効にする AD 監視機能を選択します。

**戻り値:**
なし

### 3.2.3.9 ADC_GetConvertResult

AD 変換レジスタの変換結果を読み出します。

**関数のプロトタイプ宣言:**
ADC_Result ADC_GetConvertResult(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***ResultREGx*:** 以下から、AD 変換結果レジスタを選択します。
- **ADC_REG0**: ADxREG0
- **ADC_REG1**: ADxREG1
- **ADC_REG2**: ADxREG2
- **ADC_REG3**: ADxREG3
- **ADC_REG4**: ADxREG4
- **ADC_REG5**: ADxREG5
- **ADC_REG6**: ADxREG6
- **ADC_REG7**: ADxREG7
- **ADC_REG8**: ADxREG8
- **ADC_REG9**: ADxREG9
- **ADC_REG10**: ADxREG10
- **ADC_REG11**: ADxREG11

**機能:**
***ResultREGx*** に設定されている AD 変換結果格納フラグ、オーバーランフラグ、変換結果を読み出します。

**戻り値:**
AD 変換結果のそれぞれのビットの意味は以下です。
**Stored**(Bit0): AD 変換結果格納状態
**OverRun**(Bit1):オーバーラン状態
**ADResult**(Bit4 to Bit15): AD 変換結果値

### 3.2.3.10 ADC_SelectPMDTrgProgNum

ADC ユニットの PMD が発生するトリガ信号 PMD0 から PMD11 に対して、起動するプログラム番号 0 から 5 を選択します。

**関数のプロトタイプ宣言:**
Void ADC_SelectPMDTrgProgNum(TSB_AD_TypeDef * ***ADx***,
PMD_TRG_PROG_SELx ***SELx***,
uint8_t ***MacroProgNum***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***SELx*:** PMD トリガ用プログラム選択番号を下記から選択します。

- **PMD_TRG_PROG_SEL0**: ADxPSEL0
- **PMD_TRG_PROG_SEL1**: ADxPSEL1
- **PMD_TRG_PROG_SEL2**: ADxPSEL2
- **PMD_TRG_PROG_SEL3**: ADxPSEL3
- **PMD_TRG_PROG_SEL4**: ADxPSEL4
- **PMD_TRG_PROG_SEL5**: ADxPSEL5
- **PMD_TRG_PROG_SEL6**: ADxPSEL6
- **PMD_TRG_PROG_SEL7**: ADxPSEL7
- **PMD_TRG_PROG_SEL8**: ADxPSEL8
- **PMD_TRG_PROG_SEL9**: ADxPSEL9
- **PMD_TRG_PROG_SEL10**: ADxPSEL10
- **PMD_TRG_PROG_SEL11**: ADxPSEL11

***MacroProgNum*:** マクロ TRG_ENABLE(x)/TRG_DISABLE(x) を使用し、選択するプログラム番号のディセーブル/イネーブルを設定します。

- **TRG_ENABLE(PMD_PROG0)**: PMD トリガ用プログラム選択レジスタ プログラム 0 をイネーブル
- **TRG_DISABLE(PMD_PROG0)**: PMD トリガ用プログラム選択レジスタ プログラム 0 をディセーブル
- **TRG_ENABLE(PMD_PROG1)**: PMD トリガ用プログラム選択レジスタ プログラム 1 をイネーブル
- **TRG_DISABLE(PMD_PROG1)**: PMD トリガ用プログラム選択レジスタ プログラム 1 をディセーブル
- **TRG_ENABLE(PMD_PROG2)**: PMD トリガ用プログラム選択レジスタ プログラム 2 をイネーブル
- **TRG_DISABLE(PMD_PROG2)**: PMD トリガ用プログラム選択レジスタ プログラム 2 をディセーブル
- **TRG_ENABLE(PMD_PROG3)**: PMD トリガ用プログラム選択レジスタ プログラム 3 をイネーブル
- **TRG_DISABLE(PMD_PROG3)**: PMD トリガ用プログラム選択レジスタ プログラム 3 をディセーブル
- **TRG_ENABLE(PMD_PROG4)**: PMD トリガ用プログラム選択レジスタ プログラム 4 をイネーブル
- **TRG_DISABLE(PMD_PROG4)**: PMD トリガ用プログラム選択レジスタ プログラム 4 をディセーブル
- **TRG_ENABLE(PMD_PROG5)**: PMD トリガ用プログラム選択レジスタ プログラム 5 をイネーブル
- **TRG_DISABLE(PMD_PROG5)**: PMD トリガ用プログラム選択レジスタ プログラム 5 をディセーブル

**機能:**
本関数は、***SELx*** で設定される ADC ユニットの PMD トリガ用プログラム選択レジスタを設定し、***MacroProgNum*** でレジスタのイネーブル/ディセーブルを選択します。

**戻り値:**
なし

### 3.2.3.11 ADC_SetPMDTrgProgINT

ADC ユニットのプログラム 0 から 5 に対して割り込み発生を選択します。

**関数のプロトタイプ宣言:**

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

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***TrgProgINT*:** 全 PMD トリガ用プログラムの割り込み設定の構造体。
```
typedef struct {
   PMD_INT_NAME INTProg0;
   PMD_INT_NAME INTProg1;
   PMD_INT_NAME INTProg2;
   PMD_INT_NAME INTProg3;
   PMD_INT_NAME INTProg4;
   PMD_INT_NAME INTProg5;
} PMD_TrgProgINTTypeDef
```
この構造体の詳細は、“データ構造”を参照してください。

**機能:**
本関数は、***TrgProgINT*** により ADC ユニットのプログラム 0 から 5 に対して割り込みを設定します。

**戻り値:**
なし

### 3.2.3.12 ADC_SetPMDTrg

ADC ユニットの PMD トリガ用プログラムレジスタの設定

**関数のプロトタイプ宣言:**
void ADC_SetPMDTrg(TSB_AD_TypeDef * ***ADx***,
PMD_TrgTypeDef * ***PMDTrg***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***PMDTrg*:** PMD トリガ用プログラムレジスタの割り込み設定の構造体。
```
typedef struct {
   PMD_PROGx ProgNum;
   VE_PHASE Reg0_Phase;
   VE_PHASE Reg1_Phase;
   VE_PHASE Reg2_Phase;
   VE_PHASE Reg3_Phase;
   uint8_t Reg0_AINx;
   uint8_t Reg1_AINx;
   uint8_t Reg2_AINx;
   uint8_t Reg3_AINx;
} PMD_TrgTypeDef
```
この構造体の詳細は、“データ構造”を参照してください。

**機能:**

本関数は、***PMDTrg*** により ADC ユニットの PMD トリガ用プログラムレジスタを設定します。

**戻り値:**
なし

### 3.2.3.13 ADC_SetTimerTrg

ADC ユニットのタイマトリガ用プログラムレジスタの設定

**関数のプロトタイプ宣言:**
void ADC_SetTimerTrg(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***,
uint8_t ***MacroAINx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***ResultREGx*:** ADC ユニットのタイマトリガ用プログラムレジスタの設定を下記から選択します。

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

***MacroAINx***: TRG_ENABLE(x)/TRG_DISABLE(x) のマクロを使用し、イネーブル/ディセーブルを AIN 端子に設定します。

- **TRG_ENABLE(ADC_AIN0)**: ADC_AIN0 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN0)**:ADC_AIN0 に対しレジスタをディセーブル設定
- **TRG_ENABLE(ADC_AIN1)**: ADC_AIN1 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN1)**:ADC_AIN1 に対しレジスタをディセーブル設定
- **.** .
- **.** .
- **.** .
- **TRG_ENABLE(ADC_AIN15)**:ADC_AIN15 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN15)**:ADC_AIN15 に対しレジスタをディセーブル設定
- **TRG_ENABLE(ADC_AIN16)**:ADC_AIN16 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN16)**:ADC_AIN16 に対しレジスタをディセーブル設定

*** 注:** **Unit A 用: ADC_AIN0～ADC_AIN14**
**Unit B 用: ADC_AIN0～ADC_AIN16**

**機能:**
本関数は、**ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11** に対し、***ResultREGx*** で ADC のユニットの AD 変換結果レジスタの設定を行い、タイマトリガプログラム用レジスタの ***MacroAINx*** で AIN 端子に対しレジスタのイネーブル/ディセーブルの設定を行います。

**戻り値:**
なし

### 3.2.3.14 ADC_SetSWTrg

ADC ユニットのソフトウェアトリガ用プログラムレジスタを設定

**関数のプロトタイプ宣言:**
void ADC_SetSWTrg(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***,
uint8_t ***MacroAINx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***ResultREGx*:** ソフトウェアトリガプログラム用 AD 変換結果レジスタの設定値を以下から選択。

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

***MacroAINx***: TRG_ENABLE(x)/TRG_DISABLE(x) マクロを用いて、選択する AIN 端子の選択、イネーブル/ディセーブル設定を行います。
- **TRG_ENABLE(ADC_AIN0)**:ADC_AIN0 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN0)**:ADC_AIN0 に対しレジスタをディセーブル設定
- **TRG_ENABLE(ADC_AIN1)**: ADC_AIN1 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN1)**:ADC_AIN1 に対しレジスタをディセーブル設定
- **.** .
- **.** .
- **.** .
- **TRG_ENABLE(ADC_AIN15)**:ADC_AIN15 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN15)**:ADC_AIN15 に対しレジスタをディセーブル設定

- **TRG_ENABLE(ADC_AIN16)**:ADC_AIN16 に対しレジスタをイネーブル設定
- **TRG_DISABLE(ADC_AIN16)**:ADC_AIN16 に対しレジスタをディセーブル設定

*** 注:** **Unit A: ADC_AIN0～ADC_AIN14**
**Unit B: ADC_AIN0～ADC_AIN16**

**機能:**
本関数は、**ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11** に対し、***ResultREGx*** で ADC ユニットの AD 変換結果レジスタ の設定を行い、ソフトウェアトリガプログラム用レジスタの ***MacroAINx*** で AIN 端子に対しレジスタのイネーブル/ディセーブルの設定を行います。

**戻り値:**
なし

### 3.2.3.15 ADC_SetConstantTrg

ADC ユニットの常時トリガ用プログラムレジスタの設定

**関数のプロトタイプ宣言:**
void ADC_SetConstantTrg(TSB_AD_TypeDef * ***ADx***,
ADC_REGx ***ResultREGx***,
uint8_t ***MacroAINx***)

**引数:**
***ADx:*** ADC ユニットを選択します。
- **TSB_ADA:** ADC ユニット A
- **TSB_ADB:** ADC ユニット B

***ResultREGx*:** 常時トリガプログラム用 AD 変換結果レジスタの設定を選択します。

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

***MacroAINx***: TRG_ENABLE(x)/TRG_DISABLE(x) のマクロにより、AIN 端子の選択、およびイネーブル/ディセーブル設定を行います。

- **TRG_ENABLE(ADC_AIN0):ADC_AIN0** に対し、レジスタイネーブル
- **TRG_DISABLE(ADC_AIN0):ADC_AIN0** に対し、レジスタディセーブル
- **TRG_ENABLE(ADC_AIN1): ADC_AIN1** に対し、レジスタイネーブル
- **TRG_DISABLE(ADC_AIN1):ADC_AIN1** に対し、レジスタディセーブル

- . .
- . .
- . .
- **TRG_ENABLE(ADC_AIN15):ADC_AIN15** に対し、レジスタイネーブル
- **TRG_DISABLE(ADC_AIN15):ADC_AIN15** に対し、レジスタディセーブル
- **TRG_ENABLE(ADC_AIN16) ADC_AIN16** に対し、レジスタイネーブル
- **TRG_DISABLE(ADC_AIN16):ADC_AIN16** に対し、レジスタディセーブル

*** 注:** **Unit A 用: ADC_AIN0～ADC_AIN14**
**Unit B 用: ADC_AIN0～ADC_AIN16**

**機能:**
本関数は、**ADC_REG_0**, **ADC_REG_1**, **ADC_REG_2**, **ADC_REG_3**, **ADC_REG_4**, **ADC_REG_5**, **ADC_REG_6**, **ADC_REG_7**, **ADC_REG_8**, **ADC_REG_9**, **ADC_REG_10**, **ADC_REG_11** に対し、***ResultREGx*** で ADC ユニットの AD 変換結果レジスタ の設定を行い、常時トリガ用プログラムレジスタの ***MacroAINx*** で AIN 端子に対し、レジスタイネーブル/ディセーブルの設定を行います。

**戻り値:**
なし

## 3.2.4 データ構造

### 3.2.4.1 ADC_MonitorTypeDef

**メンバ:**

ADC_CMPCRx
***CMPCRx :*** コンペア制御レジスタを以下から選択します。

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

ADC_REGx
***ResultREGx***: AD 変換結果レジスタを選択します。
- **ADC_REG0:** ADxREG0
- **ADC_REG1:** ADxREG1
- **ADC_REG2:** ADxREG2
- **ADC_REG3:** ADxREG3
- **ADC_REG4:** ADxREG4
- **ADC_REG5:** ADxREG5
- **ADC_REG6:** ADxREG6
- **ADC_REG7:** ADxREG7
- **ADC_REG8:** ADxREG8
- **ADC_REG9:** ADxREG9
- **ADC_REG10:** ADxREG10
- **ADC_REG11:** ADxREG11

uint32_t
***CmpTimes :***比較のカウント数を設定します。
:
**1** から **16**

ADC_CmpCondition

***Condition*** : ADxREGn、ADxCMPm の比較設定を下記から選択
( x = A, B; n= 0 ～11, m = 0～1 )

- **ADC_LARGER_THAN_CMP_REG:** 変換結果レジスタ値が比較レジスタ 0 より大きい場合、割込みを発生。
- **ADC_SMALLER_THAN_CMP_REG:** 変換結果レジスタ値が比較レジスタ 0 より小さい場合、割込みを発生。

uint32_t
***CmpValue*** :ADxCMP0 または ADxCMP1 に設定する比較値。
**0 ～4095**

### 3.2.4.2 PMD_TrgProgINTTypeDef

**メンバ:**

PMD_INT_NAME
***INTProg0*** :プログラム 0 に対して割り込みを設定。
- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDA:** INTADPDA output
- **PMD_INTADPDB:** INTADPDB output

PMD_INT_NAME
***INTProg1*** :プログラム 1 に対して割り込みを設定。
- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDA:** INTADPDA 出力
- **PMD_INTADPDB:** INTADPDB 出力

PMD_INT_NAME
***INTProg2*** :プログラム 2 に対して割り込みを設定。

- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDA:** INTADPDA 出力
- **PMD_INTADPDB:** INTADPDB 出力

PMD_INT_NAME
***INTProg3***:プログラム 3 に対して割り込みを設定。
- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDA:** INTADPDA 出力
- **PMD_INTADPDB:** INTADPDB 出力

PMD_INT_NAME
***INTProg4*** :プログラム 4 に対して割り込みを設定。
- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDA:** INTADPDA 出力
- **PMD_INTADPDB:** INTADPDB 出力

PMD_INT_NAME
***INTProg5*** :プログラム 5 に対して割り込みを設定。
- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDA:** INTADPDA 出力
- **PMD_INTADPDB:** INTADPDB 出力

### 3.2.4.3 PMD_TrgTypeDef

**メンバ:**

PMD_PROGx
***ProgNum*** :ADxPSETn (x = A, B; n = 0 to 5) に対しプログラム番号を選択
**PMD_PROG0 ～ PMD_PROG5**

VE_PHASE
***Reg0_Phase*** :ADxPSETn の REG0 に対し、以下からフェイズを選択。
- **VE_PHASE_NONE:** 指定なし
- **VE_PHASE_U:** U
- **VE_PHASE_V:** V
- **VE_PHASE_W:** W

VE_PHASE
***Reg1_Phase*** : ADxPSETn の REG1 に対し、以下からフェイズを選択。他の情報は上記と同じ。

VE_PHASE
***Reg2_Phase*** **:**ADxPSETn の REG2 に対し、以下からフェイズを選択。他の情報は上記と同じ。

VE_PHASE
***Reg3_Phase*** **:**ADxPSETn の REG3 に対し、以下からフェイズを選択。他の情報は上記と同じ。

uint8_t
***Reg0_AINx*** ADxPSETn の REG0 に対してイネーブル/ディセーブル設定を行い、またアナログ入力チャネルを選択。下記の形式でマクロと共に入力。**TRG_ENABLE(y), TRG_DISABLE(y).**
上記の 'y' は、下記のいずれの値。
(ユニット A): **ADC_AIN0 ～ ADC_AIN14**
(ユニット B): **ADC_AIN0 ～ ADC_AIN16**

uint8_t
***Reg1_AINx*** :ADxPSETn の REG1 に対してイネーブル/ディセーブル設定を行い、またアナログ入力チャネルを選択。他の情報は上記設定と同様。

uint8_t
***Reg2_AINx*** :ADxPSETn の REG2 に対してイネーブル/ディセーブル設定を行い、またアナログ入力チャネルを選択。他の情報は上記設定と同様。

uint8_t
***Reg3_AINx*** :ADxPSETn の REG3 に対してイネーブル/ディセーブル設定を行い、またアナログ入力チャネルを選択。他の情報は上記設定と同様。

### 3.2.4.4 ADC_Result

**メンバ:**

uint32_t
***All:*** AD 変換結果
***Bit***

uint32_t
***Stored***: 1 AD 変換結果の格納状態
uint32_t
***OverRun***: 1 AD 変換オーバーランフラグ
uint32_t
***Reserved1***: 2 予約
uint32_t
***ADResult***: 12 AD 変換結果
uint32_t
***Reserved2***: 16予約

# 4. CG

## 4.1 概要

本ドライバは TMPM37x CG 使用において以下の機能を提供します。
- 高速発振器、PLL(逓倍回路)の設定
- クロックギア、プリスケーラクロック、PLL、発振器の設定
- ウォームアップタイマの設定と結果の読み出し
- 低消費電力モードの設定
- 動作モードの変更 (ノーマルモード、低速モード、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。
TX03_Periph_Driver\src\tmpm37x_cg.c
TX03_Periph_Driver\inc\tmpm37x_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** : X1端子より入力されるクロック
**fpll** : PLLにより逓倍されたクロック
**fc** : CGPLLSEL<PLLSEL>で選択されたクロック(高速クロック)
**fgear** : CGSYSCR<GEAR2:0>で選択されたクロック
**fsys** : CGCKSEL<SYSCK>で選択されたクロック(システムクロック)
**fperiph** : CGSYSCR<FPSEL>で選択されたクロック.
**ΦT0** : CGSYSCR<PRCK2:0>で選択されたクロック(プリスケーラクロック)

## 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_SetWarmUpTime(uint16_t ***Time***)
- void CG_StartWarmUp(void)
- WorkState CG_GetWarmUpState(void)
- Result CG_SetPLL(FunctionalState ***NewState***)
- FunctionalState CG_GetPLLState(void)
- void CG_SetFosc(FunctionalState ***NewState***)
- FunctionalState CG_GetFoscState(void)
- void CG_SetPortM(CG_PortMMode ***Mode***)
- CG_INTReqState CG_GetINTReq(CG_INTSrc ***INTSource***)
- void CG_SetSTBYMode(CG_STBYMode ***Mode***)

- CG_STBYMode CG_GetSTBYMode(void)
- void CG_SetPinStateInStopMode(FunctionalState ***NewState***)
- FunctionalState CG_GetPinStateInStopMode(void)
- Result CG_SetFcSrc(CG_FcSrc ***Source***)
- CG_FcSrc CG_GetFcSrc(void)
- void CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,
  CG_INTActiveState ***ActiveState***,
  FunctionalState ***NewState***)
- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***)
- void CG_ClearINTReq(CG_INTSrc ***INTSource***)
- CG_NMIFactor CG_GetNMIFlag(void)
- CG_ResetFlag CG_GetResetFlag(void)

## 4.2.2 関数の種類

CG API は主に以下の 3 種類に分けられます。

1) クロックの選択
   CG_SetFgearLevel(), CG_GetFgearLevel(), CG_SetPhiT0Src(),
   CG_GetPhiT0Src(), CG_SetPhiT0Level(), CG_GetPhiT0Level(),
   CG_SetWarmUpTime(), CG_StartWarmUp(), CG_GetWarmUpState(),
   CG_SetPLL(), CG_GetPLLState(), CG_SetFosc(), CG_GetFoscState(),
   CG_SetFcSrc(), CG_GetFcSrc(), CG_SetPortM()

2) スタンバイモードの設定
   CG_SetSTBYMode(), CG_GetSTBYMode( ),
   CG_SetPinStateInStopMode(), CG_GetPinStateInStopMode().

3) 割り込みの設定
   CG_SetSTBYReleaseINTSrc(),
   CG_GetSTBYReleaseINTState(),CG_ClearINTReq(), CG_GetNMIFlag(),
   CG_GetResetFlag(), CG_GetINTReq()

## 4.2.3 関数仕様

### 4.2.3.1 CG_SetFgearLevel

fgear,fc 間の分周レベルを設定します。

**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 間の分周レベルを取得します。

**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:** 無効なデータ

#### 4.2.3.3 CG_SetPhiT0Src

PhiT0(ΦT0) ,fc 間の PhiT0(ΦT0) ソースを設定します。

**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**:PhiT0 ソースが fgear
- **CG_PHIT0_SRC_FC**: PhiT0 ソースが fs

**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) ソースを取得します。

**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** : PhiT0 ソースが fgear
**CG_PHIT0_SRC_FC** : PhiT0 ソースが fc

### 4.2.3.5 CG_SetPhiT0Level

PhiT0 (ΦT0) と fc 間の分周レベルを設定します。

**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:**
プリスケー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(ΦT0) ,fc 間の分周レベルを取得します。

**関数のプロトタイプ宣言:**
CG_DivideLevel
CG_GetPhiT0Level(void)

**引数:**
なし

**機能:**
PhiT0(ΦT0) ,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(ΦT0) ,fc 間の分周レベルを以下から設定します。
**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
**CG_DIVIDE_UNKNOWN**：無効なデータの読み込み

### 4.2.3.7 CG_SetWarmUpTime

ウォームアップ時間を設定します。

**関数のプロトタイプ宣言:**
void CG_SetWarmUpTime (uint16_t ***Time***)

**引数:**
***Time:*** クロックは 0U から 0x1000U になります。

**機能:**
本関数はウォームアップ時間とウォームアップカウンタを設定します。計算式は下記になります。
Setting_value = ((warm-up time) / (input cycle time by frequency))/16

ウォームアップ時間の計算レジスタ値例:
/* set up warm time 100us, input cycle by frequency is 8M */
So value = 100*10E(-6)/(1/(8*10E(6)))/16=0x0320>>4=0x32

**戻り値**:
なし.

### 4.2.3.8 CG_StartWarmUp

ウォームアップを開始します。

**関数のプロトタイプ宣言:**
void
CG_StartWarmUp (void)

**引数:**
なし

**機能:**
ウォームアップを開始します。

**戻り値:**
なし

### 4.2.3.9 CG_GetWarmUpState

ウォーミングアップ動作状態 (動作中、完了)を確認します。

**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。

ウォーミングアップタイマの使用例:

```
CG_SetWarmUpTime(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

### 4.2.3.10 CG_SetPLL

PLL 回路を設定します。

**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 回路を設定します。fc として PLL が選択されている場合は **ERROR** を返します。

**戻り値:**
**SUCCESS**: 成功
**ERROR**: エラー

### 4.2.3.11 CG_GetPLLState

PLL 回路の状態を取得します。

**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

### 4.2.3.12 CG_SetFosc

高速発振器を設定します。

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

**引数:**
***NewState***: 高速発振器を選択します。
- **ENABLE**: 高速発振器有効
- **DISABLE**: 高速発振器無効

**機能:**
高速発振器を設定します。.

**戻り値:**
なし

### 4.2.3.13 CG_GetFoscState

高速発振器の状態を取得します。

**関数のプロトタイプ宣言:**
FunctionalState CG_GetFoscState(void)

**引数:**
なし

**機能:**
高速発振器の状態を取得します。

**戻り値:**
高速発振器の状態
**ENABLE**: 高速発振器有効
**DISABLE**: 高速発振器無効

### 4.2.3.14 CG_SetPortM

ポート M の設定(X1/X2 または汎用ポート)

**関数のプロトタイプ宣言:**
void
CG_SetPortM(CG_PortMMode ***Mode***)

**引数:**

***Mode***:
- **CG_PORTM_AS_GPIO** : 汎用ポート
- **CG_PORTM_AS_HOSC**: X1/X2

**機能:**
ポート M の設定を行います。***Mode*** が **CG_PORTM_AS_GPIO** の時は汎用ポート、***Mode*** が **CG_PORTM_AS_HOSC** の時は X1/X2 に設定します。

**補足:**
この API は、M372/3/4 のみ使用可能です。

**戻り値:**
なし

### 4.2.3.15 CG_GetINTReq

割り込み発生状態の取得

**関数のプロトタイプ宣言:**
CG_INTReqState
CG_GetINTReq(CG_INTSrc ***INTSource***)

**引数:**
***INTSource*** : 以下から、割り込みソースを選択します。
- **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_C** : INT12
- **CG_INT_SRC_D** : INT13
- **CG_INT_SRC_E** : INT14
- **CG_INT_SRC_F** : INT15

**機能:**
割り込み発生状態を取得します。

**補足:**
この API は、M372/3/4 のみ使用可能です。
M373: CG_INT_SRC_3 と CG_INT_SRC_8 は無効です。
M374: CG_INT_SRC_3 と CG_INT_SRC_4 と CG_INT_SRC_8 は無効です。

**戻り値:**
割り込み発生状態:
**CG_NO_INT_REQ**:< INTxF >=0
**CG_INT_REQ**:< INTxF >=1

### 4.2.3.16 CG_SetSTBYMode

スタンバイモードを設定します。

**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_STOP**: STOP モード (内部発振器を含めてすべての内部回路を停止)
- **CG_STBY_MODE_IDLE**: IDLE モード(CPU 停止)

**機能:**
スタンバイ命令使用時のスタンバイモードを設定します。

**戻り値:**
なし

### 4.2.3.17 CG_GetSTBYMode

スタンバイモードの設定状態を取得します。

**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。レジスタから読み出した値が“Reserved”の場合、**CG_STBY_MODE_UNKNOWN** を返します。

**戻り値:**
低電力モード:
**CG_STBY_MODE_STOP**: STOP モード.
**CG_STBY_MODE_IDLE**: IDLE モード
**CG_STBY_MODE_UNKNOWN**: 無効なデータの読み込み

### 4.2.3.18 CG_SetPinStateInStopMode

STOP 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_SetPinStateInStopMode (FunctionalState ***NewState***)

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

“<DRVE>=0”、“<DRVE>=1” の端子設定に関しては、『STOP モードの端子設定』を参照ください。

**機能:**
STOP モード中の端子状態を設定します。

**戻り値:**

なし

#### 4.2.3.19 CG_GetPinStateInStopMode

STOP モード中の端子状態を取得します。

**関数のプロトタイプ宣言:**
FunctionalState CG_GetPinStateInStopMode (void)

**引数:**
なし

**機能:**
STOP モード中の端子状態を取得します。

**戻り値:**
STOP モード中の端子状態
**DISABLE**: <DRVE>=0
**ENABLE**: <DRVE>=1

#### 4.2.3.20 CG_SetFcSrc

fc のクロックソースを設定します。

**関数のプロトタイプ宣言:**
Result CG_SetFcSrc(CG_FcSrc ***Source***)

**引数:**
***Source***: 以下から、fc のクロックソースを選択します。
- **CG_FC_SRC_FOSC** : fosc
- **CG_FC_SRC_FPLL**: fpll

**機能:**
fc のクロックソースを設定します。本関数を実行する前に以下を設定してください。
a) 高速発振器を ON にする。
b) ***Source*** で **CG_FC_SRC_FPLL** を選択した場合は、**CG_SetPLL**(ENABLE)を実行し、PLL 回路を有効にしてください。
上記の条件以外の場合は、**ERROR** が返されます。

**戻り値:**
**SUCCESS**: 成功
**ERROR**: エラー

#### 4.2.3.21 CG_GetFcSrc

fc のクロックソースを取得します。

**関数のプロトタイプ宣言:**
CG_FcSrc CG_GetFosc (void)

**引数:**
なし

**機能:**

fc のクロックソースを取得します。

**戻り値:**
fc のクロックソースは下記の値のいずれかになります。
**CG_FC_SRC_FOSC**: fosc.
**CG_FC_SRC_FPLL**: fpll.

### 4.2.3.22 CG_SetSTBYReleaseINTSrc

スタンバイモードの解除割り込みソースを設定します。

**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** : INT10
- **CG_INT_SRC_B** : INT11
- **CG_INT_SRC_C** : INT12
- **CG_INT_SRC_D** : INT13
- **CG_INT_SRC_E** : INT14
- **CG_INT_SRC_F** : INT15

***ActiveState***: 以下から、解除トリガのアクティブ状態を選択します。
- **CG_INT_ACTIVE_STATE_L**: 低レベル
- **CG_INT_ACTIVE_STATE_H**: 高レベル
- **CG_INT_ACTIVE_STATE_FALLING**: 立ち下がりエッジ
- **CG_INT_ACTIVE_STATE_RISGING**: 立ち上がりエッジ
- **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**: 割り込み発生とアクティブ状態が一致したときにスタンバイモードを解除します
- **DISABLE:** 割り込み発生とアクティブ状態が一致したときでもスタンバイモードを解除しません

**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

### 4.2.3.23 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を取得します。

**関数のプロトタイプ宣言**:
CG_INT_ActiveState CG_GetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***)

**引数:**
***INTSource***: 以下から解除割り込みソースを選択します。
**CG_INT_SRC_0, CG_INT_SRC_1, CG_INT_SRC_2, CG_INT_SRC_3,**
**CG_INT_SRC_4, CG_INT_SRC_5, CG_INT_SRC_6, CG_INT_SRC_7,**
**CG_INT_SRC_8, CG_INT_SRC_9, CG_INT_SRC_A, CG_INT_SRC_B,**
**CG_INT_SRC_C, CG_INT_SRC_D, CG_INT_SRC_E, CG_INT_SRC_F.**

**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_FALLING**: 立ち下がりエッジ
**CG_INT_ACTIVE_STATE_RISING**: 立ち上がりエッジ
**CG_INT_ACTIVE_STATE_BOTH_EDGES**: 両エッジ
**CG_INT_ACTIVE_STATE_INVALID**: 無効

### 4.2.3.24 CG_ClearINTReq

スタンバイ解除割り込み要求をクリアします。

**関数のプロトタイプ宣言**:
void CG_ClearINTReq(CG_INTSrc ***INTSource***)

**引数:**
***INTSource***: 以下から、解除割り込みソースを選択します。
**CG_INT_SRC_0, CG_INT_SRC_1, CG_INT_SRC_2, CG_INT_SRC_3,**
**CG_INT_SRC_4, CG_INT_SRC_5, CG_INT_SRC_6, CG_INT_SRC_7,**
**CG_INT_SRC_8, CG_INT_SRC_9, CG_INT_SRC_A, CG_INT_SRC_B,**
**CG_INT_SRC_C, CG_INT_SRC_D, CG_INT_SRC_E, CG_INT_SRC_F.**

**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

### 4.2.3.25 CG_GetNMIFlag

NMI 起動要因フラグを取得します。

**関数のプロトタイプ宣言**:
CG_NMIFactor CG_ GetNMIFlag (void)

**引数**:
なし

**機能:**
NMI 起動要因フラグを取得します。

**戻り値:**
NMI 起動要因
**WDT** (Bit 0) :WDT により起動

#### 4.2.3.26 CG_GetResetFlag

リセットフラグの取得とクリアを行います。

**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:
**PowerOn** (Bit 0) :power-on からのリセット
**ResetPin** (Bit 1) :Reset 端子からのリセット
**WDTReset** (Bit 2):WDT からのリセット
**VLTDReset** (Bit 3) :LVTD からのリセット
**DebugReset** (Bit 4) :SYSRESETREQ からのリセット
**OFDReset** (Bit 5) :OFD からのリセット

### 4.2.4 データ構造

#### 4.2.4.1 CG_NMIFactor

**メンバ:**
uint32_t ***All*** :CGNMI ソース起動状態を指定します。

**ビットフィールド**:
uint32_t ***WDT***(Bit 0):WDT から

#### 4.2.4.2 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 ***PowerOn***(Bit 0) :power-on からのリセット
uint32_t ***ResetPin***(Bit 1) :Reset 端子からのリセット
uint32_t ***WDTReset***(Bit 2) :WDT からのリセット
uint32_t ***VLTDReset*** (Bit 3) :VLTD からのリセット
uint32_t ***DebugReset***(Bit 4) :デバッガからのリセット

uint32_t ***OFDReset***(Bit 5) : OFD からのリセット

# 5. FC

## 5.1 概要

TMPM37xFYFG MCU は 256K バイトの NANO flash を搭載しています。

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。

フラッシュメモリのブロック情報は、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 定義を格納する以下のファイルで構成されています。
\Libraries\TX03_Periph_Driver\src\tmpm37x_fc.c
\Libraries\TX03_Periph_Driver \inc\tmpm37x_fc.h

## 5.2 API 関数

### 5.2.1 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)

### 5.2.2 関数の種類

上記関数は 4 種類に分類されます。

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) プロテクト状態の取得:
   FC_GetBusyState(), FC_GetBlockProtectState(),
3) プロテクトの設定と解除:
   FC_ProgramBlockProtectState(), FC_EraseBlockProtectState().
4) 書き込みと消去:
   FC_WritePage(), FC_EraseBlock(), FC_EraseChip().

### 5.2.3 関数仕様

#### 5.2.3.1 FC_SetSecurityBit

SECBIT レジスタの設定

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

**引数:**
***NewState***: 以下から SECBIT レジスタの状態を選択します。
- **DISABLE**: セキュリティ機能設定不可
- **ENABLE**: セキュリティ機能設定可能

**機能:**
1) 本関数を使用するには、書き込み/消去に使用する全保護ビット (FLCS<BLPRO> ビット) を”1” に設定します。
2) SECBIT <SECBIT> ビットも”1” に設定します。

同時に上記の二つの条件が一致したときのみ、Flash ROM データ読み出し、デバッグを制限するセキュリティ機能が使用できます。この時、JTAG/SW 通信は禁止となりますので、 JTAG を使用したデバッグを行うことはできません。API で SECBIT<SEBIT> に “1” を設定する際は、ご注意ください。

起動直後のパワーオンリセット時、SECBIT <SECBIT> bit が “1” に設定されます。

**戻り値:**
なし

### 5.2.3.2 FC_GetSecurityBit

SECBIT レジスタの取得

**関数のプロトタイプ宣言:**
FunctionalState
FC_GetSecurityBit(void)

**引数:**
なし

**機能:**
SECBIT レジスタの値を取得します。
SECBIT <SECBIT> ビット値が”1” (**ENABLE**) の時、有効。
SECBIT <SECBIT> ビット値が”0” (**DISABLE**) の時, 無効。

**戻り値:**
SECBIT レジスタの状態
**DISABLE**: セキュリティ機能設定不可
**ENABLE**: セキュリティ機能設定可能

### 5.2.3.3 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:**
Flash メモリが自動動作の時は、"0" を出力し、ビジーであることを示します。自動動作終了時、レディ状態となり、次のコマンドを受け付けられることを示す “1” を表示します。

**戻り値:**
フラッシュメモリの自動動作状態
**BUSY**: 自動動作中
**DONE**: 自動動作終了(コマンド待ち状態)

### 5.2.3.4 FC_GetBlockProtectState

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***: 以下からブロック番号を選択します。
- **FC_BLOCK_0**: block 0
- **FC_BLOCK_1**: block 1
- **FC_BLOCK_2:** block 2
- **FC_BLOCK_3**: block 3
- **FC_BLOCK_4:** block 4
- **FC_BLOCK_5:** block 5

**機能:**
各プロテクトビットはプロテクト状態を示します。ビットが "1" に設定されている場合、プロテクトされていることを示します。プロテクト状態の時には、データを書き込むことはできません。Flash メモリのブロック設定については、概要を参照ください。

**戻り値:**
ブロックのプロテクト状態
**DISABLE**: プロテクトされていない
**ENABLE**: プロテクトされている

### 5.2.3.5 FC_ProgramBlockProtectState

プロテクトの設定

**関数のプロトタイプ宣言:**
FC_Result
FC_ProgramProtectState(uint8_t ***BlockNum***)

**引数**:
***BlockNum***: 以下からブロック番号を選択します。
- **FC_BLOCK_0**: block 0
- **FC_BLOCK_1**: block 1
- **FC_BLOCK_2:** block 2

- **FC_BLOCK_3**: block 3
- **FC_BLOCK_4:** block 4
- **FC_BLOCK_5:** block 5

**機能:**
各ブロックに対応するビットを"1"にするとプロテクトが設定されます。プロテクト状態の時には、データの書き込とチップ消去はできません。

**戻り値:**
プロテクトビット設定の結果
**FC_SUCCESS**:プロテクト設定成功
**FC_ERROR_PROTECTED**:プロテクト設定済
**FC_ERROR_OVER_TIME**: タイムアウトエラー

#### 5.2.3.6 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***: 以下からグループを選択します。
- **FC_BLOCK_GROUP_1**: ブロック 4、5
- **FC_BLOCK_GROUP_0**: ブロック 0～3

**機能:**
指定されたグループに対するブロックのプロテクトビットを消去します。

**戻り値:**
プロテクトビット消去の結果
**FC_SUCCESS**: プロテクト消去成功
**FC_ERROR_OVER_TIME**: タイムアウトエラー

#### 5.2.3.7 FC_WritePage

データの書き込み

**関数のプロトタイプ宣言:**
FC_Result
FC_WritePage(uint32_t ***PageAddr***, uint32_t * ***Data***)

**引数**:
***PageAddr***:書き込み対象ページの開始アドレス
***Data***: データバッファポインタ。サイズは 256 バイト固定

**機能:**
データをページ単位で書き込みます。TMPM37x は 64word/ページです。
Flash 書き込み前にデータ消去されている必要があります。
**(注)** データ消去しない Flash 書き込みは、MCU にダメージを与えます。

**戻り値:**

データ書き込み結果
**FC_SUCCESS**: 書き込み成功
**FC_ERROR_PROTECTED**: プロテクト状態のため書き込み失敗。
**FC_ERROR_OVER_TIME**: タイムアウトエラー

#### 5.2.3.8 FC_EraseBlock

データの消去

**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

**機能:**
ブロック消去を行います。プロテクトが設定されている場合はブロック消去されません。

**戻り値:**
ブロック消去結果
**FC_SUCCESS**: ブロック消去成功
**FC_ERROR_PROTECTED**: プロテクト状態のためブロック消去失敗
**FC_ERROR_OVER_TIME**: タイムアウトエラー

#### 5.2.3.9 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。プロテクト状態のブロックがある場合はプロテクト状態のブロック以外が消去されます。

**戻り値:**
チップ消去結果
**FC_SUCCESS**: チップ消去成功。ただし、プロテクト状態のブロックがある場合は、プロテクト状態のブロック以外が消去されます。
**FC_ERROR_PROTECTED**: 全ブロックがプロテクト状態のためチップ消去失敗
**FC_ERROR_OVER_TIME**: タイムアウトエラー

### 5.2.4 データ構造

なし。

# 6. GPIO

## 6.1 概要

TMPM37x 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、オープンドレイン、CMOS などを設定します。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37**x**_gpio.c
/Libraries/TX03_Periph_Driver/inc/tmpm37**x**_gpio.h

## 6.2 API 関数

### 6.2.1 関数一覧

- uint8_t GPIO_ReadData(GPIO_Port **GPIO**_**x**);
- uint8_t GPIO_ReadDataBit(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***) ;
- void GPIO_WriteData(GPIO_Port ***GPIO**_**x***, uint8_t ***Data***) ;
- void GPIO_WriteDataBit(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***, uint8_t ***BitValue***) ;
- void GPIO_Init(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***,
  GPIO_InitTypeDef ***** ***GPIO_InitStruct***);
- void GPIO_SetOutput(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***);
- void GPIO_SetInput(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***);
- void GPIO_SetOutputEnableReg(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***,
  FunctionalState ***NewState***);
- void GPIO_SetInputEnableReg(GPIO_Port **GPIO**_**x**, uint8_t **Bit**_**x**,
  FunctionalState ***NewState***);
- void GPIO_SetPullUp(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***,
  FunctionalState ***NewState*** );
- void GPIO_SetPullDown(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***,
  FunctionalState ***NewState***);
- void GPIO_SetOpenDrain(GPIO_Port ***GPIO**_**x***, uint8_t ***Bit**_**x***,
  FunctionalState ***NewState***);
- void GPIO_EnableFuncReg(GPIO_Port ***GPIO**_**x***, uint8_t ***FuncReg**_**x***, uint8_t ***Bit**_**x***);
- void GPIO_DisableFuncReg(GPIO_Port ***GPIO**_**x***, uint8_t ***FuncReg**_**x***, uint8_t ***Bit**_**x***);

### 6.2.2 関数の種類

上記関数は 3 種類に分類できます。

1) 入出力ポートへの書き込み/読み出し
   GPIO_ReadData(), GPIO_ReadDataBit(), GPIO_WriteData() and GPIO_WriteDataBit().
2) 入出力ポートの初期化と設定
   GPIO_SetOutput(), GPIO_SetInput(),GPIO_SetOutputEnableReg(),

GPIO_SetInputEnableReg(), GPIO_SetPullUp(),GPIO_SetPullDown(),
GPIO_SetOpenDrain() and GPIO_Init().

3) その他
GPIO_EnableFuncReg() and GPIO_DisableFuncReg()

## 6.2.3 関数仕様

### 6.2.3.1 GPIO_ReadData

DATA 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***: 以下から、GPIO ポートを選択します。

- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

**機能:**

DATA レジスタを読み込みます。

**戻り値:**

DATA レジスタの値

### 6.2.3.2 GPIO_ReadDataBit

ビット単位での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_ReadDataBit(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***)

**引数:**

***GPIO_x***: 以下から、GPIO ポートを選択します。

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

- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***Bit_x***: ビット番号を選択します。
- **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:**
DATA データレジスタのビット値:
- **GPIO_BIT_VALUE_0**: 値 0
- **GPIO_BIT_VALUE_1**: 値 1

### 6.2.3.3 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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***Data***: DATA レジスタに書き込むデータを指定します。

**機能:**
DATA レジスタにデータを書き込みます。

**戻り値:**
なし

### 6.2.3.4 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***: 以下から、GPIO ポートを選択します。

- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***Bit_x***: 以下から、GPIO ビット番号を選択します。

- **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 ビットに書き込む値

- **GPIO_BIT_VALUE_0**: 値 0
- **GPIO_BIT_VALUE_1**: 値 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

### 6.2.3.5 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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

***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。**GPIO_SetOutput(), GPIO_SetInput(), GPIO_SetPullUP() and GPIO_SetOpenDrain()** は本関数で呼び出されます。

**戻り値:**
なし

### 6.2.3.6 GPIO_SetOutput
ポートの出力設定

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G

- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

**機能:**
出力ポートに設定します。

**戻り値:**
なし

### 6.2.3.7 GPIO_SetInput
ポートの入力設定

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

**機能:**
入力ポートに設定します。

**戻り値:**
なし

### 6.2.3.8 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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

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

**機能:**
出力ポートの許可/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

### 6.2.3.9 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***: 以下から、GPIO ポートを選択します。

- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

***NewState***:

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

**機能:**

入力ポートの許可/禁止設定を行います。
***NewState*** が **ENABLE** の時、入力許可、
***NewState*** が **DISABLE** の時、入力禁止。.

**戻り値:**

なし

### 6.2.3.10 GPIO_SetPullUp

ポートのプルアップ設定

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

***NewState***:
- **ENABLE:** プルアップする
- **DISABLE:** プルアップしない

**機能:**
ポートのプルアップ設定を行います。
***NewState*** が **ENABLE** の時、プルアップを行います。
***NewState*** が **DISABLE** の時、プルアップを行いません。

**戻り値:**
なし

### 6.2.3.11 GPIO_SetPullDown
ポートのプルダウン設定

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B

- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

***NewState***:
- **ENABLE:** プルダウンする
- **DISABLE:** プルダウンしない

**機能:**
ポートのプルダウン設定を行います。
***NewState*** が **ENABLE** の時、プルアップを行います。
***NewState*** が **DISABLE** の時、プルアップを行いません。

**戻り値:**
なし

### 6.2.3.12 GPIO_SetOpenDrain

ポートの CMOS/オープンドレイン設定

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J

- **GPIO_PK:** GPIO port K

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:** 全 GPIO 端子の設定可能

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

**機能:**
ポートの CMOS/オープンドレイン設定を行います。
***NewState*** が **ENABLE** の時、指定された GPIO 端子をオープンドレインに設定
***NewState*** が **DISABLE** の時、指定された GPIO 端子を CMOS に設定。

**戻り値:**
なし

### 6.2.3.13 GPIO_EnableFuncReg

ポートの機能設定

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***FuncReg_x***: 以下から、GPIO ファンクションレジスタを選択します。
- **GPIO_FUNC_REG_1:** GPIO ファンクションレジスタ 1
- **GPIO_FUNC_REG_2:** GPIO ファンクションレジスタ 2
- **GPIO_FUNC_REG_3:** GPIO ファンクションレジスタ 3

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:**
ポートの機能設定を行います。

**戻り値:**
なし

### 6.2.3.14 GPIO_DisableFuncReg

ポートの機能設定解除

**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***: 以下から、GPIO ポートを選択します。
- **GPIO_PA:** GPIO port A
- **GPIO_PB:** GPIO port B
- **GPIO_PC:** GPIO port C
- **GPIO_PD:** GPIO port D
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PH:** GPIO port H
- **GPIO_PI :** GPIO port I
- **GPIO_PJ:** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PL:** GPIO port L

***FuncReg_x***: 以下から、GPIO ファンクションレジスタを選択します。
- **GPIO_FUNC_REG_1** GPIO ファンクションレジスタ 1,
- **GPIO_FUNC_REG_2** GPIO ファンクションレジスタ 2,
- **GPIO_FUNC_REG_3** GPIO ファンクションレジスタ 3.

***Bit_x***: 以下から、GPIO ビット番号を選択します。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:**
ポートの機能設定を解除します。

**戻り値:**
なし

## 6.2.4 データ構造

### 6.2.4.1 GPIO_InitTypeDef

**メンバ:**

uint8_t
**IOMode** 指定された GPIO 端子を入力ポートまたは出力ポートに設定。以下から選択。

- **GPIO_INPUT:** GPIO 端子を入力ポートに設定
- **GPIO_OUTPUT:** GPIO 端子を出力ポートに設定
- **GPIO_IO_MODE_NONE:** 入出力モードを変更しない

uint8_t
**PullUp** GPIO 端子のプルアップを設定

- **GPIO_PULLUP_ENABLE :** 指定された GPIO 端子をプルアップ
- **GPIO_PULLUP_DISABLE:** 指定された GPIO 端子をプルアップ禁止
- **GPIO_PULLUP_NONE:** プルアップ機能がない、または設定変更の必要がない

uint8_t
**OpenDrain** 指定された GPIO 端子のオープンドレイン、CMOS ポート設定

- **GPIO_OPEN_DRAIN_ENABLE:** 指定された GPIO 端子をオープンドレイン設定
- **GPIO_OPEN_DRAIN_DISABLE:** 指定された GPIO 端子を CMOS ポート設定
- **GPIO_OPEN_DRAIN_NONE:** プルアップ機能がない、または設定変更の必要がない

uint8_t
**PullDown** GPIO 端子のプルダウンを設定

- **GPIO_PULLDOWN_ENABLE:** 指定された GPIO 端子をプルダウン
- **GPIO_PULLDOWN_DISABLE:** 指定された GPIO 端子をプルダウン禁止
- **GPIO_PULLDOWN_NONE:** プルアップ機能がない、または設定変更の必要がない

# 7. OFD

## 7.1 概要

本デバイスは周波数検知回路(OFD)を内蔵しています。この回路は、クロックの異常状態や停止状態を検出するとリセットを発生する回路です。

OFD ドライバ API は、OFD 動作のの許可/禁止、検知周波数設定、OFD 回路の状態の取得などを行う関数セットです。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37**x**_ofd.c
/Libraries/TX03_Periph_Driver/inc/tmpm37**x**_ofd.h

**補足:** TMPM372/M373/M374 を TMPM37x と記載します。

## 7.2 API 関数

### 7.2.1 関数一覧

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

### 7.2.2 関数の種類

- OFD の初期化と設定
  OFD_SetRegWriteMode(), OFD_SetDetectionFrequency(),OFD_Enable(), OFD_Disable()

### 7.2.3 関数仕様

#### 7.2.3.1 OFD_SetRegWriteMode

OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF の書き込みのイネーブル / ディセーブル設定.

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

**引数:**
***NewState*** :
OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF レジスタの書き込みステータスです。下記のいずれかの値を選択します。

- **ENABLE:**

OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF レジスタに書き込み許可

- **DISABLE:**
  OFDCR2/OFDMNPLLON/OFDMNPLLOFF/OFDMXPLLON/OFDMXPLLOFF レジスタに書き込み禁止

**機能:**
***NewState*** が **ENABLE** の時に、OFDCR2 / OFDMNPLLON / OFDMNPLLOFF / OFDMXPLLON / OFDMXPLLOFF レジスタの書き込みを許可し、**DISABLE** の時に書き込みを禁止します。

**戻り値:**
なし

### 7.2.3.2 OFD_Enable

周波数検知回路(OFD) をイネーブルに設定します。

**関数のプロトタイプ宣言:**
void
OFD_Enable(void)

**引数:**
なし.

**機能:**
周波数検知回路 (OFD) 機能を許可します。

**戻り値:**
なし

### 7.2.3.3 OFD_Disable

周波数検知回路(OFD) をディセーブルに設定します。

**関数のプロトタイプ宣言:**
void
OFD_Disable(void)

**引数:**
なし.

**機能:**
周波数検知回路 (OFD) 機能を禁止します。

**戻り値:**
なし

#### 7.2.3.4 OFD_SetDetectionFrequency

検出周波数のカウント値を設定

**関数のプロトタイプ宣言:**
void
OFD_SetDetectionFrequency(OFD_PLL_State **State**,
uint32_t ***HigherDetectionCount***,
uint32_t ***LowerDetectionCount***)

**引数:**
***State***: 以下のいずれかの PLL の状態を選択します。
- **OFD_PLL_ON:** CG PLL ON 時
- **OFD_PLL_OFF:** CG PLL OFF 時

***HigherDetectionCount***: 検出周波数の上限値を設定します。
***LowerDetectionCount***: 検出周波数の下限値を設定します。

**機能:**
PLL OFF 時、または PLL ON 時の検出周波数上限値、検出周波数下限値を設定します。

**戻り値:**
なし

### 7.2.4 データ構造

なし

# 8. TMRB

## 8.1 概要

本デバイスは、8 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 ～TMRB7) を内蔵しています。各チャ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 チャネルの出力設定可能)

また、キャプチャ機能を利用することで、次のような用途に使用することができます。
- 周波数測定
- パルス幅測定
- 時間差測定

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37**x**_tmrb.c(*),
/Libraries/TX03_Periph_Driver/inc/tmpm37**x**_tmrb.h(*)

**補足:** TMPM376/M377 を TMPM37x と記載します。

## 8.2 API 関数

### 8.2.1 関数一覧

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

## 8.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

1) 各タイマの設定
   TMRB_Enable(), TMRB_Disable(), TMRB_Init(), TMRB_SetRunState(), TMRB_ChangeLeadingTiming(), TMRB_ChangeTrailingTiming().
2) キャプチャ機能の設定
   TMRB_SetCaptureTiming(), TMRB_ExecuteSWCapture().
3) ステータスの確認
   TMRB_GetINTFactor(), TMRB_GetUpCntValue(), TMRB_GetCaptureValue().
4) その他
   TMRB_SetFlipFlop(), TMRB_SetINTMask(), TMRB_SetIdleMode(), TMRB_SetDoubleBuf(), TMRB_SetExtStartTrg()

## 8.2.3 関数仕様

**補足**: 引数に記述されている “TSB_TB_TypeDef* ***TBx***” は下記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**

### 8.2.3.1 TMRB_Enable

TMRB チャネルの起動を行う。

**関数のプロトタイプ宣言:**
void
TMRB_Enable(TSB_TB_TypeDef* ***TBx***)

**引数:**
***TBx***: TMRB チャネルを選択します。

**機能:**
***TBx*** により選択された TMRB チャネルを有効にします。

**戻り値:**
なし

### 8.2.3.2 TMRB_Disable

TMRB チャネルの終了を行う。

**関数のプロトタイプ宣言:**
void
TMRB_Disable(TSB_TB_TypeDef* ***TBx***)

**引数:**
***TBx*:** TMRB チャネルを選択します。

**機能:**
***TBx*** により選択された TMRB チャネルを無効にします。

**戻り値:**
なし

### 8.2.3.3 TMRB_SetRunState

TMRB チャネルのカウンタ開始/停止の設定を行う。

**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 チャネルを選択します。
***Cmd***:以下からアップカウンタの動作状態を選択します。

- **TMRB_RUN**:動作開始
- **TMRB_STOP**:停止

**機能:**

***Cmd*** が **TMRB_RUN** の時、指定された TMRB チャネルのアップカウンタがカウントを開始。***Cmd*** が **TMRB_STOP** の時、アップカウンタはカウントを停止(同時にカウンタのクリア)。

**戻り値:**

なし

### 8.2.3.4 TMRB_Init

TMRB チャネルの初期化を行う。

**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 チャネルを選択します。
***InitStruct***:カウントモード、ソースクロック分周、デューティ値、サイクル値、アップカウンタ動作モード。(詳細はデータ構造を参照)。

**機能:**

本関数は、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

### 8.2.3.5 TMRB_SetCaptureTiming

キャプチャタイミングの設定を行う。

**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 チャネルを選択します。
**TSB_TB1, TSB_TB2, TSB_TB3, TSB_TB5, TSB_TB6, TSB_TB7**

***CaptureTiming*:** 以下からキャプチャタイミングを選択します。

- **TMRB_DISABLE_CAPTURE**: 指定された TMRB チャネルのキャプチャ機能を無効に設定します。
- **TMRB_CAPTURE_IN_RISING**: TMRB チャネルの端子入力に立ち上がりでキャプチャレジスタにカウンタ値を取り込みます。
- **TMRB_CAPTURE_IN_RISING_FALLING**: TBxIN 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、TBxIN 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。

**機能:**

***CaptureTiming*** が **TMRB_CAPTURE_IN_RISING** に設定されている場合、指定された TMRB チャネルの TBxIN 端子入力の立ち上がりで、アップカウンタのカウンタ値をキャプチャレジスタ 0(TBxCP0)に取り込みます。
***CaptureTiming*** が **TMRB_CAPTURE_IN_RISING_FALLING** に設定されている場合、指定された TMRB チャネルの TBxIN 端子入力の立ち上がりで、アップカウンタのカウンタ値をキャプチャレジスタ 0(TBxCP0)に取り込みます。また、指定された TMRB チャネルの TBxIN 端子入力の立ち下がりで、アップカウンタのカウンタ値をキャプチャレジスタ 1(TBxCP1)に取り込みます。
TMRB2、TMRB5、TMRB7 のフリップフロップ出力は他チャンネルのキャプチャトリガとして使用可能です。

**TMRB3~5: TB2OUT**
**TMRB6~7: TB5OUT**
**TMRB0~2: TB7OUT**

**戻り値:**

なし

### 8.2.3.6 TMRB_SetFlipFlop

フリップフロップ機能の設定を行います。

**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 チャネルを選択します。
***FFStruct***: TMRB のフリップフロップ機能(フリップフロップ出力レベル、フリップフロップ反転トリガ)に関する構造体です。(詳細はデータ構造 を参照)

**機能:**

本関数は、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

### 8.2.3.7 TMRB_GetINTFactor

割り込み要因の取得を行います。

**関数のプロトタイプ宣言:**

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

**引数:**

***TBx***: TMRB チャネルを選択します。

**機能:**

本関数は ISR で使用され、割り込み要因を取得します。**MatchLeadingTiming** ビットはアップカウンタ値のデューティ値との一致を、**MatchTrailingTiming** ビットは、デューティ値のサイクル値との一致を、**Overflow** ビットは割り込み前にオーバーフローが起きたことを表します。

**戻り値:**

TMRB 割り込み要因
**MatchLeadingTiming**(Bit0): デューティ値との一致検出
**MatchTrailingTiming**(Bit1): サイクル値との一致検出
**OverFlow**(Bit2): アップカウンタのオーバーフロー

**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
}
```

### 8.2.3.8 TMRB_SetINTMask

TMRB 割り込みマスクの設定を行います。

**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 チャネルを選択します。
***INTMask***: 以下から割り込みマスクを選択します。

- **TMRB_MASK_MATCH_TRAILINGTIMING_INT**: アップカウンタ値とサイクル値が一致した場合の割り込みをマスクします。
- **TMRB_MASK_MATCH_LEADINGTIMING_INT**: アップカウンタ値とデューティ値が一致した場合の割り込みをマスクします。
- **TMRB_MASK_OVERFLOW_INT**: オーバフロー発生時の割り込みをマスクします。
- **TMRB_NO_INT_MASK**: 割り込みをマスクしません。

**機能:**

**TMRB_MASK_MATCH_TRAILINGTIMING_INT** 選択時、アップカウンタ値とサイクル値が一致した場合、割り込みは発生しません。
**TMRB_MASK_MATCH_LEADINGTIMING_INT** 選択時、アップカウンタ値とデューティ値が一致した場合、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

### 8.2.3.9 TMRB_ChangeLeadingTiming

指定された TMRB チャネルのデューティの変更を行います。

**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 チャネルを選択.
***LeadingTiming*** : デューティを指定（最大値 0xFFFF）

**機能:**

本関数は、指定されたTMRBの絶対値を設定します。実際のデューティのインターバルは、CGの校正と ***ClkDiv*** の値によります。(詳細はデータ構造 を参照)

**戻り値:**

なし

**補足:**

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

### 8.2.3.10 TMRB_ChangeTrailingTiming

指定された TMRB チャネルの周期の変更を行います。

**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 チャネルを選択します。
***TrailingTiming***:サイクルを指定します(最大値 0xFFFF)

**機能:**
本関数は、周期を変更します。実際の周期は、CG の構成と、***ClkDiv*** の値によります。
(詳細はデータ構造 を参照)

**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。TBxRG0/1 値は、PPG モードで TBxRG0 < TBxRG1 の関係で設定します。

### 8.2.3.11 TMRB_GetUpCntValue

指定された TMRB チャネルのアップカウンタ値の読み込みを行います。

**関数のプロトタイプ宣言:**
```
uint16_t
TMRB_GetUpCntValue(TSB_TB_TypeDef* TBx)
```

**引数:**
***TBx:***指定された TMRB チャネル

**機能:**
本関数は、指定された TMRB チャネルの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

### 8.2.3.12 TMRB_GetCaptureValue

指定された TMRB チャネルのキャプチャレジスタ 0 の読み込みを行います。

**関数のプロトタイプ宣言:**
```
uint16_t
TMRB_GetCaptureValue(TSB_TB_TypeDef* TBx,
                     uint8_t CapReg)
```

**引数:**
***TBx***:TMRB チャネルを選択します。
***CapReg***:以下から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** の時、キャプチャレジスタ 0 の値を読み込み、***CapReg*** が **TMRB_CAPTURE_1** の時、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

### 8.2.3.13 TMRB_ExecuteSWCapture

指定された TMRB チャネルのソフトウェアキャプチャの実行を行います。

**関数のプロトタイプ宣言:**
void
TMRB_ExecuteSWCapture(TSB_TB_TypeDef* ***TBx***)

**引数:**
***TBx***: TMRB チャネルを選択します。

**機能:**
本関数は、指定された TMRB チャネルのソフトウェアキャプチャを実行し、キャプチャレジスタ 0 に保存します。

**戻り値:**
なし

### 8.2.3.14 TMRB_SetIdleMode

指定された TMRB チャネルのアイドルモード時の動作設定を行います。

**関数のプロトタイプ宣言:**
void
TMRB_SetIdleMode(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***)

**引数:**
***TBx***: TMRB チャネルを選択します。
***NewState***: 以下から、Idle モード時の動作の有効/無効を選択します。
- **ENABLE** : 有効
- **DISABLE** : 無効

**機能:**
***NewState*** が **ENABLE** の時は、システムが Idle モードでも指定された TMRB チャネルは動作します。**DISABLE** 時はシステムが Idle モード時に動作を停止します。

**戻り値:**
なし

### 8.2.3.15 TMRB_SetDoubleBuf

指定された TMRB チャネルのダブルバッファの設定を行います。

**関数のプロトタイプ宣言:**
void
TMRB_SetDoubleBuf(TSB_TB_TypeDef* ***TBx***,
FunctionalState ***NewState***, uint8_t ***WriteRegMode***)

**引数:**
***TBx***: TMRB チャネルを選択します。

***NewState*** : 以下からダブルバッファの有効/無効を選択します。

- **ENABLE** : ダブルバッファ有効
- **DISABLE**: ダブルバッファ無効

***WriteRegMode*** は、ダブルバッファが有効なときの タイマレジスタ 0、タイマレジスタ 1 に書き込むタイミングを設定します。

- **TMRB_WRITE_REG_SEPARATE:** タイマレジスタ 0、タイマレジスタ 1 は個別に書き込みが可能です。1 つのレジスタにのみ書き込み準備が行われていても、可能です。
- **TMRB_WRITE_REG_SIMULTANEOUS:** タイマレジスタ 0、タイマレジスタ 1 両方に書き込み準備ができていない場合、書き込みを行うことはできません。

**機能:**

TBxRG0 (***LeadingTiming***)、TBxRG1 (***TrailingTiming***)、バッファは同じアドレスに割り当てられます。ダブルバッファが無効の場合、同じ値がレジスタとバッファに書き込まれます。ダブルバッファが有効の場合は、各バッファにのみ書き込まれるので、TBxRG0 (***LeadingTiming***)と TBxRG1 (***TrailingTiming***)に初期値を書き込むために、ダルバッファを無効にしてください。その後、ダブルバッファを有効にすれば、一致割り込み発生時に自動的にデータがロードされます。

**戻り値:**

なし

### 8.2.3.16 TMRB_SetExtStartTrg

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。

**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 チャネルを選択します。

***NewState*** : 外部トリガの使用有無を選択します。

- **ENABLE:** 外部トリガ使用
- **DISABLE:** ソフトウェアトリガ使用

***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:** 外部トリガの立ち上がりエッジ
- **TMRB_TRG_EDGE_FALLING:** 外部トリガの立下りエッジ

**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。

**戻り値:**

なし

## 8.2.4 データ構造

### 8.2.4.1 TMRB_InitTypeDef

**メンバ:**

uint32_t
**Mode** 以下からタイマモードを選択してください。

- **TMRB_INTERVAL_TIMER**: インターバルタイマモード
- **TMRB_EVENT_CNT**: イベントカウンタモード

uint32_t
**ClkDiv** : 以下から、インターバルタイマのソースクロックの分周を選択してください。

- **TMRB_CLK_DIV_2**:fperiph 2
- **TMRB_CLK_DIV_8**:fperiph 8
- **TMRB_CLK_DIV_32**:fperiph 32

uint32_t
**TrailingTiming**: TBnRG1 へ書き込む周期 (最大 0xFFFF)

uint32_t
**UpCntCtrl**:以下からアップカウンタの動作を選択してください。

- **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**:周期が一致したときに、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)

### 8.2.4.2 TMRB_FFOutputTypeDef

**メンバ:**

uint32_t
**FlipflopCtrl** : 以下から、フリップフロップのレベルを選択します。

- **TMRB_FLIPFLOP_INVERT**: 反転(ソフト反転)します。
- **TMRB_FLIPFLOP_SET**: “1” にセットします。
- **TMRB_FLIPFLOP_CLEAR:** “0” にクリアします。

uint32_t
**FlipflopReverseTrg** : 以下から、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_TRAILINGTIMING**: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_LEADINGTIMING**: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。

### 8.2.4.3 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　-

# 9. SIO/UART

## 9.1 概要

TMPM370 と TMPM372 は 4 本(SC0~SC3)、TMPM373 と TMPM374 は 3 本(SC0~SC2)のシリアル I/O チャネルを持っています。それぞれのチャネルは I/O インタフェースモード(同期通信)と 7, 8, 9 ビット長の UART モード(非同期通信)を実装しています。9 ビット UART モードでは、シリアルリンク(マルチコントローラ・システム) でマスタコントローラがスレーブコントローラを起動するときにウェイクアップ機能が使用されます。

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

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37**x**_uart.c(*),
/Libraries/TX03_Periph_Driver/inc/tmpm37**x**_uart.h(*)

## 9.2 API 関数

### 9.2.1 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_TRxDisable TrxAutoDisable
- void UART_RxFIFOINTCtrl(TSB_SC_TypeDef * UARTx, FunctionalState NewState)
- void UART_TxFIFOINTCtrl(TSB_SC_TypeDef * UARTx, FunctionalState NewState)
- void UART_RxFIFOByteSel(TSB_SC_TypeDef * UARTx, uint32_t BytesUsed)
- void UART_RxFIFOFillLevel(TSB_SC_TypeDef * UARTx,uint32_t RxFIFOLevel)
- void UART_RxFIFOINTSel(TSB_SC_TypeDef * UARTx, uint32_t RxINTCondition)
- void UART_RxFIFOClear(TSB_SC_TypeDef * UARTx)
- void UART_TxFIFOFillLevel(TSB_SC_TypeDef * UARTx, uint32_t TxFIFOLevel)
- void UART_TxFIFOINTSel(TSB_SC_TypeDef * UARTx, uint32_t TxINTCondition)
- void UART_TxFIFOClear(TSB_SC_TypeDef * UARTx)
- uint32_t UART_GetRxFIFOFillLevelStatus(TSB_SC_TypeDef * UARTx)

- uint32_t UART_GetRxFIFOOverRunStatus(TSB_SC_TypeDef * UARTx)
- uint32_t UART_GetTxFIFOFillLevelStatus(TSB_SC_TypeDef * UARTx)
- uint32_t UART_GetTxFIFOUnderRunStatus(TSB_SC_TypeDef * UARTx)
- void SIO_Enable(TSB_SC_TypeDef * SIOx)
- void SIO_Disable(TSB_SC_TypeDef * SIOx)
- uint8_t SIO_GetRxData(TSB_SC_TypeDef * SIOx)
- void SIO_SetTxData(TSB_SC_TypeDef * SIOx, uint8_t Data)
- void SIO_Init(TSB_SC_TypeDef * SIOx, uint32_t IOClkSel, SIO_InitTypeDef * InitStruct)

## 9.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

1) 初期化と設定
   UART_Enable(), UART_Disable(), UART_Init(), UART_DefaultConfig(),
   SIO_Enable(), SIO_Disable(),SIO_,SIO_Init()
2) 送受信設定とエラー確認
   UART_GetBufState(), UART_GetRxData(), UART_SetTxData() and
   UART_GetErrState(), SIO_GetRxData(), SIO_SetTxData()
3) その他
   UART_SWReset(), UART_SetWakeUpFunc(), UART_SetIdleMode()
4) FIFO モードの設定:
   UART_TrxAutoDisable(), UART_RxFIFOINTCtrl(), UART_TxFIFOINTCtrl(),
   UART_RxFIFOByteSel(), UART_RxFIFOFillLevel(), UART_RxFIFOINTSel(),
   UART_RxFIFOClear(), UART_TxFIFOFillLevel(), UART_TxFIFOINTSel(),
   UART_TxFIFOClear(), UART_GetRxFIFOFillLevelStatus(),
   UART_GetRxFIFOOverRunStatus(), UART_GetTxFIFOFillLevelStatus(),
   UART_GetTxFIFOUnderRunStatus()

## 9.2.3 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。
**TMPM370, TMPM372: UART0**, **UART1**, **UART2**, **UART3**
**TMPM373, TMPM374: UART0**, **UART1**, **UART2**
引数に記述している“TSB_SC_TypeDef* ***SIOx***” は、以下から選択してください。
**TMPM370, TMPM372: SIO0**, **SIO1**, **SIO2**, **SIO3**
**TMPM373, TMPM374: SIO0**, **SIO1**, **SIO2**

### 9.2.3.1 UART_Enable

UART チャネルの起動を行います。

**関数のプロトタイプ宣言:**
void UART_Enable(TSB_SC_TypeDef* ***UARTx***)

**引数:**
***UARTx***:UART チャネルを選択します。

**機能:**
***UARTx*** により 選択された ***UART*** チャネルを起動します。

**戻り値:**
なし

### 9.2.3.2 UART_Disable

UART チャネルの終了を行います。

**関数のプロトタイプ宣言:**
void UART_Disable(TSB_SC_TypeDef* ***UARTx***)

**引数:**
***UARTx***:UART チャネルを選択します。

**機能:**
***UARTx*** により 選択された UART チャネルを終了します。

**戻り値:**
なし

### 9.2.3.3 UART_GetBufState

送受信バッファの状態の読み込みを行います。

**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 チャネルを選択します。
***Direction:*** 以下から送信/受信を選択します。
- **UART_RX** :受信
- **UART_TX** :送信

**機能:**
**Direction** が **UART_RX** に指定されている時は、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** に指定されている時は、以下の受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**:バッファの読み込み/書き込みが可能です。
**BUSY**:転送中

### 9.2.3.4 UART_SWReset

指定された UART チャネルのソフトウェアリセットを行います。

**関数のプロトタイプ宣言:**
void UART_SWReset(TSB_SC_TypeDef* ***UARTx***)

**引数:**
***UARTx***: UART チャネルの選択

**機能:**
***UARTx*** によって選択された UART チャネルをリセットします。

**戻り値:**
なし

### 9.2.3.5 UART_Init

UART チャネルの初期化を行います。

**関数のプロトタイプ宣言:**
void UART_Init(TSB_SC_TypeDef* ***UARTx***,
UART_InitTypeDef* ***InitStruct***)

**引数:**
***UARTx***: チャネルを初期化します。
***InitStruct***: 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 UART の設定に関する構造体です。

(詳細はデータ構造 を参照)

**機能:**
本関数は、***UARTx*** によって選択される UART チャネルの初期化、設定 (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) を行います。

**戻り値:**
なし

### 9.2.3.6 UART_GetRxData

UART チャネルから受信データの読み込みを行います。

**関数のプロトタイプ宣言:**
uint32_t UART_GetRxData(TSB_SC_TypeDef* ***UARTx***)

**引数:**
***UARTx***: UART チャネルを選択します。

**機能:**
***UARTx*** により選択される UART チャネルからの受信データの読み込み。本関数は、**UART_GetBufState(*UARTx*, UART_RX)**で **DONE** が返されるか、UART (シリアルチャネル) 割り込み関数の中で実行してください。

**戻り値:**
受信データ

### 9.2.3.7 UART_SetTxData

UART チャネルからの送信データの設定を行います。

**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 チャネルを選択します。
***Data***:送信データ(7 ビット、8 ビット、9 ビット)

**機能:**
***UARTx*** により選択される UART チャネルから送信されるデータを設定します。本関数は、**UART_GetBufState(*UARTx,* UART_TX)**で **DONE** が返されるか、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

### 9.2.3.8 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 チャネルを選択します。

**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

### 9.2.3.9 UART_GetErrState

指定 UART チャネルからの転送エラーフラグの読み出しを行います。

**関数のプロトタイプ宣言:**
UART_Err UART_GetErrState(TSB_SC_TypeDef* ***UARTx***)

**引数:**
***UARTx*** :UART チャネルを選択します

**機能:**
本関数は、転送時にエラーが発生有無を確認し、その結果を返します。

**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

### 9.2.3.10 UART_SetWakeUpFunc

9 ビットモード時のウェイクアップ機能の設定を行います。

**関数のプロトタイプ宣言:**
void UART_SetWakeUpFunc(TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

**引数:**
***UARTx***:UART チャネルを選択します。
***NewState***:以下から、ウェイクアップ機能の有効/無効を選択します。

**ENABLE, DISABLE**

**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** の時、ウェイクアップ機能は有効に、
***NewState*** が **DISABLE** の時、ウェイクアップ機能は無効に設定されます。
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

### 9.2.3.11 UART_SetIdleMode

Idle モード時の UART チャネルの動作設定を行います。

**関数のプロトタイプ宣言:**
void UART_SetIdleMode(TSB_SC_TypeDef* ***UARTx***,
FunctionalState ***NewState***)

**引数:**
***UARTx*:**UART チャネルを選択します。
***NewState*** : 以下から、アイドルモード時の動作の有効/無効を選択します。

**ENABLE, DISABLE**

**機能:**
アイドルモード時の動作を設定します。
***NewState*** が **ENABLE** の時は有効、
***NewState*** が **DISABLE** の時は無効。

**戻り値:**
なし

### 9.2.3.12 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

### 9.2.3.13 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

### 9.2.3.14 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

### 9.2.3.15 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

### 9.2.3.16 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

### 9.2.3.17 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

### 9.2.3.18 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 チャネルを指定します。
***RxFIFOLevel***: 受信 FIFO の fill レベルを選択します。

| ***RxFIFOLevel*** | 半二重 | 全二重 |
|---|---|---|
| **UART_RXFIFO4B_FLEVLE_4_2B** | 4 バイト | 2 バイト |
| **UART_RXFIFO4B_FLEVLE_1_1B** | 1 バイト | 1 バイト |
| **UART_RXFIFO4B_FLEVLE_2_2B** | 2 バイト | 2 バイト |
| **UART_RXFIFO4B_FLEVLE_3_1B** | 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

### 9.2.3.19 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。

### 9.2.3.20 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

### 9.2.3.21 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 チャネルを指定します。
***TxFIFOLevel***: 受信 FIFO の fill レベルを選択します。

| ***TxFIFOLevel*** | 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

### 9.2.3.22 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

### 9.2.3.23 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

### 9.2.3.24 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

### 9.2.3.25 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

### 9.2.3.26 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

### 9.2.3.27 UART_GetTxFIFOUnderRunStatus

送信 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_ GetTxFIFOUnderRunStatus (TSB_SC_TypeDef* ***UARTx***);

**引数:**
***UARTx***: UART チャネルを指定します。

**機能:**
送信 FIFO オーバーラン状態を取得します。

**戻り値:**
**UART_TXFIFO_UNDERRUN:** オーバーラン発生

### 9.2.3.28 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 チャネルを指定します。

**機能:**
SIO 動作を許可します。

**戻り値:**
なし。

### 9.2.3.29 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 チャネルを指定します。

**機能:**
SIO 動作を禁止します。

**戻り値:**
なし

### 9.2.3.30 SIO_GetRxData

受信用バッファ

**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)

### 9.2.3.31 SIO_SetTxData

送信用バッファ

**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

#### 9.2.3.32 SIO_Init

SIO チャネルの初期化

**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

### 9.2.4 データ構造

#### 9.2.4.1 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
- **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
- **UART_STOP_BITS_1:** 1 ビット
- **UART_STOP_BITS_2 :** 2 ビット

uint32_t
***Parity*** : パリティを以下から選択
- **UART_NO_PARITY :** パリティなし
- **UART_EVEN_PARITY**: 偶数(Even) パリティ
- **UART_ODD_PARITY**: 偶数(Even) パリティ

uint32_t

***Mode***: 以下から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 演算子によって接続して指定してください。

- **UART_ENABLE_TX:** 送信許可
- **UART_ENABLE_RX:**受信許可

uint32_t
***FlowCtrl***：以下から、フローコントロールモードを選択します(**)。
- **UART_NONE_FLOW_CTRL** :CTS 無効

*: 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。
**:本バージョンのドライバでは、ハンドシェイク機能に対応していないため、CTSUART_NONE_FLOW_CTRL のみ選択できます。

### 9.2.4.2 SIO_InitTypeDef

**メンバ:**
uint32_t
***InputClkEdge***: 入力クロックエッジを選択します。"0"(SIO_SCLKS_TXDF_RXDR) のみ指定可能です。

uint32_t
***IntervalTime***: 連続転送時のインターバル時間を選択します。
- **SIO_SINT_TIME_NONE**: なし
- **SIO_SINT_TIME_SCLK_1**: 1*SCLK
- **SIO_SINT_TIME_SCLK_2**: 2*SCLK
- **SIO_SINT_TIME_SCLK_4**: 4*SCLK
- **SIO_SINT_TIME_SCLK_8**: 8*SCLK
- **SIO_SINT_TIME_SCLK_16**: 16*SCLK
- **SIO_SINT_TIME_SCLK_32**: 32*SCLK
- **SIO_SINT_TIME_SCLK_64**: 64*SCLK

uint32_t
***TransferMode***: 転送モードを選択します。
- **SIO_TRANSFER_PROHIBIT**: 転送禁止
- **SIO_TRANSFER_HALFDPX_RX**: 半二重(受信)
- **SIO_TRANSFER_HALFDPX_TX**: 半二重(送信)
- **SIO_TRANSFER_FULLDPX**: 全二重

uint32_t
***TransferDir***: 転送方向を選択します。
- **SIO_LSB_FRIST**: LSB FRIST
- **SIO_MSB_FRIST**: MSB FRIST

uint32_t
***Mode***: 送受信を制御します。有効ビットの組み合わせが可能です。
- **SIO_ENABLE_TX**: 送信許可
- **SIO_ENABLE_RX**: 受信許可

uint32_t
***DoubleBuffer***: ダブルバッファの許可/禁止を選択します。
- **SIO_WBUF_ENABLE**: 許可

- **SIO_WBUF_DISABLE**: 禁止

uint32_t
***BaudRateClock***: ボーレートジェネレータ入力クロックを選択します。
- **SIO_BR_CLOCK_T1**: φT1
- **SIO_BR_CLOCK_T4**: φT4
- **SIO_BR_CLOCK_T16**: φT16
- **SIO_BR_CLOCK_T64**: φT64

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

# 10. VLTD

## 10.1 概要

電圧検出回路は、電源電圧の低下を検出し、リセット信号を発生します。

VLTD ドライバ API は、VLTD 機能のイネーブル/ディセーブルの設定、検出電圧の設定、電源電圧の状態の取得を設定する関数セットです。

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37**x**_vltd.c(*),
/Libraries/TX03_Periph_Driver/inc/tmpm37**x**_vltd.h(*)

## 10.2 API 関数

### 10.2.1 関数一覧

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

### 10.2.2 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) 初期化と設定
   VLTD_Enable(), VLTD_Disable()
2) 検知電圧の選択
   VLTD_SetVoltage()

### 10.2.3 関数仕様

#### 10.2.3.1 VLTD_Enable

VLTD モジュールの有効設定を行います。

**関数のプロトタイプ宣言:**
void VLTD_Enable(void)

**引数:**
なし

**機能:**
LVTD モジュールを有効にします。

**戻り値:**
なし

#### 10.2.3.2 VLTD_Disable

VLTD モジュールを無効にします。

**関数のプロトタイプ宣言:**
void VLTD_Disable(void)

**引数:**
なし

**機能:**
LVTD モジュールを無効にします。

**戻り値:**
なし

#### 10.2.3.3 VLTD_SetVoltage

検知電圧を選択します。

**関数のプロトタイプ宣言:**
void VLTD_SetVoltage(uint32_t ***Voltage***)

**引数:**
***Voltage*:** 以下から検知電圧を選択します。

- **VLTD_DETECT_VOLTAGE_41:** 検知電圧 = 4.1V ± 0.2V
- **VLTD_DETECT_VOLTAGE_44:** 検知電圧 = 4.4V ± 0.2V
- **VLTD_DETECT_VOLTAGE_46:** 検知電圧 = 4.6V ± 0.2V

**機能:**
本関数は、検知電圧を設定します。

**戻り値:**
なし

### 10.2.4 データ構造

なし

# 11. WDT

## 11.1 概要

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

本ドライバは、以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpm37**x**_wdt.c(*),
/Libraries/TX03_Periph_Driver/inc/tmpm37**x**_wdt.h(*)

## 11.2 API 関数

### 11.2.1 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)

### 11.2.2 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・停止など
   WDT_SetIdleMode().

### 11.2.3 関数仕様

#### 11.2.3.1 WDT_SetDetectTime

WDT 検出時間の設定を行います。

**関数のプロトタイプ宣言:**
void WDT_SetDetectTime(uint32_t ***DetectTime***)

**引数:**
***DetectTime:*** 検出時間の設定は下記から選択されます。

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

**機能:**
WDT の検出時間を設定します。

**戻り値:**
なし

### 11.2.3.2 WDT_SetIdleMode

アイドルモード時の動作の設定を行います。

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

**引数:**
***NewState***: 以下から、アイドルモード時の動作の有効/無効を選択します。
- **ENABLE**: WDT カウンタの動作有効
- **DISABLE**. WDT カウンタの動作無効

**機能:**
本関数は、アイドルモード時の 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

### 11.2.3.3 WDT_SetOverflowOutput

カウンタオーバーフロー時の WDT 動作(NMI 割り込みを発生、またはリセット)の設定を行います。

**関数のプロトタイプ宣言:**
void WDT_SetOverflowOutput(uint32_t ***OverflowOutput***)

**引数:**
***OverflowOutput***: 以下から、カウンタオーバーフロー時の設定を選択します。
- **WDT_NMIINT**: NMI 割り込み発生
- **WDT_WDOUT**: リセット

**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***i が **WDT_NMIINT** の時、カウンタオーバーフローが発生すると NMI 割り込みが発生します。

**戻り値:**
なし

### 11.2.3.4 WDT_Init

WDT の初期化を行います。

**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

### 11.2.3.5 WDT_Enable

ウォッチドッグタイマ(WDT) の起動を行います。

**関数のプロトタイプ宣言:**
void WDT_Enable(void)

**引数:**
なし

**機能:**
本関数はウォッチドッグタイマ(WDT) を起動します。

**戻り値:**
なし

### 11.2.3.6 WDT_Disable

ウォッチドッグタイマ(WDT) の終了を行います。

**関数のプロトタイプ宣言:**
void WDT_Disable(void)

**引数:**
なし

**機能:**
ウォッチドッグタイマ(WDT) を終了します。

**戻り値:**
なし

#### 11.2.3.7 WDT_WriteClearCode

クリアコードの書き込みを行います。

**関数のプロトタイプ宣言:**
void WDT_WriteClearCode (void)

**引数:**
なし

**機能:**
WDT カウンタにクリアコードを書き込みます。

**戻り値:**
なし

### 11.2.4 データ構造

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

uint32_t
***OverflowOutput*** : 以下から、カウンタオーバーフロー時の設定を選択します。

- **WDT_WDOUT:** リセット
- **WDT_NMIINT:** NMI 割り込み

# 12. ENC

## 12.1 概要

本デバイスは、エンコーダ入力回路を内蔵しています(ENC)。インクリメンタルエンコーダの信号を直接入力し、モータの絶対位置を用意に得ることができます。

エンコーダ入力回路は、エンコーダモード、センサモード(2 種類)、タイマモードの 4 つの動作モードに対応しています。

- インクリメンタルエンコーダおよびホール IC センサ対応（センサ信号を直接入力可能）
- 汎用 24 ビットタイマ機能
- 4 逓倍 (6 逓倍) 回路内蔵
- 回転方向検出回路内蔵
- カウンタ (24 ビット) 内蔵
- コンペア許可／禁止設定可能
- 割り込み出力1本
- 入力信号についてデジタルノイズフィルタ内蔵

ENC ドライバ API は、各モジュールの設定機能を持ち、チャネル選択、モード設定、比較機能設定、ソフトウェアキャプチャ設定、ステータスリード、ENC カウント値の取得などの機能を提供します。

本ドライバは、以下のファイルで構成されています。
\Libraries\TX03_Periph_Driver\src\tmpm37x_enc.c
\Libraries\TX03_Periph_Driver\inc\tmpm37x_enc.h

補足: "**x**"は 0, 2, 3, 4 を指します。

## 12.2 TMPM370/2/3/4 の違い

TMPM370: 2 チャネル: ENC0, ENC1
TMPM372/3/4: 1 チャネル: ENC1

## 12.3 API 関数

### 12.3.1 関数一覧

◆ void ENC_Enable(TSB_EN_TypeDef * ***ENx***);
◆ void ENC_Disable(TSB_EN_TypeDef * ***ENx***);
◆ void ENC_Init(TSB_EN_TypeDef * ***ENx***, ENC_InitTypeDef * ***InitStruct***);
◆ void ENC_SetSWCapture(TSB_EN_TypeDef * ***ENx***, uint32_t ***ENC_Mode***);
◆ void ENC_ClearCounter (TSB_EN_TypeDef * ***ENx***);
◆ ENC_FlagStatus ENC_GetENCFlag (TSB_EN_TypeDef * ***ENx***);
◆ void ENC_SetCounterReload (TSB_EN_TypeDef *___ENx***, uint32_t ***ENC_Mode***,
uint32_t ***PeriodValue***);

◆ void ENC_SetCompareValue(TSB_EN_TypeDef * ***ENx***,uint32_t ***ENC_Mode***,
uint32_t ***CompareValue***);
◆ uint32_t ENC_GetCompareValue(TSB_EN_TypeDef * ***ENx***);
◆ uint32_t ENC_GetCounterValue(TSB_EN_TypeDef * ***ENx***);

## 12.3.2 関数の種類

上記関数は 3 つのグループに分けられます。

1) ENC の設定:
ENC_Init(), ENC_ClearCounter(), ENC_SetCounterReload(), ENC_SetSWCapture(), ENC_SetCompareValue().
2) ENC の許可/禁止:
ENC_Enable(), ENC_Disable()
3) ENC 状態、またはデータリード:
ENC_GetENCFlag(), ENC_GetCounterValue(), ENC_GetCompareValue()

## 12.3.3 関数仕様

**補足**: 下記の全 API において、パラメータ “TSB_EN_TypeDef * ***ENx***” は 以下のいずれかを選択してください。
TMPM370: ***EN0, EN1***
TMPM372/3/4: ***EN1***

### 12.3.3.1 ENC_Enable

エンコーダ動作の許可

**関数のプロトタイプ宣言:**
void
ENC_Enable(TSB_EN_TypeDef * ***ENx***)

**引数:**
***ENx:*** ENC チャネルを選択します。

**機能:**
エンコーダ動作を許可します。

**戻り値:**
なし

### 12.3.3.2 ENC_Disable

エンコーダ動作の禁止

**関数のプロトタイプ宣言:**
void
ENC_Disable(TSB_EN_TypeDef * ***ENx***)

**引数:**
***ENx:*** ENC チャネルを選択します。

**機能:**
エンコーダ動作を禁止します。

**戻り値:**
なし

#### 12.3.3.3 ENC_Init

エンコーダ動作の初期化

**関数のプロトタイプ宣言:**
void
ENC_Init(TSB_EN_TypeDef * ***ENx***, ENC_InitTypeDef * ***InitStruct***)

**引数:**
***ENx:*** ENC チャネルを選択します。

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

**機能:**
エンコーダ動作の初期設定を行います。

**戻り値:**
なし

#### 12.3.3.4 ENC_SetSWCapture

ソフトキャプチャの実行(タイマモード/センサモード (タイマカウント)時)

**関数のプロトタイプ宣言:**
void
ENC_SetSWCapture(TSB_EN_TypeDef * ***ENx***, uint32_t ***ENC_Mode***)

**引数:**
***ENx:*** ENC チャネルを選択します。

***ENC_Mode:*** 以下から、エンコーダ動作モードを選択します。
- **ENC_TIMER_MODE**: タイマモード
- **ENC_SENSOR_TIME_MODE**: センサモード

**機能:**
ソフトキャプチャの実行を行います。

**戻り値:**
なし

#### 12.3.3.5 ENC_ClearCounter

エンコーダパルスカウンタクリア

**関数のプロトタイプ宣言:**
void
ENC_ClearCounter(TSB_EN_TypeDef * ***ENx***)

**引数:**
***ENx:*** ENC チャネルを選択します。

**機能:**
エンコーダパルスカウンタをクリアします。

**戻り値:**
なし

### 12.3.3.6 ENC_GetENCFlag

エンコーダコンペアフラグ/反転エラーフラグ/ Z 相通過検出/エンコーダ回転方向の取得

**関数のプロトタイプ宣言:**
ENC_FlagStatus
ENC_GetENCFlag (TSB_EN_TypeDef * ***ENx***)

**引数:**
***ENx:*** ENC チャネルを選択します。

**機能:**
エンコーダコンペアフラグ/反転エラーフラグ/ Z 相通過検出/エンコーダ回転方向を取得します。各フラグの意味については、MCU データシートを参照してください。

**戻り値:**
エンコーダフラグです。
***ZPhaseDetectFlag***(bit12): Z 相通過検出
***RotationDirection*** (bit13): エンコーダ回転方向
***ReverseErrorFlag*** (bit14): 反転エラーフラグ
***CompareFlag*** (bit15): エンコーダコンペアフラグ

### 12.3.3.7 ENC_SetCounterReload

エンコーダカウンタの周期設定

**関数のプロトタイプ宣言:**
void
ENC_SetCounterReload (TSB_EN_TypeDef * ***ENx***, uint32_t ***PeriodValue***)

**引数:**
***ENx:*** ENC チャネルを選択します。

***PeriodValue***: エンコーダカウンタの周期を選択します。値は **0x0000 ～ 0xFFFF** まで選択可能です。

**機能:**
エンコーダカウンタの周期を設定します。

**戻り値:**

なし

### 12.3.3.8 ENC_SetCompareValue

カウンタ比較値の設定

**関数のプロトタイプ宣言:**
void
ENC_SetCompareValue(TSB_EN_TypeDef * ***ENx***,uint32_t ***ENC_Mode***,
uint32_t ***CompareValue***)

**引数:**
***ENx:*** ENC チャネルを選択します。

***ENC_Mode:*** 以下から、エンコーダ動作モードを選択します。
- **ENC_ENCODER _MODE**: エンコーダモード
- **ENC_SENSOR_EVENT_MODE**: センサモード(イベントカウント)
- **ENC_SENSOR_TIME_MODE**: センサモード(タイマカウント)
- **ENC_TIMER_MODE**: タイマモード

***CompareValue***: 以下から、カウンタ比較値を設定します。
エンコーダモードとセンサモード(イベントカウント)の場合: **0x0000 - 0xFFFF**
センサモード(タイマカウント)とタイマモードの場合: **0x000000 - 0xFFFFFF**

**機能:**
カウンタ比較値を設定します。

**戻り値:**
なし

### 12.3.3.9 ENC_GetCompareValue

カウンタ比較値の取得

**関数のプロトタイプ宣言:**
uint32_t
ENC_GetCompareValue(TSB_EN_TypeDef * **ENx**)

**引数:**
***ENx:*** ENC チャネルを選択します。

**機能:**
カウンタ比較値を取得します。

**戻り値:**
カウンタ比較値

### 12.3.3.10 ENC_GetCounterValue

エンコードカウンタ/キャプチャ値の取得

**関数のプロトタイプ宣言:**

uint32_t
ENC_GetCounterValue(TSB_EN_TypeDef * ***ENx***)

**引数:**
***ENx:*** ENC チャネルを選択します。

**機能:**
エンコードカウンタ/キャプチャ値を取得します。

**戻り値:**
エンコードカウンタ/キャプチャ値の取得

## 12.3.4 データ構造

### 12.3.4.1 ENC_InitTypeDef

**メンバ:**
uint32_t
***ModeType*** **:** エンコーダ入力モード:
- **ENC_ENCODER_MODE**: エンコーダモード
- **ENC_SENSOR_EVENT_MODE**: センサモード(イベントカウンタ)
- **ENC_SENSOR_TIME_MODE**: センサモード(タイマカウント)
- **ENC_TIMER_MODE**: タイマモード

uint32_t
***PhaseType***: 2 相/3 相入力選択:
- **ENC_TWO_PHASE**: 2 相入力
- **ENC_THREE_PHASE**: 3 相入力

uint32_t
***EdgeType***: ENCZ の使用エッジ選択:
- **ENC_RISING_EDGE**: 立ち上がりエッジ
- **ENC_FALLING_EDGE**: 立下りエッジ

uint32_t
***CompareStatus***: コンペアイネーブル:
- **ENC_COMPARE_DISABLE**: コンペア実行しない
- **ENC_COMPARE_ENABLE**: コンペア実行する

uint32_t
***ZphaseStatus***: Z 相イネーブル:
- **ENC_ZPHASE_DISABLE**: 禁止
- **ENC_ZPHASE_ENABLE**: 許可

uint32_t
***FilterValue***: ノイズフィルタ:
- **ENC_NO_FILTER**: ノイズフィルタなし
- **ENC_FILTER_VALUE31**: 31/fsys 未満のパルスはノイズとして除去
- **ENC_FILTER_VALUE63**: 63/fsys 未満のパルスはノイズとして除去
- **ENC_FILTER_VALUE127**: 127/fsys 未満のパルスはノイズとして除去

uint32_t
***IntEn*** ENC 割り込みの許可/禁止
- **ENC_INTERRUPT_DISABLE**: 禁止
- **ENC_INTERRUPT_ENABLE**: 許可

uint32_t
***PulseDivFactor***: エンコーダパルス分周比:
- **ENC_PULSE_DIV1**: 1 分周
- **ENC_PULSE_DIV2**: 2 分周
- **ENC_PULSE_DIV4**: 4 分周
- **ENC_PULSE_DIV8**: 8 分周
- **ENC_PULSE_DIV16**: 16 分周
- **ENC_PULSE_DIV32**: 32 分周
- **ENC_PULSE_DIV64**: 64 分周
- **ENC_PULSE_DIV128**: 128 分周

### 12.3.4.2 ENC_FlagStatus

**メンバ:**

uint32_t
***All*** **:** 全ての ENC フラグの状態

uint32_t
***ZPhaseDetectFlag***(bit12): Z 相通過検出

uint32_t
***RotationDirection*** (bit13): 回転方向

uint32_t
***ReverseErrorFlag*** (bit14): 反転エラーフラグ(センサモード(タイマカウント)時)

uint32_t
***CompareFlag*** (bit15): コンペア発生フラグ

# 13. PMD

## 13.1 概要

本デバイスはモーター制御回路(PMD)を内蔵しています。本製品のPMDは1シャントセンサレスモータ制御を実現する為に通電出力制御や、DC 過電圧検出入力を追加し、ADC を連携させたモータ制御を可能としています。

PMD（プログラマブルモータドライバ）回路は波形生成回路と同期トリガ生成回路の2ブロックから成り、波形生成回路はパルス幅変調回路、通電制御回路、保護制御回路、デッドタイム制御回路で構成されています。
- パルス幅変調回路はPWM 周波数が等しい3 相の独立したPWM 波形を生成します。
- 通電制御回路はU、V、W 相の各上下相の出力パターンを決定します。
- 保護制御回路では異常検出入力による緊急出力停止を行ないます。
- デッドタイム制御回路では上下相の切り替え時の短絡を防止します。
- 同期トリガ生成回路ではADC への同期トリガ信号を生成します。

全ドライバ API は、アプリで使用する API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver\src\tmpm37x_pmd.c
/Libraries/TX03_Periph_Driver\inc\tmpm37x_pmd.h

補足: "**x**"は 0, 2, 3, 4 を指します。

## 13.2 TMPM370/2/3/4 の違い

TMPM370: PMD を 2 チャネル内蔵しています。(PMD0, PMD1)
TMPM372/3/4: PMD を 1 チャネル内蔵しています。(PMD1)

## 13.3 API 関数

### 13.3.1 関数一覧

◆ void PMD_Enable(TSB_PMD_TypeDef * ***PMDx***);
◆ void PMD_Disable(TSB_PMD_TypeDef * ***PMDx***);
◆ void PMD_SetPortControl(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PortMode***);
◆ void PMD_Init(TSB_PMD_TypeDef * ***PMDx***,
PMD_InitTypeDef * ***InitStruct***);
◆ void PMD_ChangePWMCycle(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***CycleTiming***);
◆ uint32_t PMD_GetCntFlag(TSB_PMD_TypeDef * ***PMDx***);
◆ uint16_t PMD_GetCntValue(TSB_PMD_TypeDef * ***PMDx***);
◆ void PMD_SetCompareValue(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDPhase***,
uint32_t ***Timing***);
◆ void PMD_SetPortOutputMode(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Mode***);

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

◆ void PMD_SetOVVInputSrc(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Source***);
◆ void PMD_SetOVVAutoRelease(TSB_PMD_TypeDef * ***PMDx***,
FunctionalState ***NewState***);
◆ uint32_t PMD_GetOVVAbnormalLevel(TSB_PMD_TypeDef * ***PMDx***);
◆ uint32_t PMD_GetOVVCondition(TSB_PMD_TypeDef * ***PMDx***);

## 13.3.2 関数の種類

関数は、主に以下の 7 種類に分かれています。

1) PMD の共通設定:
PMD_Enable(), PMD_Disable(), PMD_SetPortControl(), PMD_Init(),
PMD_ChangePWMCycle(), PMD_SetCompareValue(),
PMD_SetAllPhaseCompareValue(), PMD_ChangeDutyMode(),
PMD_SetSelectMode()
2) PMD ポート出力の設定:
PMD_SetPortOutputMode(), PMD_SetOutputPhasePolarity(),
PMD_SetReflectTime(), PMD_SetPortOutput()
3) EMG 保護制御回路の設定:
PMD_EnableEMG(), PMD_DisableEMG(), PMD_SetEMGNoiseElimination(),
PMD_SetToolBreakOutput(), PMD_SetEMGMode(), PMD_EMGRelease(),
PMD_SetEMGInputSrc()
4) 動作状態の取得:
PMD_GetCntFlag(), PMD_GetCntValue(), PMD_GetEMGAbnormalLevel(),
PMD_GetEMGCondition(), PMD_GetOVVAbnormalLevel(), PMD_GetOVVCondition()
5) デッドタイム制御:
PMD_SetDeadTime()
6) ADC トリガ要求:
PMD_SetTrgCmpValue(), PMD_SetTrgMode(), PMD_SetTrgUpdate(),
PMD_SetEMGTrg(), PMD_SetTrgOutput()
7) OVV 保護制御回路の設定:
PMD_EnableOVV(), PMD_DisableOVV(), PMD_SetOVVNoiseElimination(),
PMD_SetADCMonitorInput(), PMD_SetOVVMode(), PMD_SetOVVAutoRelease(),
PMD_SetOVVInputSrc()

## 13.3.3 関数仕様

**補足**: 下記の全 API において、パラメータ “TSB_PMD_TypeDef * ***PMDx***” は 以下のいずれかを選択してください。
TMPM370: ***PMD0, PMD1***
TMPM372/3/4: ***PMD1***

### 13.3.3.1 PMD_Enable

PMD 機能の許可

**関数のプロトタイプ宣言:**
void
PMD_Enable(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
PMD 機能を許可します。

**戻り値:**
なし

#### 13.3.3.2 PMD_Disable

PMD 機能の禁止

**関数のプロトタイプ宣言:**
void
PMD_Disable(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
PMD 機能を禁止します。

**戻り値:**
なし

#### 13.3.3.3 PMD_SetPortControl

ポート制御の設定

**関数のプロトタイプ宣言:**
void
PMD_SetPortControl(TSB_PMD_TypeDef * ***PMDx***
uint32_t ***PortMode***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***PortMode:*** ポート制御の種類を選択します。
- **PMD_PORT_MODE_0:** 上相 High-z / 下相 High-z
- **PMD_PORT_MODE_1:** 上相 High-z / 下相 PMD 出力
- **PMD_PORT_MODE_2:** 上相 PMD 出力 / 下相 High-z
- **PMD_PORT_MODE_3:** 上相 PMD 出力 / 下相 PMD 出力

**機能:**
ポート制御を設定します。

**戻り値:**
なし

#### 13.3.3.4 PMD_Init

PMD の初期化

**関数のプロトタイプ宣言:**
void

PMD_Init(TSB_PMD_TypeDef * ***PMDx***,
PMD_InitTypeDef * ***InitStruct***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***InitStruct:*** PMD の基本設定内容を格納した構造体を指定します。
(詳細は “データ構造” 参照)

**機能:**
PMD を初期化します。

**戻り値:**
なし

### 13.3.3.5 PMD_ChangePWMCycle

PWM 周期の設定

**関数のプロトタイプ宣言:**
void
PMD_ChangePWMCycle(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***CycleTiming***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***CycleTiming:*** PWM 周期を 0x0000 ~ 0xFFFF の間で設定します。

**機能:**
PWM 周期を設定します。

**戻り値:**
なし

**補足**:
設定値は 0x10 以上の値を設定してください。0x10 未満の値を設定した場合、0x10 が設定されたものとして動作します。(設定値を取得すると設定した値が読み出せます)

### 13.3.3.6 PMD_GetCntFlag

PWM カウンタフラグの取得

**関数のプロトタイプ宣言:**
uint32_t
PMD_GetCntFlag(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
PWM カウンタフラグを取得します。

**戻り値:**
PWM カウンタフラグ:
**PMD_COUNTER_UP:** アップカウント中
**PMD_COUNTER_DOWN :** ダウンカウント中

### 13.3.3.7 PMD_GetCntValue

PWM 周期カウント値の取得

**関数のプロトタイプ宣言:**
uint16_t
PMD_GetCntValue(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
PWM 周期カウント値を取得します。

**戻り値:**
PWM 周期カウント値

### 13.3.3.8 PMD_SetCompareValue

PWM パルス幅の設定

**関数のプロトタイプ宣言:**
void
PMD_SetCompareValue(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDPhase***,
uint32_t ***Timing***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***PMDPhase:*** 3 相のいぞれか、または 3 相すべてを選択します。
- **PMD_PHASE_U:** U 相
- **PMD_PHASE_V:** V 相
- **PMD_PHASE_W:** W 相
- **PMD_PHASE_ALL:** 3 相すべて

***Timing:*** コンペア値を 0x0000 ~ 0xFFFF の間で設定します。

**機能:**
PWM パルス幅を設定します。

**戻り値:**
なし

### 13.3.3.9 PMD_SetPortOutputMode

U,V,W 相のポート出力設定

**関数のプロトタイプ宣言:**
void
PMD_SetPortOutputMode(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Mode***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Mode:*** U,V,W 相のポート出力を設定します。
- **PMD_PORT_OUTPUT_MODE_0:** PMDxMDCR<SYNTMD>=0
- **PMD_PORT_OUTPUT_MODE_1:** PMDxMDCR<SYNTMD>=1

**機能:**
U,V,W 相のポート出力設定を行います。

**補足**:
PMDxMDCR<SYNTMD>, PMDxMDPOT<POLH><POLL>, PMDxMDOUT <UPWN><VPWN><WPWN> <UOC> <VOC> <WOC>の内容により出力ポートの制御を行います。(x=0, 1)

PMD_SetPortOutputMode()により PMDxMDCR<SYNTMD>を設定します。
PMD_SetOutputPhasePolarity()により PMDxMDPOT<POLH><POLL>を設定します
PMD_SetPortOutput()により PMDxMDOUT<UPWN><VPWN> <WPWN> <UOC> <VOC> <WOC>を設定します。

上記による設定によって得られる端子出力の関係については下表を参照してください。

MTPDxMDCR<SYNTMD>=0

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

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

MTPDxMDCR<SYNTMD>=0

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

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

MTPDxMDCR<SYNTMD>=1

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

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

MTPDxMDCR<SYNTMD>=1

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

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

**戻り値:**
なし

### 13.3.3.10 PMD_SetOutputPhasePolarity

上相/下相の出力ポート極性の選択

**関数のプロトタイプ宣言:**
void
PMD_SetOutputPhasePolarity(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***OutputPhase***,
uint32_t ***Polarity***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***OutputPhase:*** 出力ポートの上相/下相を選択します。
- **PMD_OUTPUT_PHASE_UPPER:** 上相の出力ポート
- **PMD_OUTPUT_PHASE_LOWER:** 下相の出力ポート

***Polarity:*** 極性を選択します。
- **PMD_POLARITY_LOW:** ロー・アクティブ
- **PMD_POLARITY_HIGH:** ハイ・アクティブ

**機能:**
上相/下相の出力ポートの極性を選択します。

**補足**:
1 詳細は PMD_SetPortOutputMode() 関数を参照してください。
2 PWM を無効の状態で選択を行ってください。

**戻り値:**
なし

### 13.3.3.11 PMD_SetReflectTime

U, V, W 相出力設定のポート出力反映時のタイミング選択

**関数のプロトタイプ宣言:**
void
PMD_SetReflectTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***ReflectedTime***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***ReflectedTime:*** U, V, W 相出力設定のポート出力反映時のタイミングを選択します。
- **PMD_REFLECTED_TIME_WRITE:** 書き込み時に反映
- **PMD_REFLECTED_TIME_MIN:** PWM カウンタ MDCNT= "1"(最小)の時、反映
- **PMD_REFLECTED_TIME_MAX:** PWM カウンタ MDCNT= PMDxMDPRD<MDPRD>(最大)の時、反映
- **PMD_REFLECTED_TIME_MIN_MAX:** PWM カウンタ MDCNT="1"(最小)および PMDxMDPRD<MDPRD>(最大)の時、反映

**機能:**
U, V, W 相出力設定のポート出力反映時のタイミングを選択します。

**補足**:
PWM を無効の状態で選択を行ってください。

**戻り値:**
なし

### 13.3.3.12 PMD_EnableEMG

EMG 保護回路の許可

**関数のプロトタイプ宣言:**
void
PMD_EnableEMG(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
EMG 保護回路を許可します。

**戻り値:**
なし

### 13.3.3.13 PMD_DisableEMG

EMG 保護回路の禁止

**関数のプロトタイプ宣言:**
void
PMD_DisableEMG(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
EMG 保護回路を禁止します。

**戻り値:**
なし

### 13.3.3.14 PMD_SetEMGNoiseElimination

異常検出入力のノイズ除去時間の設定

**関数のプロトタイプ宣言:**
void
PMD_SetEMGNoiseElimination(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***NoiseElimination***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***NoiseElimination:***異常検出入力のノイズ除去時間を選択します。

- **PMD_NOISE_ELIMINATION_NONE:** ノイズフィルタを経由しません。
- **PMD_NOISE_ELIMINATION_16:** 入力ノイズ除去時間 16/fsys[s]
- **PMD_NOISE_ELIMINATION_32:** 入力ノイズ除去時間 32/fsys[s]
- **PMD_NOISE_ELIMINATION_48:** 入力ノイズ除去時間 48/fsys[s]
- **PMD_NOISE_ELIMINATION_64:** 入力ノイズ除去時間 64/fsys[s]
- **PMD_NOISE_ELIMINATION_80:** 入力ノイズ除去時間 80/fsys[s]
- **PMD_NOISE_ELIMINATION_96:** 入力ノイズ除去時間 96/fsys[s]
- **PMD_NOISE_ELIMINATION_112:** 入力ノイズ除去時間 112/fsys[s]
- **PMD_NOISE_ELIMINATION_128:** 入力ノイズ除去時間 128/fsys[s]
- **PMD_NOISE_ELIMINATION_144:** 入力ノイズ除去時間 144/fsys[s]
- **PMD_NOISE_ELIMINATION_160:** 入力ノイズ除去時間 160/fsys[s]
- **PMD_NOISE_ELIMINATION_176:** 入力ノイズ除去時間 176/fsys[s]
- **PMD_NOISE_ELIMINATION_192:** 入力ノイズ除去時間 192/fsys[s]
- **PMD_NOISE_ELIMINATION_208:** 入力ノイズ除去時間 208/fsys[s]
- **PMD_NOISE_ELIMINATION_224:** 入力ノイズ除去時間 224/fsys[s]
- **PMD_NOISE_ELIMINATION_240:** 入力ノイズ除去時間 240/fsys[s]

**機能:**
異常検出入力のノイズ除去時間を設定します。

**戻り値:**
なし

### 13.3.3.15 PMD_SetToolBreakOutput

ツールブレーク時の PWM 出力状態の選択

**関数のプロトタイプ宣言:**
void
PMD_SetToolBreakOutput(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Status***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Status:*** ツールブレーク時の PWM 出力状態を選択します。
- **PMD_BREAK_STATUS_PMD:** PMD 出力継続
- **PMD_BREAK_STATUS_HIGH_IMPEDANCE:** ハイ・インピーダンス

**機能:**
ツールブレーク時の PWM 出力状態を選択します。

**戻り値:**
なし

### 13.3.3.16 PMD_SetEMGMode

EMG 保護モードの選択

**関数のプロトタイプ宣言:**
void
PMD_SetEMGMode(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Mode***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Mode:*** EMG 保護モードを選択します。
- **PMD_EMG_MODE_0:** 全相オン/PORT ハイ・インピーダンス
- **PMD_EMG_MODE_1:** 全相オフ/PORT ハイ・インピーダンス
- **PMD_EMG_MODE_2:** 全相オン/PORT 出力許可
- **PMD_EMG_MODE_3:** 全相オフ/PORT ハイ・インピーダンス

**機能:**
EMG 保護モードを選択します。

**戻り値:**
なし

### 13.3.3.17 PMD_EMGRelease

EMG 保護状態からの復帰

**関数のプロトタイプ宣言:**
void
PMD_EMGRelease(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
EMG 保護状態から復帰します。

**補足**:
本関数をコールすると、PMDxMDOUT<UPWN><VPWN><WPWN>、
PMDxMDOUT<UOC> <VOC> <WOC>に 0 を設定します。(x=0, 1)

**戻り値:**
なし

### 13.3.3.18 PMD_GetEMGAbnormalLevel

異常状態入力のレベルモニタ

**関数のプロトタイプ宣言:**
uint32_t
PMD_GetEMGAbnormalLevel (TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
異常状態入力のレベルをモニタします。

**戻り値:**
異常状態入力の状態
**PMD_ABNORMAL_LEVEL_L:** 異常状態入力のレベルが"L"
**PMD_ABNORMAL_LEVEL_H :** 異常状態入力のレベルが"H"

### 13.3.3.19 PMD_GetEMGCondition

EMG 保護の状態モニタ

**関数のプロトタイプ宣言:**
uint32_t
PMD_GetEMGCondition (TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**

EMG 保護の状態をモニタします。

**戻り値:**
EMG 保護の状態
**0**：通常動作中
**1**：EMG 保護中

### 13.3.3.20 PMD_SetDeadTime

デッドタイムの設定

**関数のプロトタイプ宣言:**
void
PMD_SetDeadTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Time***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Time:*** デッドタイムを 0x00 ～ 0xFF の間で設定します。

**機能:**
デッドタイムを設定します。

**補足**:
PWM を無効の状態で選択を行ってください。

**戻り値:**
なし

### 13.3.3.21 PMD_SetAllPhaseCompareValue

PWM パルス幅の設定

**関数のプロトタイプ宣言:**
void
PMD_SetAllPhaseCompareValue(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***UPhaseTiming***,
uint32_t ***VPhaseTiming***,
uint32_t ***WPhaseTiming***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***UPhaseTiming:*** U 相に出力するパルス幅を 0x0000～0xFFFF の間で設定します。

***VPhaseTiming:*** V 相に出力するパルス幅を 0x0000～0xFFFF の間で設定します。

***WPhaseTiming:*** W 相に出力するパルス幅を 0x0000～0xFFFF の間で設定します。

**機能:**
PWM パルス幅の設定をします。

**戻り値:**
なし

### 13.3.3.22 PMD_ChangeDutyMode

DUTY モードの設定

**関数のプロトタイプ宣言:**
void
PMD_ChangeDutyMode(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***DutyMode***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***DutyMode:*** DUTY モードを選択します。
- **PMD_DUTY_MODE_U_PHASE:** U 相共通
- **PMD_DUTY_MODE_3_PHASE:** 3 相独立

**機能:**
DUTY モードを設定します。

**戻り値:**
なし

### 13.3.3.23 PMD_SetPortOutput

UVW 相出力の設定

**関数のプロトタイプ宣言:**
Result
PMD_SetPortOutput(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDPhase***,
uint8_t ***Output***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***PMDPhase:*** UVW 相を選択します。
- **PMD_PHASE_U:** U 相
- **PMD_PHASE_V:** V 相
- **PMD_PHASE_W:** W 相
- **PMD_PHASE_ALL:** 全相

***Output:*** 出力を選択します。
- **PMD_OUTPUT_L_L:** 上相出力"L"、下相出力"L"
- **PMD_OUTPUT_L_H:** 上相出力"L",下相出力"H"
- **PMD_OUTPUT_H_L:** 上相出力"H", 下相出力"L"
- **PMD_OUTPUT_H_H:** 上相出力"H", 下相出力"H"
- **PMD_OUTPUT_PWM_IPWM:** 上相出力"PWM", 下相出力 IPWM

- **PMD_OUTPUT_IPWM_PWM:** 上相出力"IPWM", 下相出力 PWM
- **PMD_OUTPUT_H_PWM:** 上相出力"H", 下相出力"PWM"
- **PMD_OUTPUT_L_PWM:** 上相出力"L", 下相出力"PWM"
- **PMD_OUTPUT_PWM_L:** 上相出力"PWM", 下相出力"L"
- **PMD_OUTPUT_H_IPWM:** 上相出力"H", 下相出力"IPWM"
- **PMD_OUTPUT_L_IPWM:** 上相出力"L", 下相出力"IPWM"
- **PMD_OUTPUT_IPWM_H:** 上相出力"IPWM", 下相出力 H"

**機能:**
UVW 相出力を設定します。

**戻り値:**
実行結果:
**SUCCESS:** PMD 出力設定成功
**ERROR:** PMD 出力設定失敗

**補足**:
1. IPWM は PWM の反転です。
2. 詳細は PMD_SetPortOutputMode()関数を参照してください。

### 13.3.3.24 PMD_SetTrgCmpValue

トリガコンペアレジスタ値の設定

**関数のプロトタイプ宣言:**
void
PMD_SetTrgCmpValue(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***TRGCMP0Timing***,
uint32_t ***TRGCMP1Timing***,
uint32_t ***TRGCMP2Timing***,
uint32_t ***TRGCMP3Timing***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***TRGCMP0Timing:*** トリガコンペアレジスタ 0 の値を 0x0001～[<MDPRO>-1]の間で設定してください。

***TRGCMP1Timing:*** トリガコンペアレジスタ 1 の値を 0x0001～[<MDPRO>-1]の間で設定してください。

***TRGCMP2Timing:*** トリガコンペアレジスタ 2 の値を 0x0001～[<MDPRO>-1]の間で設定してください。

***TRGCMP3Timing:*** トリガコンペアレジスタ 3 の値を 0x0001～[<MDPRO>-1]の間で設定してください。

**機能:**
トリガコンペアレジスタ値を設定します。

**戻り値:**

なし

**補足**: PMDnTRGCMPx (x=0, 1)は 1～ [<MDPRD> – 1]の間で設定してください。

### 13.3.3.25 PMD_SetTrgMode

トリガモードの設定

**関数のプロトタイプ宣言:**
void
PMD_SetTrgMode (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDTrg***,
uint32_t ***Mode***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***PMDTrg:*** PMD トリガを選択します。
- **PMD_ADC_TRG_0:** トリガ 0
- **PMD_ADC_TRG_1:** トリガ 1
- **PMD_ADC_TRG_2:** トリガ 2
- **PMD_ADC_TRG_3:** トリガ 3

***Mode:*** PMD トリガを選択します。
- **PMD_TRG_MODE_0:** トリガ出力禁止
- **PMD_TRG_MODE_1:** ダウンカウント時の一致でトリガ出力
- **PMD_TRG_MODE_2:** アップカウント時の一致でトリガ出力
- **PMD_TRG_MODE_3:** アップ/ダウンカウント時の一致でトリガ出力
- **PMD_TRG_MODE_4:** PWM キャリアピークでトリガ出力
- **PMD_TRG_MODE_5:** PWM キャリアボトムでトリガ出力
- **PMD_TRG_MODE_6:** PWM キャリアピーク/ボトムでトリガ出力
- **PMD_TRG_MODE_7:** トリガ出力禁止

**機能:**
トリガモードを設定します。

**戻り値:**
なし

### 13.3.3.26 PMD_SetTrgUpdate

トリガコンペアレジスタの更新タイミング

**関数のプロトタイプ宣言:**
void
PMD_SetTrgUpdate (TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***PMDTrg***,
uint32_t ***UpdateTiming***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***PMDTrg:*** PMD トリガを選択します。
- **PMD_ADC_TRG_0:** トリガ 0
- **PMD_ADC_TRG_1:** トリガ 1
- **PMD_ADC_TRG_2:** トリガ 2
- **PMD_ADC_TRG_3:** トリガ 3

***Mode:*** PMDTRG0 ～ PMDTRG1 を更新タイミングを選択します。
- **PMD_TRG_UPDATE_SYNC:** PWM 同期更新
- **PMD_TRG_UPDATE_ASYNC:** 非同期更新(バッファの非同期更新を許可します。書き込み後、直ちに反映)

**機能:**
トリガコンペアレジスタの更新タイミングを設定します。

**戻り値:**
なし

### 13.3.3.27 PMD_SetEMGTrg

EMG 保護動作中の出力許可設定

**関数のプロトタイプ宣言:**
```
void
PMD_SetEMGTrg (TSB_PMD_TypeDef * PMDx,
                FunctionalState NewState)
```

**引数:**
***PMDx:*** PMD チャネルを指定します。

***NewState:*** EMG 保護動作中の出力許可/禁止を選択します。
- **ENABLE:** 保護動作時トリガ出力許可
- **DIABLE:** 保護動作時トリガ出力禁止

**機能:**
EMG 保護動作中の出力許可を設定します。

**戻り値:**
なし

### 13.3.3.28 PMD_SetTrgOutput

トリガ出力の設定

**関数のプロトタイプ宣言:**
```
void
PMD_SetTrgOutput(TSB_PMD_TypeDef * PMDx,
                 uint32_t TrgMode,
                 uint32_t TrgChannel);
```

**引数:**

***PMDx:*** PMD チャネルを指定します。

***TrgMode:*** トリガ出力モードを選択します。
- **PMD_TRG_FIXED_OUTPUT:** トリガ固定出力
- **PMD_TRG_VARIABLE_OUTPUT:** トリガ選択出力

***TrgChannel:*** トリガ出力ポートを選択します。
***TrgMode*** == **PMD_TRG_FIXED_OUTPUT の場合**:
- **PMD_TRG_OUTPUT_0:** PMDTRG0 より出力
- **PMD_TRG_OUTPUT_1:** PMDTRG1 より出力
- **PMD_TRG_OUTPUT_2:** PMDTRG2 より出力
- **PMD_TRG_OUTPUT_3:** PMDTRG3 より出力

***TrgMode*** == **PMD_TRG_VARIABLE_OUTPUT の場合**:
- **PMD_TRG_OUTPUT_0:** PMDTRG0 より出力
- **PMD_TRG_OUTPUT_1:** PMDTRG1 より出力
- **PMD_TRG_OUTPUT_2:** PMDTRG2 より出力
- **PMD_TRG_OUTPUT_3:** PMDTRG3 より出力
- **PMD_TRG_OUTPUT_4:** PMDTRG4 より出力
- **PMD_TRG_OUTPUT_5:** PMDTRG5 より出力

**機能:**
トリガ出力を設定します。

**戻り値:**
なし

### 13.3.3.29 PMD_SetSelectMode

モードの選択

**関数のプロトタイプ宣言:**
```
void
PMD_SetSelectMode(TSB_PMD_TypeDef * PMDx,
                  uint32_t Mode);
```

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Mode:*** モードを選択します。
- **PMD_BUS_MODE:** ダブルバッファ後段へ入力するデータをバスから選択したレジスタ値を使用します。(バスモード)
- **PMD_VE_MODE:** ダブルバッファ後段へ入力するデータをベクトルエンジン(VE)からの値を使用します。(VE モード)

**機能:**
モードを選択します。

**戻り値:**
なし

### 13.3.3.30 PMD_SetEMGInputSrc

EMG 入力選択

**関数のプロトタイプ宣言:**
void
PMD_SetEMGInputSrc(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Source***);

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Source:*** EMG 入力を選択します。
- **PMD_EMG_PORT_INPUT:** ポート入力
- **PMD_EMG_COMPARATOR_OUTPUT:** COMP 入力

**機能:**
EMG 入力を選択します。

**補足:**
本関数は M370 のみサポートします。

**戻り値:**
なし

### 13.3.3.31 PMD_EnableOVV

OVV 保護回路の許可

**関数のプロトタイプ宣言:**
void
PMD_EnableOVV(TSB_PMD_TypeDef * ***PMDx***);

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
OVV 保護回路を許可します。

**戻り値:**
なし

### 13.3.3.32 PMD_DisableOVV

OVV 保護回路の禁止

**関数のプロトタイプ宣言:**
void
PMD_DisableOVV(TSB_PMD_TypeDef * ***PMDx***);

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
OVV 保護回路を禁止します。

**戻り値:**
なし

### 13.3.3.33 PMD_SetOVVNoiseElimination

OVV 入力検出時間の設定

**関数のプロトタイプ宣言:**
void
PMD_SetOVVNoiseElimination(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***NoiseElimination***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

***NoiseElimination:*** OVV 入力検出時間を選択します。
- **PMD_NOISE_ELIMINATION_16:** OVV 入力検出時間 X16/fsys[s]
- **PMD_NOISE_ELIMINATION_32:** OVV 入力検出時間 X32/fsys[s]
- **PMD_NOISE_ELIMINATION_48:** OVV 入力検出時間 X48/fsys[s]
- **PMD_NOISE_ELIMINATION_64:** OVV 入力検出時間 X64/fsys[s]
- **PMD_NOISE_ELIMINATION_80:** OVV 入力検出時間 X80/fsys[s]
- **PMD_NOISE_ELIMINATION_96:** OVV 入力検出時間 X96/fsys[s]
- **PMD_NOISE_ELIMINATION_112:** OVV 入力検出時間 X112/fsys[s]
- **PMD_NOISE_ELIMINATION_128:** OVV 入力検出時間 X128/fsys[s]
- **PMD_NOISE_ELIMINATION_144:** OVV 入力検出時間 X144/fsys[s]
- **PMD_NOISE_ELIMINATION_160:** OVV 入力検出時間 X160/fsys[s]
- **PMD_NOISE_ELIMINATION_176:** OVV 入力検出時間 X176/fsys[s]
- **PMD_NOISE_ELIMINATION_192:** OVV 入力検出時間 X192/fsys[s]
- **PMD_NOISE_ELIMINATION_208:** OVV 入力検出時間 X208/fsys[s]
- **PMD_NOISE_ELIMINATION_224:** OVV 入力検出時間 X224/fsys[s]
- **PMD_NOISE_ELIMINATION_240:** OVV 入力検出時間 X240/fsys[s]

**機能:**
OVV 入力検出時間を設定します。

**戻り値:**
なし

### 13.3.3.34 PMD_SetADCMonitorInput

ADC 監視割り込み入力の許可/禁止

**関数のプロトタイプ宣言:**
void
PMD_SetADCMonitorInput(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***Monitor***,
FunctionalState ***NewState***);

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Monitor:*** OVV 保護用 ADC 監視割り込み入力を選択します。
- **PMD_ADC_MONITOR_A:** ADC A 監視割り込み
- **PMD_ADC_MONITOR_B:** ADC B 監視割り込み

***NewState:*** ADC 監視割り込み入力の許可/禁止を選択します。
- **ENABLE:** 入力許可
- **DIABLE:** 入力禁止

**機能:**
OVV 保護用 ADC 監視割り込み入力の許可/禁止を選択します。

**戻り値:**
なし

### 13.3.3.35 PMD_SetOVVMode

OVV 保護モードの選択

**関数のプロトタイプ宣言:**
```
void
PMD_SetOVVMode(TSB_PMD_TypeDef * PMDx,
                               uint32_t Mode);
```

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Mode:*** OVV 保護モードを選択します。
- **PMD_OVV_MODE_0:** 出力制御なし。
- **PMD_OVV_MODE_1:** 全上相オン、全下相オフ。
- **PMD_OVV_MODE_2:** 全上相オフ、全下相オン。
- **PMD_OVV_MODE_3:** 全相オフ (オン = High, OFF = Low [ハイアクティブ (PLL/H=1)時)

**機能:**
OVV 保護モードを選択します。

**戻り値:**
なし

### 13.3.3.36 PMD_SetOVVInputSrc

OVV 入力選択

**関数のプロトタイプ宣言:**
```
void
PMD_SetOVVInputSrc(TSB_PMD_TypeDef * PMDx,
                                  uint32_t Source);
```

**引数:**
***PMDx:*** PMD チャネルを指定します。

***Source:*** OVV 入力を選択します。
- **PMD_OVV_PORT_INPUT:** ポート入力
- **PMD_OVV_ADC_MONITOR:** ADC 監視信号

**機能:**
OVV 入力を選択します。

**戻り値:**
なし

### 13.3.3.37 PMD_SetOVVAutoRelease

OVV 保護状態からの自動復帰設定

**関数のプロトタイプ宣言:**
void
PMD_SetOVVAutoRelease(TSB_PMD_TypeDef * ***PMDx***,
FunctionalState ***NewState***);

**引数:**
***PMDx:*** PMD チャネルを指定します。

***NewState:*** OVV 保護状態からの自動復帰の許可/禁止を選択します。
- **ENABLE:** 保護状態からの自動復帰許可
- **DIABLE:** 保護状態からの自動復帰禁止

**機能:**
OVV 保護状態からの自動復帰設定を行います。

**戻り値:**
なし

### 13.3.3.38 PMD_GetOVVAbnormalLevel

OVV 入力状態の取得

**関数のプロトタイプ宣言:**
uint32_t
PMD_GetOVVAbnormalLevel(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
OVV 入力状態を取得します。

**戻り値:**

OVV 入力状態
**PMD_ABNORMAL_LEVEL_L:** OVV 保護入力が"L"
**PMD_ABNORMAL_LEVEL_H :** OVV 保護入力が"H"

#### 13.3.3.39 PMD_GetOVVCondition

OVV 保護状態の取得

**関数のプロトタイプ宣言:**
uint32_t
PMD_GetOVVCondition(TSB_PMD_TypeDef * ***PMDx***)

**引数:**
***PMDx:*** PMD チャネルを指定します。

**機能:**
OVV 保護状態を取得します。

**戻り値:**
OVV 保護状態:
**PMD_OVV_NORMAL:** 通常動作中
**PMD_OVV_PROTECTED:** 保護中

### 13.3.4 データ構造

#### 13.3.4.1 PMD_InitTypeDef

**メンバ:**
uint32_t
***CycleMode:*** PWM 周期延長モードを指定します。
- **PMD_PWM_NORMAL_CYCLE:** 通常周期
- **PMD_PWM_4_FOLD_CYCLE:** 4 倍周期

uint32_t
***DutyMode:*** DUTY モードを指定します。
- **PMD_DUTY_MODE_U_PHASE:** U 相共通
- **PMD_DUTY_MODE_3_PHASE:** 3 相独立

uint32_t
***IntTiming:*** PWM モード 1(三角波)の時の PWM 割り込みタイミングを選択します。
- **PMD_PWM_INT_TIMING_MINIMUM:** PWM カウンタ MDCNT="1"の時(最小)割り込み要求
- **PMD_PWM_INT_TIMING_MAXIMUM:** PWM カウンタ MDCNT= MTPDxMDPRD<MDPRD>の時 (最大)割り込み要求

uint32_t
***IntCycle:*** PWM 割り込み周期を選択します。
- **PMD_PWM_INT_CYCLE_HALF:** PWM 0.5 周期毎に割り込み (PWM モード 1(三角波)のみ設定可能です)
- **PMD_PWM_INT_CYCLE_1:** PWM 1 周期毎に割り込み
- **PMD_PWM_INT_CYCLE_2:** PWM 2 周期毎に割り込み

- **PMD_PWM_INT_CYCLE_4:** PWM 4 周期毎に割り込み

uint32_t
***CarrierMode:*** PWM キャリア波形を指定します。
- **PMD_CARRIER_WAVE_MODE_0:** PWM モード 0 (エッジ PWM, ノコギリ波)
- **PMD_CARRIER_WAVE_MODE_1:** PWM モード 1 (センターPWM, 三角波)

uint32_t
***CycleTiming:*** PWM 周期を 0x0000～0xFFFF の間で指定します。

**補足**:
設定値が 0x10 以下の 場合は 0x10 が設定されたものとして動作します。

# 14. TRMOSC

## 14.1 概要

TMPM372, M373, M374 には、内蔵高速発振の周波数を調整する機能があります。(注)

内蔵高速発振調整機能は、16 ビットタイマ/ イベントカウンタ(TMRB) のパルス幅測定機能を使用して周波数の調整を行います。

TRMOSC ドライバ API は、TRMOSC の設定機能を持ち、トリミング制御、トリミング値の設定、トリミング値の取得などの機能を提供します。

本ドライバは、以下のファイルで構成されています。
\Libraries\TX03_Periph_Driver\src\tmpm37x_trmosc.c
\Libraries/TX03_Periph_Driver\inc\tmpm37x_ trmosc.h

**補足**: この調整機能は、OFD 用基準クロックには適用されません。

## 14.2 API 関数

### 14.2.1 関数リスト

- void SetTrmOsc(TRMOSC_ControlTypeDef ***ControlStruct***);
- uint32_t GetTrimmingControlValue(void);
- uint32_t GetInitCoarseTrimValue(void);
- uint32_t GetInitFineTrimValue(void);
- uint32_t GetCoarseTrimValueSet(void);
- uint32_t GetFineTrimValueSet(void);

### 14.2.2 関数の種類

上記関数は 2 のグループに分けられます。

1) TRMOSC の設定:
   SetTrmOsc()
2) TRMOSC 動作状態と結果の取得:
   GetTrimmingControlValue(), GetInitCoarseTrimValue(), GetInitFineTrimValue(), GetCoarseTrimValueSet(), GetFineTrimValueSet()

### 14.2.3 関数仕様

#### 14.2.3.1 SetTrmOsc

TRMOSC レジスタの設定

**関数のプロトタイプ宣言:**
void
SetTrmOsc(TRMOSC_ControlTypeDef ***ControlStruct***)

**引数:**

***ControlStruct*:** TRMOSC の制御内容を設定します。

**機能:**
TRMOSC レジスタの設定を行います。

**戻り値:**
なし

### 14.2.3.2 GetTrimmingControlValue

トリミング制御状態の取得

**関数のプロトタイプ宣言:**
uint32_t
GetTrimmingControlValue(void)

**引数:**
なし

**機能:**
トリミング制御状態を取得します。

**戻り値:**
1: 許可
0: 禁止

### 14.2.3.3 GetInitCoarseTrimValue

初期 粗トリミング値の取得

**関数のプロトタイプ宣言:**
uint32_t
GetInitCoarseTrimValue(void)

**引数:**
なし

**機能:**
初期 粗トリミング値を取得します。

**戻り値:**
初期 粗トリミング値

### 14.2.3.4 GetInitFineTrimValue

初期 微トリミング値の取得

**関数のプロトタイプ宣言:**
uint32_t
GetInitFineTrimValue(void)

**引数:**
なし

**機能:**
初期 微トリミング値を取得します。

**戻り値:**
初期 微トリミング値

#### 14.2.3.5 GetCoarseTrimValueSet

設定した粗トリミング値の取得

**関数のプロトタイプ宣言:**
uint32_t
GetFineTrimValueSet(void)

**引数:**
なし

**機能:**
設定した粗トリミング値を取得します。

**戻り値:**
設定した粗トリミング値

#### 14.2.3.6 GetFineTrimValueSet

設定した微トリミング値の取得

**関数のプロトタイプ宣言:**
uint32_t
GetFineTrimValueSet(void)

**引数:**
なし

**機能:**
設定した微トリミング値を取得します。

**戻り値:**
設定した微トリミング値

### 14.2.4 データ構造

#### 14.2.4.1 TRMOSC_ControlTypeDef

**メンバ:**
uint32_t
***TrimmingControl***: 以下のいずれかのトリミング制御を選択します。
- **1:** 許可

- **0:** 禁止

uint32_t
***CoarseTrimmingValue***: 初期粗トリミング値

uint32_t
***FineTrimmingValue***: 初期微トリミング値