TOSHIBA

TX03 ペリフェラルドライバ
ユーザーガイド
(TMPM37AFSQG)

第一版
2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M37AUG-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

目次.....2
1. はじめに.....1
2. TX03 ペリフェラルドライバの構成.....1
3. ADC.....2
3.1 概要.....2
3.2 API 関数.....2
3.2.1 関数一覧.....2
3.2.2 関数の種類.....3
3.2.3 関数仕様.....3
3.2.4 データ構造:.....12
4. CG.....15
4.1 概要.....15
4.2 API 関数.....15
4.2.1 関数一覧.....15
4.2.2 関数の種類.....16
4.2.3 関数仕様.....16
4.2.4 データ構造:.....29
5. FC.....30
5.1 概要.....30
5.2 API 関数.....30
5.2.1 関数一覧.....30
5.2.2 関数の種類.....30
5.2.3 関数仕様.....31
5.2.4 データ構造:.....35
6. GPIO.....36
6.1 概要.....36
6.2 API 関数.....36
6.2.1 関数一覧.....36
6.2.2 関数の種類.....36
6.2.3 関数仕様.....37
6.2.4 データ構造:.....47
7. SBI.....48
7.1 概要.....48
7.2 API 関数.....48
7.2.1 関数一覧.....48
7.2.2 関数の種類.....48
7.2.3 関数仕様.....49
7.2.4 データ構造:.....54
8. OFD.....56
8.1 概要.....56
8.2 API 関数.....56
8.2.1 関数一覧.....56
8.2.2 関数の種類.....56
8.2.3 関数仕様.....56
8.2.4 データ構造:.....58
9. TMRB.....59
9.1 概要.....59

9.2 API 関数 59
9.2.1 関数一覧 59
9.2.2 関数の種類 60
9.2.3 関数仕様 60
9.2.4 データ構造: 69
10. SIO/UART 71
10.1 概要 71
10.2 API 関数 71
10.2.1 関数一覧 71
10.2.2 関数の種類 72
10.2.3 関数仕様 72
10.2.4 データ構造: 87
11. VLTD 90
11.1 概要 90
11.2 API 関数 90
11.2.1 関数一覧 90
11.2.2 関数の種類 90
11.2.3 関数仕様 90
11.2.4 データ構造: 91
12. WDT 92
12.1 概要 92
12.2 API 関数 92
12.2.1 関数一覧 92
12.2.2 関数の種類 92
12.2.3 関数仕様 92
12.2.4 データ構造: 95
13. PMD 96
13.1 概要 96
13.2 API 関数 96
13.2.1 関数一覧 96
13.2.2 関数の種類 98
13.2.3 関数仕様 98
13.2.4 データ構造 120

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。TMPM37AFSQG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TMPM37AFSQGシリーズMCU用です。

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

TMPM37AFSQG ペリフェラルドライバは以下の仕様に基づいています。
- スタートアップルーチンといくつかの関数を除き、C 言語で記述されています。
- すべての周辺機能を網羅しています。

2. TX03 ペリフェラルドライバの構成

/Libraries
TX03 CMSIS ファイルと TMPM37AFSQG ペリフェラルドライバが格納されています。

/Libraries/ TX03_CMSIS
このフォルダには TMPM37AFSQG CMSIS ファイルのデバイス・ペリフェラル・アクセス・レイヤーが格納されています。

/Libraries/TX03_Periph_Driver
TMPM37AFSQG 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
TMPM37AFSQG 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
TMPM37AFSQG 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
TMPM37AFSQG 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
TMPM37AFSQG 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
TMPM37AFSQG ペリフェラルドライバの使用例が格納されています。

/Utilities/TMPM37A-EVAL
TMPM37AFSQG ボードのハードウェアリソース用の設定ファイル、およびドライバファイル (例: led, key) が格納されています。

3. ADC

3.1 概要

本デバイスは、12 ビット逐次変換方式アナログ/デジタルコンバータ(AD コンバータ)を内蔵しています。

AD コンバータユニット B は 5 本のアナログ入力を持っています。4 本はモータ 0 の測定用に使用可能です。この内 1 本は IC 内蔵でオペアンプの出力に接続されていますので、外部から入力可能なアナログ入力端子は 4 本です。

4本の外部アナログ入力端子(AINB9～AINB12)は、入出力専用ポートと兼用です。

機能と特徴:
(1) PMD やタイマ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) ソフトウエア起動、常時起動において任意のアナログ入力を変換する事ができます。
(3) AD 変換値レジスタが12 個あります。
(4) TMRB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、常時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。

ADC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、モニタ機能設定、割り込み設定、ステータスリード、AD 変換結果の取得などの機能を提供します。

全ドライバ API は、アプリで使用する API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver\src\tmpM37A_adc.c
/Libraries/TX03_Periph_Driver\inc\tmpM37A_adc.h

3.2 API 関数

3.2.1 関数一覧

- 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 ***Re sultREGx***);
- 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

関数は、主に以下の 3 種類に分かれています。

1) AD 変換設定:
   ADC_SetClk(), ADC_SetMonitor(), ADC_DisableMonitor(),
   ADC_SelectPMDTrgProgNum(), ADC_SetPMDTrgProgINT(), ADC_SetPMDTrg(),
   ADC_SetTimerTrg(), ADC_SetSWTrg(), ADC_SetConstantTrg()
2) AD 変換の許可/禁止と開始/終了:
   ADC_Enable(), ADC_Disable(), ADC_Start(), ADC_StopConstantTrg()
3) AD 変換ステータス/結果の読み出し:
   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(SCLK)の設定

関数のプロトタイプ宣言:

```
void
ADC_SetClk(TSB_AD_TypeDef * ADx,
           uint32_t Sample_HoldTime,
           ADC_PRESCALER Prescaler_Output)
```

引数:

ADx: AD 変換のユニットを指定します。

- **TSB_ADB:** AD コンバータユニット 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> に“1001b” をライトします。

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

戻り値:

なし

3.2.3.2 ADC_Enable

AD 変換の許可

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

機能:
AD 変換を許可します。

戻り値:
なし

3.2.3.3 ADC_Disable

AD 変換の禁止

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

機能:
AD 変換を禁止します。

戻り値:
なし

3.2.3.4 ADC_Start

AD 変換の開始

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

Trg**:** 以下からトリガタイプを選択します。
- **ADC_TRG_SW:** ソフトウェア変換
- **ADC_TRG_CONSTANT:** 常時 AD 変換

機能:
選択したトリガタイプで 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 の AD 変換停止

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

機能:
通常起動時の AD 変換を停止します。

戻り値:
なし

3.2.3.6 ADC_GetConvertState

AD 変換状態の確認

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

Trg**:** 以下から起動方法を選択します。
- **ADC_TRG_SW:** ソフトウェア起動
- **ADC_TRG_CONSTANT:** 常時 AD 変換
- **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 変換状態を確認します。

戻り値:
AD 変換状態:
BUSY: 変換中
DONE: 停止

3.2.3.7 ADC_SetMonitor

AD 監視機能の許可

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット 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 AD 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 監視機能の禁止

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

引数:
ADx: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット 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 変換結果の読み出し

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: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット 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): オーバーランフラグ
- **ADResult**(Bit4 ～ Bit15): AD 変換結果

3.2.3.10 ADC_SelectPMDTrgProgNum

AD 変換ユニットの PMD が発生するトリガ信号 PMD6 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: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

***SELx*:** PMD トリガ用プログラム選択番号を以下から選択します。

- **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*:**どの PMDx (x = 6 ～ 11)トリガによってプログラムを起動するか選択します。

- **TRG_ENABLE(PMD_PROGy)**: 有効にする PMD トリガ用プログラム y **(y = 0 ～5)**
- **TRG_DISABLE(PMD_PROGy)**: 無効にする PMD トリガ用プログラム y **(y = 0 ～5)**

機能:
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*** でレジスタの有効/無効を選択します。

戻り値:
なし

3.2.3.11 ADC_SetPMDTrgProgINT

PMD トリガ用割り込みプログラムの設定。

関数のプロトタイプ宣言:
```
void
ADC_SetPMDTrgProgINT(TSB_AD_TypeDef * ADx,
                     PMD_TrgProgINTTypeDef * TrgProgINT)
```

引数:
ADx: AD 変換のユニットを指定します。

- **TSB_ADB:** AD コンバータユニット B

***TrgProgINT*:** PMD トリガ用割り込みプログラムの構造体です。
```
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
```
詳細は後述の “データ構造:”を参照してください。

機能:
TrgProgINT により PMD 用トリガ割り込みプログラムを設定します。

戻り値:
なし

3.2.3.12 ADC_SetPMDTrg

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: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット 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
```

詳細は後述の “データ構造:”を参照してください。

機能:
PMDTrg により 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

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: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

***ResultREGx*:** 以下から、タイマトリガ用プログラム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

MacroAINx: 以下から、許可/禁止付 AD チャネルを選択します。
- **TRG_ENABLE(y)**: ***ResultREGx*** に対する AD チャネル'y'を許可
- **TRG_DISABLE(y)**: ***ResultREGx*** に対する AD チャネル'y'を禁止

以下から、'y'を選択します。
ADC_AIN9 ～ ADC_AIN12, ADC_AIN16

機能:
ResultREGx により AD 変換結果レジスタの設定を行い、タイマトリガ用プログラムレジスタの ***MacroAINx*** により AIN 端子に対するレジスタの許可/禁止を設定します。

戻り値:
なし

3.2.3.14 ADC_SetSWTrg

ソフトウェアトリガ用レジスタの設定

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: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

***ResultREGx*:**ソフトウェアトリガ用プログラムによる 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

MacroAINx: 以下から、許可/禁止付 AD チャネルを選択します。
- **TRG_ENABLE(y)**: ***ResultREGx*** に対する AD チャネル'y'を許可
- **TRG_DISABLE(y)**: ***ResultREGx*** に対する AD チャネル'y'を禁止

以下から、'y'を選択します。

ADC_AIN9 to ADC_AIN12, ADC_AIN16

機能:
ResultREGx により AD 変換結果レジスタの設定を行い、ソフトウェアトリガ用プログラムレジスタの ***MacroAINx*** により AIN 端子に対するレジスタの許可/禁止を設定します。

戻り値:
なし

3.2.3.15 ADC_SetConstantTrg

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: AD 変換のユニットを指定します。
- **TSB_ADB:** AD コンバータユニット B

***ResultREGx*:** 以下から、常時トリガプログラム用 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

MacroAINx: 以下から、許可/禁止付 AD チャネルを選択します。
- **TRG_ENABLE(y)**: ***ResultREGx*** に対する AD チャネル'y'を許可
- **TRG_DISABLE(y)**: ***ResultREGx*** に対する AD チャネル'y'を禁止

以下から、'y'を選択します。
ADC_AIN9 to ADC_AIN12, ADC_AIN16

機能:
ResultREGx により 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*** により AIN 端子に対するレジスタの許可/禁止を設定します。

戻り値:
なし

3.2.4 データ構造:

3.2.4.1 ADC_MonitorTypeDef

メンバ:

ADC_CMPCRx
CMPCRx 以下からコンペア制御レジスタを選択します。
- **ADC_CMPCR_0:** ADxCMPCR0
- **ADC_CMPCR_1:** ADxCMPCR1

ADC_REGx
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

uint32_t
CmpTimes 以下から比較カウント数を選択します。
- **1 ～ 16**

ADC_CmpCondition
Condition 以下から ADxREGm と ADxCMPn の比較設定を選択します。(m = 0 ～ 11、x = B、n = 0～1)
- **ADC_LARGER_THAN_CMP_REG:** 変換結果レジスタ値が比較レジスタ 0 より大きい場合に割り込みを発生します。
- **ADC_SMALLER_THAN_CMP_REG:** 変換結果レジスタ値が比較レジスタ 0 より小さい場合に割り込みを発生します。

uint32_t
CmpValue 以下から ADxCMP0、または ADxCMP1 に設定する比較値を選択します。(x = B)
- **0 ～ 4095**

3.2.4.2 PMD_TrgProgINTTypeDef

メンバ:

PMD_INT_NAME
INTProg0 以下からプログラム 0 に対する割り込みを選択します。
- **PMD_INTNONE:** 割り込み出力なし
- **PMD_INTADPDB:** INTADPDB 出力

PMD_INT_NAME
INTProg1 プログラム 1 に対する割り込みを選択する以外 ***INTProg0*** と同じです。

PMD_INT_NAME
INTProg2 プログラム 2 に対する割り込みを選択する以外 ***INTProg0*** と同じです。

PMD_INT_NAME
INTProg3 プログラム 3 に対する割り込みを選択する以外 ***INTProg0*** と同じです。

PMD_INT_NAME
INTProg4 プログラム 4 に対する割り込みを選択する以外 ***INTProg0*** と同じです。

PMD_INT_NAME
INTProg5 プログラム 5 に対する割り込みを選択する以外 ***INTProg0*** と同じです。

3.2.4.3 PMD_TrgTypeDef

メンバ:

PMD_PROGx
ProgNum 以下から、ADxPSETn (x = B、n = 0 ～ 5) に対するプログラム番号を選択します。

- **PMD_PROG0:** プログラム番号 0
- **PMD_PROG1:** プログラム番号 1
- **PMD_PROG2:** プログラム番号 2
- **PMD_PROG3:** プログラム番号 3
- **PMD_PROG4:** プログラム番号 4
- **PMD_PROG5:** プログラム番号 5

VE_PHASE
Reg0_Phase 以下から、ADxPSETn (x = B、n = 0 ～ 5)に対する REG0 の相を選択します。
- **VE_PHASE_NONE:** 指定なし
- **VE_PHASE_U:** U 相
- **VE_PHASE_V:** V 相
- **VE_PHASE_W:** W 相

VE_PHASE
Reg1_Phase ADxPSETn (x = B、n = 0 ～ 5)に対する REG1 の相を選択します。選択する相は ***Reg0_Phase*** と同じです。

VE_PHASE
Reg2_Phase ADxPSETn (x = B、n = 0 ～ 5)に対する REG2 の相を選択します。選択する相は ***Reg0_Phase*** と同じです。

VE_PHASE
Reg3_Phase ADxPSETn (x = B、n = 0 ～ 5)に対する REG3 の相を選択します。選択する相は ***Reg0_Phase*** と同じです。

uint8_t
Reg0_AINx 以下から、ADxPSETn の REG0 に対する許可/禁止付 AD チャネルを選択します。
- **TRG_ENABLE(y)**: AD チャネル‘y’を許可

➢ **TRG_DISABLE(y)**: AD チャネル'y'を禁止
以下から、'y'を選択します。
ADC_AIN9 ～ ADC_AIN12, ADC_AIN16

uint8_t
Reg1_AINx ADxPSETn の REG1 に対する許可/禁止付 AD チャネルを選択します。
選択する AD チャネルは ***Reg0_AINx*** と同じです。

uint8_t
Reg2_AINx ADxPSETn の REG2 に対する許可/禁止付 AD チャネルを選択します。
選択する AD チャネルは ***Reg0_AINx*** と同じです。

uint8_t
Reg3_AINx ADxPSETn の REG3 に対する許可/禁止付 AD チャネルを選択します。
選択する AD チャネルは ***Reg0_AINx*** と同じです。

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 CG API は TMPM37AFSQG CG において以下の機能を提供します。

- システムクロックの制御、クロック逓倍回路（PLL）の制御
- プリスケーラクロックの制御
- ウォーミングアップタイマの制御
- 各種低消費電力モードの制御

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\tmpM37A_cg.c
/Libraries/TX03_Periph_Driver\inc\tmpM37A_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。

fosc1 : X1, X2 端子より入力されるクロック
fosc2 : 内蔵発振器より入力されるクロック
fosc : fosc1 または fosc2 どちらか選択されたシステムクロック
fPLL : PLL により逓倍(4 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<GEAR[2:0]> で選択されたクロック
fsys : fgear と同一クロック(システムクロック)
fperiph : CGSYSCR<FPSEL> で選択されたクロック
φT0 : CGSYSCR<PRCK[2:0]> 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(CG_WarmUpSrc ***Source***, uint16_t ***Time***);
- void CG_StartWarmUp(void);
- WorkState CG_GetWarmUpState(void);
- Result CG_SetFPLLValue(uint32_t NewValue);
- uint32_t CG_GetFPLLValue(void);
- Result CG_SetPLL(FunctionalState ***NewState***);
- FunctionalState CG_GetPLLState(void);
- Result CG_SetFosc(CG_FoscSrc ***Source***, FunctionalState ***NewState***);
- void CG_SetFoscSrc(CG_FoscSrc ***Source***);

- CG_FoscSrc CG_GetFoscSrc(void);
- FunctionalState CG_GetFoscState(CG_FoscSrc ***Source***);
- void CG_SetPortM(CG_PortMMode ***Mode***);
- void CG_SetSTBYMode(CG_STBYMode ***Mode***);
- CG_STBYMode CG_GetSTBYMode(void);
- void CG_SetPinStateInStopMode(FunctionalState ***NewState***);
- FunctionalState CG_GetPinStateInStopMode(void);
- Result CG_SetFcSrc(CG_FcSrc ***Source***);
- CG_FcSrc CG_GetFcSrc(void);
- void CG_SetSTBYReleaseINTSrc(CG_INTSrc ***INTSource***,
  CG_INTActiveState ***ActiveState***,
  FunctionalState ***NewState***);
- CG_INTActiveState CG_GetSTBYReleaseINTState(CG_INTSrc ***INTSource***);
- void CG_ClearINTReq(CG_INTSrc ***INTSource***);
- CG_NMIFactor CG_GetNMIFlag(void);
- CG_ResetFlag CG_GetResetFlag(void);

4.2.2 関数の種類

CG API は 3 つのグループに分けられます。

1) クロックの種類:
   CG_SetFgearLevel(), CG_GetFgearLevel(), CG_SetPhiT0Src(), CG_GetPhiT0Src(), CG_SetPhiT0Level(), CG_GetPhiT0Level(), CG_SetWarmUpTime(), CG_StartWarmUp(), CG_GetWarmUpState(), CG_SetFPLLValue(), CG_GetFPLLValue(), CG_SetPLL(), CG_GetPLLState(), CG_SetFosc(), CG_GetFoscState(), CG_SetFcSrc(), CG_GetFcSrc(), CG_SetFoscSrc(), CG_GetFoscSrc(), CG_SetPortM()
2) スタンバイモードの設定:
   CG_SetSTBYMode(), CG_GetSTBYMode(), CG_SetPinStateInStopMode(), CG_GetPinStateInStopMode()
3) 割り込みの設定など、その他:
   CG_SetSTBYReleaseINTSrc(), CG_GetSTBYReleaseINTState(),CG_ClearINTReq(), CG_GetNMIFlag(), CG_GetResetFlag()

4.2.3 関数仕様

4.2.3.1 CG_SetFgearLevel

fgear,fc 間の分周レベル設定

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

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

- **CG_DIVIDE_1**: fgear = fc
- **CG_DIVIDE_2**: fgear = fc/2
- **CG_DIVIDE_4**: fgear = fc/4
- **CG_DIVIDE_8**: fgear = fc/8
- **CG_DIVIDE_16**: fgear = fc/16

機能: :
fgear,fc 間の分周レベルを設定します。

戻り値:
なし

4.2.3.2 CG_GetFgearLevel

fgear,fc 間の分周レベルの取得

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

引数:
なし

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

戻り値:
fgear, fc 間の分周レベルで、下記のいずれかの値になります。
CG_DIVIDE_1: fgear = fc
CG_DIVIDE_2: fgear = fc/2
CG_DIVIDE_4: fgear = fc/4
CG_DIVIDE_8: fgear = fc/8
CG_DIVIDE_16: fgear = fc/16
CG_DIVIDE_UNKNOWN: 無効データ

4.2.3.3 CG_SetPhiT0Src

PhiT0(ΦT0) ,fc 間の PhiT0(ΦT0) ソースの設定

関数のプロトタイプ宣言:
void
CG_SetPhiT0Src(CG_PhiT0Src ***PhiT0Src***)

引数:
PhiT0Src: 以下から PhiT0 ソースを選択します。
- **CG_PHIT0_SRC_FGEAR** : fgear が PhiT0 ソース
- **CG_PHIT0_SRC_FC**: fc が PhiT0 ソース

機能:
PhiT0 (ΦT0) ソースを選択します。

戻り値:
なし

4.2.3.4 CG_GetPhiT0Src

PhiT0 (ΦT0) ソースの取得

関数のプロトタイプ宣言:
CG_PhiT0Src
CG_GetPhiT0Src (void)

引数:
なし

機能:
PhiT0 (ΦT0) ソースを取得します。

戻り値:
CG_PHIT0_SRC_FGEAR：fgear が PhiT0 ソース
CG_PHIT0_SRC_FC: fc が PhiT0 ソース

4.2.3.5 CG_SetPhiT0Level

PhiT0 (ΦT0) と fc 間の分周レベルの設定

関数のプロトタイプ宣言:
Result
CG_SetPhiT0Level (CG_DivideLevel ***DividePhiT0FromFc***)

引数:
DividePhiT0FromFc: PhiT0 (ΦT0) と fc 間の分周レベルを下記の値から設定します。
- **CG_DIVIDE_1**: ΦT0 = fc
- **CG_DIVIDE_2**: ΦT0 = fc/2
- **CG_DIVIDE_4**: ΦT0 = fc/4
- **CG_DIVIDE_8**: ΦT0 = fc/8
- **CG_DIVIDE_16**: ΦT0 = fc/16
- **CG_DIVIDE_32**: ΦT0 = fc/32
- **CG_DIVIDE_64**: ΦT0 = fc/64
- **CG_DIVIDE_128**: ΦT0 = fc/128
- **CG_DIVIDE_256**: ΦT0 = fc/256
- **CG_DIVIDE_512**: ΦT0 = fc/512

機能:
プリスケ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 間の分周レベルの取得

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

引数:

なし

機能:
PhiT0(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 : 無効データ

4.2.3.7 CG_SetWarmUpTime

ウォームアップ時間の設定

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

引数:
Source: 以下から、ウォームアップカウンタを選択します。
- **CG_WARM_UP_SRC_OSC1**: fosc1
- **CG_WARM_UP_SRC_OSC2**: fosc2

Time: 0U～ 0x1000U から選択します。

機能:
ウォームアップ時間の設定を行います。設定時間の算出方法は以下です。
設定値 = ((ウォームアップ時間) / (入力クロック))/16

以下はウォームアップ時間の設定例です。
/* ウォームアップ時間が 100us で、入力クロックが 8M の場合 */
設定値= 100*10E(-6)/(1/(8*10E(6)))/16=0x0320>>4=0x32

戻り値:
なし

4.2.3.8 CG_StartWarmUp

ウォームアップの開始

関数のプロトタイプ宣言:

void
CG_StartWarmUp (void)

引数:
なし

機能:
ウォームアップを開始します。

戻り値:
なし

4.2.3.9 CG_GetWarmUpState

ウォームアップ動作状態の確認

関数のプロトタイプ宣言:
WorkState
CG_GetWarmUpState (void)

引数:
なし

機能:
ウォーミングアップ動作状態を確認します。

ウォーミングアップタイマの使用例:

```
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC1, 0x32);
/* start warm up */
CG_StartWarmUp();
/* check warm up is finished or not*/
While( CG_GetWarmUpState() == BUSY);
```

戻り値:
ウォームアップ動作状態:
DONE: 動作終了
BUSY: 動作中

4.2.3.10 CG_SetFPLLValue

PLL の逓倍数を設定

関数のプロトタイプ宣言:
Result
CG_SetFPLLValue(uint32_t ***NewValue***)

引数:
NewValue: 以下から逓倍数を選択します。

- **CG_FPLL_8M_MULTIPLY_5:** 入力クロック 8MHz、出力クロック 40MHz(5 逓倍)
- **CG_FPLL_10M_MULTIPLY_4:** 入力クロック 10MHz、出力クロック 40MHz(4 逓倍)

機能:
PLL の逓倍数を設定します。

戻り値:
SUCCESS: 成功
ERROR: 失敗

4.2.3.11 CG_GetFPLLValue

PLL の逓倍数を取得

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

引数:
なし。

機能:
PLL の逓倍数を取得します。

戻り値:
PLL の逓倍数
- **CG_FPLL_8M_MULTIPLY_5:** 入力クロック 8MHz、出力クロック 40MHz(5 逓倍)
- **CG_FPLL_10M_MULTIPLY_4:** 入力クロック 10MHz、出力クロック 40MHz(4 逓倍)

4.2.3.12 CG_SetPLL

PLL 動作の許可/禁止

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

引数:
NewState: 以下から選択します。
- **ENABLE**: 許可
- **DISABLE**: 禁止

機能:
PLL 動作の許可/禁止を選択します。
PLL として fc を選択した場合、無効にすることができません。この場合、戻り値は **ERROR** となります。

戻り値:
SUCCESS: 成功
ERROR: 失敗

4.2.3.13 CG_GetPLLState

PLL 設定状態の確認

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

引数:
なし

機能:
PLL 設定状態を確認します。

戻り値:
PLL 設定状態です
ENABLE: PLL 有効
DISABLE: PLL 無効

4.2.3.14 CG_SetFosc

高速発振器(osc1 or osc2)の有効/無効

関数のプロトタイプ宣言:
Result
CG_SetFosc(CG_FoscSrc ***Source***,
FunctionalState ***NewState***)

引数:
Source: 以下から fosc のソースクロックを選択します。
- **CG_FOSC_OSC1**: fosc1
- **CG_FOSC_OSC2**: fosc2

NewState: 以下から、fosc の有効/無効を選択します。
- **ENABLE**: 有効
- **DISABLE**: 無効

機能:
高速発振器の有効/無効を選択します。
fgear と system clock (fsys)が選択されている場合、高速発振器 (fosc) は無効にできません。この場合、戻り値は **ERROR** となります。

戻り値:
SUCCESS: 成功
ERROR: 失敗

4.2.3.15 CG_SetFoscSrc

高速発振器(fosc)のソース設定

関数のプロトタイプ宣言:
void

CG_SetFoscSrc(CG_FoscSrc ***Source***)

引数:
Source: fosc のソースを選択します。
- **CG_FOSC_OSC1**: fosc1
- **CG_FOSC_OSC2**: fosc2

機能:
高速発振器(fosc)のソースを設定します。

戻り値:
なし

4.2.3.16 CG_GetFoscSrc

高速発振器(fosc)ソースの取得

関数のプロトタイプ宣言:
CG_FoscSrc
CG_GetFoscSrc(void)

引数:
なし

機能:
高速発振器ソースを取得します。

戻り値:
fosc のソース
CG_FOSC_OSC1: fosc1
CG_FOSC_OSC2: fosc2

4.2.3.17 CG_GetFoscState

高速発振器(fosc)の状態

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

引数:
Source: 以下から fosc のソースを選択します。
- **CG_FOSC_OSC1**: fosc1
- **CG_FOSC_OSC2**: fosc2

機能:
高速発振器の状態を取得します。

戻り値:
fosc の状態
ENABLE: 有効
DISABLE: 無効

4.2.3.18 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。

戻り値:
なし

4.2.3.19 CG_SetSTBYMode

スタンバイモードの設定

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: 内部発振器含めてすべての内部回路が停止します。
- **CG_STBY_MODE_IDLE**: IDLE モード: CPU のみ停止します

機能:
スタンバイ命令実行後の低消費電力モードを選択します。

戻り値:
なし

4.2.3.20 CG_GetSTBYMode

スタンバイモード設定状態の取得

関数のプロトタイプ宣言:
CG_STBYMode
CG_GetSTBYMode (void)

引数:
なし

機能:

スタンバイモードの設定状態を取得します。
設定状態が “Reserved”の場合、戻り値は “**CG_STBY_MODE_UNKNOWN**” です。

戻り値:
低消費電力モード
CG_STBY_MODE_STOP: STOP モード
CG_STBY_MODE_IDLE: IDLE モード
CG_STBY_MODE_UNKNOWN: 無効データ

4.2.3.21 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:** STOP モード中端子をドライブしません。(<DRVE>=0)
- **ENABLE:** STOP モード中端子をドライブします(<DRVE>=1)

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

機能:
STOP モード中の端子状態を設定します。

戻り値:
なし

4.2.3.22 CG_GetPinStateInStopMode

STOP モード中の端子状態設定の取得

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設定を取得します。

戻り値:
DISABLE: STOP モード中端子をドライブしません。(<DRVE>=0)
ENABLE: STOP モード中端子をドライブします。(<DRVE>=1)

4.2.3.23 CG_SetFcSrc

fc ソースクロック選択

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 ソースクロックを選択します。
- **CG_FC_SRC_FOSC** : fosc
- **CG_FC_SRC_FPLL**: fpll

機能:
fc ソースクロックを選択します。
本 API をコールする前に以下の状態になっていることを確認してください。
a) 高速発振器を選択している
b) a)かつ ***Source*** が **CG_FC_SRC_FPLL** の場合、PLL が有効
(“**CG_SetPLL**(ENABLE)”)になっている。
上記状態になっていない場合、戻り値は **ERROR** となります。

戻り値:
SUCCESS: 成功
ERROR: 無効

4.2.3.24 CG_GetFcSrc

fc ソースクロックの取得

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

引数:
なし

機能:
fc ソースクロック設定状態を取得します。

戻り値:
CG_FC_SRC_FOSC: fosc
CG_FC_SRC_FPLL: fpll

4.2.3.25 CG_SetSTBYReleaseINTSrc

スタンバイモードの解除割り込みソースの設定

関数のプロトタイプ宣言:
void
CG_SetSTBYReleaseINTSrc (CG_INTSrc ***INTSource***,
CG_INTActiveState ***ActiveState***,
FunctionalState ***NewState***)

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

- **CG_INT_SRC_6** : INT6
- **CG_INT_SRC_7** : INT7
- **CG_INT_SRC_C** : INTC

ActiveState: 以下から、解除トリガのアクティブ状態を選択します。
- **CG_INT_ACTIVE_STATE_L**: Low レベル
- **CG_INT_ACTIVE_STATE_H**: High レベル
- **CG_INT_ACTIVE_STATE_FALLING**: 立下りエッジ
- **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、解除トリガの許可/禁止を選択します。
- **ENABLE**: 許可
- **DISABLE:** 禁止

機能:
スタンバイモードの解除割り込みソースを設定します。

戻り値:
なし

4.2.3.26 CG_GetSTBYReleaseINTState

スタンバイモードの解除割り込みソースのアクティブ状態の取得

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

引数:
INTSource: 以下から、解除割り込みソースを選択します。
CG_INT_SRC_6, CG_INT_SRC_7, CG_INT_SRC_C

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:
- **CG_INT_ACTIVE_STATE_FALLING**: 立下りエッジ
- **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

4.2.3.27 CG_ClearINTReq

スタンバイ解除割り込み要求のクリア

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

引数:

INTSource: 以下から、解除割り込みソースを選択します。
CG_INT_SRC_6, CG_INT_SRC_7, CG_INT_SRC_C

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

4.2.3.28 CG_GetNMIFlag

NMI フラグの取得

関数のプロトタイプ宣言:
CG_NMIFactor
CG_GetNMIFlag (void)

引数:
なし

機能:
NMI フラグを取得します。

戻り値:
WDT (Bit 0) : WDT による NMI 発生

4.2.3.29 CG_GetResetFlag

リセットフラグの取得

関数のプロトタイプ宣言:
CG_ResetFlag
CG_GetResetFlag(void)

引数:
なし

機能:
リセットフラグを取得します。

戻り値:
PowerOn (Bit 0) : パワーオンリセット
ResetPin (Bit 1) : 端子リセット
WDTReset (Bit 2) :WDT リセット
VLTDReset (Bit 3) :VLTD リセット
DebugReset (Bit 4) :SYSRESETREQ によるリセット
OFDReset (Bit 5) : OFD 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 ソース起動状態です。

ビットフィールド:
uint32_t
WDT(Bit 0)WDT による NMI 発生

4.2.4.2 CG_ResetFlag

メンバ:
uint32_t
All リセット要因フラグです。

ビットフィールド:
uint32_t
PowerOn(Bit 0)　Power On Reset フラグ

uint32_t
ResetPin(Bit 1)　RESET 端子フラグ

uint32_t
WDTReset(Bit 2)　WDT リセットフラグ

uint32_t
VLTDReset (Bit 3) 低電圧検出フラグ

uint32_t
DebugReset(Bit 4)　デバッグリセットフラグ

uint32_t
OFDReset(Bit 5)　OFD リセットフラグ

5. FC

5.1 概要

本デバイスは、フラッシュメモリを内蔵しています。
フラッシュメモリのサイズは 64Kbyte です。

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

ブロック構成は、デバイスのデータシートを参照してください。

全ドライバ API は、アプリで使用する API 定義、マクロ、データタイプ、構造を格納する以下のファイルで構成されています。
\Libraries\TX03_Periph_Driver\src\tmpM37A_fc.c
\Libraries\TX03_Periph_Driver \inc\tmpM37A_fc.h

5.2 API 関数

5.2.1 関数一覧

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

5.2.2 関数の種類

関数は、主に以下の 5 種類に分かれています。

1) セキュリティ設定(Flash ROM データの読み出し、デバッグ):
   FC_SetSecurityBit(), FC_GetSecurityBit()
2) 自動動作状態およびプロテクト状態の取得:
   FC_GetBusyState(), FC_GetBlockProtectState().
3) プロテクトの設定:
   FC_ProgramBlockProtectState(), FC_EraseBlockProtectState().
4) 自動実行コマンド(書き込み、チップ消去、ブロック消去):
   FC_WritePage(), FC_EraseBlock(), FC_EraseChip().
5) フラッシュバッファの制御:
   FC_ SetBufferState()

5.2.3 関数仕様

5.2.3.1 FC_SetBufferState

フラッシュバッファの無効/有効

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

引数:
NewState: 以下からフラッシュバッファの無効/有効を選択します。
- **DISABLE**: 無効(同時にバッファをクリアします)
- **ENABLE**: 有効

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

5.2.3.2 FC_SetSecurityBit

セキュリティビットの設定

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

引数:
NewState: セキュリティビットを設定します。
- **DISABLE**: セキュリティ機能設定不可
- **ENABLE**: セキュリティビット設定可能

機能:
1) 書き込み/消去プロテクト用のすべてのプロテクトビット (PSRA<BLn>, n=0,1)を”1” にします。
2) FCSECBIT<SECBIT>を”1”にします。

上記の 2 つの条件が成立すると、セキュリティ機能が有効になります。セキュリティ機能が有効な状態の制限内容は次の通りです。
- ROM 領域のデータの読み出し。
- JTAG/SW、トレースの通信

したがって、この API を使用する場合は、注意して実行してください。

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

戻り値:
なし

5.2.3.3 FC_GetSecurityBit

セキュリティビットの設定状態の取得

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

引数:
なし

機能:
セキュリティビットの設定状態を取得します。

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

5.2.3.4 FC_GetBusyState

自動動作状態の取得

関数のプロトタイプ宣言:
WorkState
FC_GetBusyState(void)

引数:
なし。

機能:
自動動作状態を取得します。

戻り値:
BUSY: 自動動作中
DONE: 自動動作終了

5.2.3.5 FC_GetBlockProtectState

ブロックのプロテクト状態の取得。

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

引数:
BlockNum: ブロック番号を選択します。
- **FC_BLOCK_0** block 0
- **FC_BLOCK_1** block 1

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

戻り値:
DISABLE: プロテクト状態ではない。
ENABLE: プロテクト状態

5.2.3.6 FC_ProgramBlockProtectState

ブロックのプロテクト設定。

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

引数:
BlockNum: ブロック番号を選択します。
- **FC_BLOCK_0** block 0
- **FC_BLOCK_1** block 1

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

戻り値:
FC_SUCCESS: プロテクト設定の成功
FC_ERROR_PROTECTED: プロテクト設定の失敗(すでにプロテクト済の場合は再度プロテクト設定を行いません)
FC_ERROR_OVER_TIME: プロテクト設定の失敗(自動動作のタイムアウト

5.2.3.7 FC_EraseBlockProtectState

プロテクトの解除

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

引数:
BlockGroup: ブロックグループを指定してください。
- **FC_BLOCK_GROUP_0** :ブロック 0, 1

機能:
プロテクトビットを"0"にすることでプロテクトを解除します。

戻り値:
FC_SUCCESS: プロテクト解除の成功
FC_ERROR_OVER_TIME: プロテクト解除の失敗(自動動作のタイムアウト)

5.2.3.8 FC_WritePage

ページ単位の書き込み

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

引数:
PageAddr: ページの開始アドレスを指定します。
Data: 書き込むデータバッファへのポインタを指定します。サイズは 128Byte です。

機能:
ページ書き込みを行います。

自動ページ書き込みは、既に消去された 1 ページにつき一回のみ実施されます。データ値が“1” または “0” のいずれかであっても、2 回以上書き込みを実施することはありません。
補足: あらかじめデータを消去せずに書き込みを行うと、デバイスに損傷を与える恐れがあります。

戻り値:
FC_SUCCESS: 書き込み成功
FC_ERROR_PROTECTED: 書き込み失敗(ブロックにプロテクトが設定されている)
FC_ERROR_OVER_TIME: 書き込みの失敗(自動動作のタイムアウト)

5.2.3.9 FC_EraseBlock

ブロック単位の消去

関数のプロトタイプ宣言:
FC_Result
FC_EraseBlock(uint32_t ***BlockAddr***)

引数:
BlockAddr: ブロック開始アドレスを指定します。

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

戻り値:
FC_SUCCESS: 消去成功
FC_ERROR_PROTECTED: 消去失敗(ブロックにプロテクトが設定されている)
FC_ERROR_OVER_TIME: 消去の失敗(自動動作のタイムアウト)

5.2.3.10 FC_EraseChip

チップ消去

関数のプロトタイプ宣言:
FC_Result
FC_EraseChip(void)

引数:
なし。

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

戻り値:

FC_SUCCESS: チップ消去成功。ただしブロックの一部にプロテクトが設定されている場合、そのブロックのデータは消去されません。
FC_ERROR_PROTECTED: 消去失敗(すべてのブロックにプロテクトが設定されている)
FC_ERROR_OVER_TIME: 消去の失敗(自動動作のタイムアウト)

5.2.4 データ構造:

なし

6. GPIO

6.1 概要

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

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

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/ tmpM37A _gpio.c
/Libraries/TX03_Periph_Driver/inc/tmpM37A _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

関数は、主に以下の 3 種類に分かれています。

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

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

3) その他:
GPIO_EnableFuncReg(), GPIO_DisableFuncReg()

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

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 DATA レジスタの読み込み

関数のプロトタイプ宣言:
uint8_t
GPIO_ReadDataBit(GPIO_Port ***GPIO_x***,
uint8_t ***Bit_x***)

引数:
GPIO_x: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3

- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7

機能:
ビット単位で DATA データレジスタを読み込みます。

戻り値:
DATA データレジスタのビット値:
- **GPIO_BIT_VALUE_0**: 0
- **GPIO_BIT_VALUE_1**: 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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Data: DATA レジスタに書き込む値を設定します。

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B

- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

BitValue: GPIO ビットに書き込む値
- **GPIO_BIT_VALUE_0**: 0
- **GPIO_BIT_VALUE_1**: 1

機能:
ビット単位で DATA データレジスタを書き込みます。

戻り値:
なし

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4

- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

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

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

戻り値:
なし

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

機能:
出力ポートに設定します。

戻り値:
なし

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

機能:
入力ポートに設定します。

戻り値:
なし

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K

- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

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

機能:
出力ポートの許可/禁止設定を行います。
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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7

- **GPIO_BIT_ALL:** すべての GPIO 端子

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

機能:
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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

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

機能:
ポートのプルアップ設定を行います。
NewState が **ENABLE** の時、プルアップ許可。
NewState が **DISABLE** の時、プルアップ禁止。

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

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

機能:
ポートのプルダウン設定を行います。
NewState が **ENABLE** の時、プルダウン許可。
NewState が **DISABLE** の時、プルダウン禁止。

戻り値:
なし

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G
- **GPIO_PJ :** GPIO port J
- **GPIO_PK:** GPIO port K
- **GPIO_PM:** GPIO port M

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7
- **GPIO_BIT_ALL:** すべての GPIO 端子

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

機能:
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、オープンドレイン許可。
NewState が **DISABLE** の時、CMOS 許可。

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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G

FuncReg_x: GPIO 機能レジスタの番号を選択します。
- **GPIO_FUNC_REG_1**: GPIO 機能レジスタ 1
- **GPIO_FUNC_REG_2**: GPIO 機能レジスタ 2
- **GPIO_FUNC_REG_3**: GPIO 機能レジスタ 3
- **GPIO_FUNC_REG_4**: GPIO 機能レジスタ 4
- **GPIO_FUNC_REG_5**: GPIO 機能レジスタ 5

Bit_x: GPIO 端子を選択します。有効ビットの組み合わせが可能です。
- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2
- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7

機能:
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: GPIO ポートを選択します。
- **GPIO_PB:** GPIO port B
- **GPIO_PE:** GPIO port E
- **GPIO_PF:** GPIO port F
- **GPIO_PG:** GPIO port G

FuncReg_x: GPIO 機能レジスタの番号を選択します。
- **GPIO_FUNC_REG_1**: GPIO 機能レジスタ 1
- **GPIO_FUNC_REG_2**: GPIO 機能レジスタ 2
- **GPIO_FUNC_REG_3**: GPIO 機能レジスタ 3
- **GPIO_FUNC_REG_4**: GPIO 機能レジスタ 4
- **GPIO_FUNC_REG_5**: GPIO 機能レジスタ 5

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

- **GPIO_BIT_0:** GPIO pin 0
- **GPIO_BIT_1:** GPIO pin 1
- **GPIO_BIT_2:** GPIO pin 2

- **GPIO_BIT_3:** GPIO pin 3
- **GPIO_BIT_4:** GPIO pin 4
- **GPIO_BIT_5:** GPIO pin 5
- **GPIO_BIT_6:** GPIO pin 6
- **GPIO_BIT_7:** GPIO pin 7

機能:
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 ポートの入出力設定

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

uint8_t

PullUp プルアップポートの許可/禁止設定
- **GPIO_PULLUP_ENABLE:** プルアップ許可
- **GPIO_PULLUP_DISABLE:** プルアップ禁止
- **GPIO_PULLUP_NONE:** プルアップ機能が無い、または設定変更しない

uint8_t
OpenDrain オープンドレインポート/CMOS ポートの設定
- **GPIO_OPEN_DRAIN_ENABLE:** オープンドレインポートに設定
- **GPIO_OPEN_DRAIN_DISABLE:** CMOS ポートに設定
- **GPIO_OPEN_DRAIN_NONE:** オープンドレイン機能がない、または設定変更しない

uint8_t
PullDown プルダウンポートの許可/禁止設定
- **GPIO_PULLDOWN_ENABLE:** プルダウン許可
- **GPIO_PULLDOWN_DISABLE:** プルダウン禁止
- **GPIO_PULLDOWN_NONE:** プルダウン機能がない、または設定変更しない

7. SBI

7.1 概要

本デバイスはシリアルバスインターフェース(SBI)が 1 本あり、マルチマスタ動作、または I2C バスで動作可能です。

I2C バスモードでは、SCL および SDA を通して外部デバイスと接続されます。
SBI チャンネルによりデータをフリーデータフォーマットで転送できます。フリーデータフォーマットでは、マスタモード時は送信、スレーブモード時は受信になります。

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

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpM37A_sbi.c
/Libraries/TX03_Periph_Driver/inc/tmpM37A_sbi.h

7.2 API 関数

7.2.1 関数一覧

- void SBI_Enable(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_Disable(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_SetI2CACK(TSB_SBI_TypeDef* ***SBIx***, FunctionalState ***NewState***);
- void SBI_InitI2C(TSB_SBI_TypeDef* ***SBIx***, SBI_InitI2CTypeDef* ***InitI2CStruct***);
- void SBI_SetI2CBitNum(TSB_SBI_TypeDef* ***SBIx***, uint32_t ***I2CBitNum***);
- void SBI_SWReset(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_ClearI2CINTReq(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_GenerateI2CStart(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_GenerateI2CStop(TSB_SBI_TypeDef* ***SBIx***);
- SBI_I2CState SBI_GetI2CState(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_SetIdleMode(TSB_SBI_TypeDef* ***SBIx***, FunctionalState ***NewState***);
- void SBI_SetSendData(TSB_SBI_TypeDef* ***SBIx***, uint32_t ***Data***);
- uint32_t SBI_GetReceiveData(TSB_SBI_TypeDef* ***SBIx***);
- void SBI_SetI2CFreeDataMode(TSB_SBI_TypeDef* ***SBIx,*** FunctionalState ***NewState***);

7.2.2 関数の種類

関数は、主に以下の 4 種類に分かれています。

1) 共通機能の設定:
   SBI_Enable(), SBI_Disable(), SBI_SetI2CACK(), SBI_SetI2CBitNum(), SBI_InitI2C()

2) 転送制御:
SBI_ClearI2CINTReq(), SBI_GenerateI2Cstart(),
SBI_GenerateI2Cstop(),SBI_SetSendData(), SBI_GetReceiveData()
3) ステータス確認:
SBI_GetI2CState()
4) その他:
SBI_SWReset(), SBI_SetIdleMode(), SBI_EnableI2CfreeDataMode()

7.2.3 関数仕様

7.2.3.1 SBI_Enable

シリアルバスインタフェース動作の許可

関数のプロトタイプ宣言:
void
SBI_Enable(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

機能:
SBI 動作を有効にします。

戻り値:
なし

7.2.3.2 SBI_Disable

シリアルバスインタフェース動作の禁止

関数のプロトタイプ宣言:
void
SBI_Disable(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

機能:
SBI 動作を無効にします。

戻り値:
なし

7.2.3.3 SBI_SetI2CACK

I2C バスモードにおける ACK 選択。

関数のプロトタイプ宣言:
void
SBI_SetI2CACK(TSB_SBI_TypeDef* ***SBIx***,

FunctionalState ***NewState***)

引数:
SBIx: SBI チャネルを指定します。

NewState: ACK の発生有無を選択します。
- **ENABLE**: 発生する。
- **DISABLE**: 発生しない。

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

戻り値:
なし

7.2.3.4 SBI_InitI2C

I2C バスモードにおける通信の初期化

関数のプロトタイプ宣言:
void
SBI_InitI2C(TSB_SBI_TypeDef* ***SBIx***,
SBI_InitI2CTypeDef* ***InitI2CStruct***)

引数:
SBIx: SBI チャネルを指定します。

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

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

戻り値:
なし

7.2.3.5 SBI_SetI2CBitNum

I2C バスモードにおける転送ビット数の選択。

関数のプロトタイプ宣言:
void
SBI_SetI2CBitNum(TSB_SBI_TypeDef* ***SBIx***,
uint32_t ***I2CBitNum***)

引数:
SBIx: SBI チャネルを指定します。

I2CBitNum: 転送ビット数(1～8)を選択します。
- **SBI_I2C_DATA_LEN_8:** データ長 8

- **SBI_I2C_DATA_LEN_1:** データ長 1
- **SBI_I2C_DATA_LEN_2:** データ長 2
- **SBI_I2C_DATA_LEN_3:** データ長 3
- **SBI_I2C_DATA_LEN_4:** データ長 4
- **SBI_I2C_DATA_LEN_5:** データ長 5
- **SBI_I2C_DATA_LEN_6:** データ長 6
- **SBI_I2C_DATA_LEN_7:** データ長 7

機能:
転送ビット数を選択します。

戻り値:
なし。

7.2.3.6 SBI_SWReset

ソフトウェアリセットの発生

関数のプロトタイプ宣言:
void
SBI_SWReset(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

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

戻り値:
なし。

7.2.3.7 SBI_ClearI2CINTReq

I2C バスモードにおける INTSBIx 割り込み要求解除

関数のプロトタイプ宣言:
void
SBI_ClearI2CINTReq(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

機能:
SBI 割り込み要求を解除します。

戻り値:
なし。

7.2.3.8 SBI_GenerateI2CStart

I2C バスモードにおけるスタート状態の発生

関数のプロトタイプ宣言:
void
SBI_GenerateI2CStart(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

機能:
I2c バスモードをマスタにし、I2c バスにスタートコンディションを出力します。

戻り値:
なし。

7.2.3.9 SBI_GenerateI2CStop

I2C バスモードにおけるストップ状態の発生。

関数のプロトタイプ宣言:
void
SBI_GenerateI2CStop(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

機能:
I2c バスモードをマスタにし、I2c バスにストップコンディションを出力します。

戻り値:
なし。

7.2.3.10 SBI_GetI2CState

I2C バスモードにおける SBI チャネルの状態の読み込み

関数のプロトタイプ宣言:
SBI_I2CState
SBI_GetI2CState(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

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

戻り値:
I2C モードでの SBI チャネルの状態。

7.2.3.11 SBI_SetIdleMode

IDLE モード時の動作の許可/禁止

関数のプロトタイプ宣言:
void
SBI_SetIdleMode(TSB_SBI_TypeDef* ***SBIx***,
FunctionalState ***NewState***)

引数:
SBIx: SBI チャネルを指定します。

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

機能:
NewState が **ENABLE** の場合 IDLE モードに遷移しても SBI チャネルは動作します。**DISABLE** を選択すると IDLE モード時に禁止されます。

戻り値:
なし。

7.2.3.12 SBI_SetSendData

データ送信

関数のプロトタイプ宣言:
void
SBI_SetSendData(TSB_SBI_TypeDef* ***SBIx***,
uint32_t ***Data***)

引数:
SBIx: SBI チャネルを指定します。

Data: 送信データ。(最大値は 0xFF です)

機能:
設定データを送信します。**SBI_GenerateI2Cstart()**の実行によりスタートコンディションを出力後、または ACK (通常は SBI 割り込みにより発生)受信後、データを送信します。

戻り値:
なし。

7.2.3.13 SBI_GetReceiveData

データ受信

関数のプロトタイプ宣言:
uint32_t
SBI_GetReceiveData(TSB_SBI_TypeDef* ***SBIx***)

引数:
SBIx: SBI チャネルを指定します。

機能:
データを受信します。**SBI_GenerateI2Cstart()**の実行によりスタートコンディションを出力後、または ACK (通常は SBI 割り込みにより発生)受信後、データを受信します。

戻り値:
受信データ。

7.2.3.14 SBI_SetI2CFreeDataMode

アドレス認識モードの指定

関数のプロトタイプ宣言:
void
SBI_SetI2CFreeDataMode(TSB_SBI_TypeDef* ***SBIx***,
FunctionalState ***NewState***)

引数:
SBIx: SBI チャネルを指定します。

NewState: アドレス認識モードを指定します。
- **ENABLE:** スレーブアドレスを認識しない。(フリーデータフォーマット)
- **DISABLE:** スレーブアドレスを認識する。

機能:
I2C モードにおけるデータフォーマットをフリーデータフォーマットにします。フリーデータフォーマットの場合、スレーブデバイスがデータ受信中にマスターデバイスは常にデータ送信を行います。転送データをノーマル I2C フォーマットにする場合は **SBI_InitI2C()**をコールしてください。

戻り値:
なし。

7.2.4 データ構造:

7.2.4.1 SBI_InitI2CTypeDef

メンバ:

uint32_t
***I2CSelfAddr*:** I2C モードにおけるスレーブアドレスを指定します。(0x01～0xFE)

uint32_t
***I2CDataLen*:** I2C モードにおける SBI チャネルの転送ビット数を指定します。
- **SBI_I2C_DATA_LEN_8:** データ長 8
- **SBI_I2C_DATA_LEN_1:** データ長 1
- **SBI_I2C_DATA_LEN_2:** データ長 2
- **SBI_I2C_DATA_LEN_3:** データ長 3
- **SBI_I2C_DATA_LEN_4:** データ長 4

- **SBI_I2C_DATA_LEN_5:** データ長 5
- **SBI_I2C_DATA_LEN_6:** データ長 6
- **SBI_I2C_DATA_LEN_7:** データ長 7

uint32_t
***I2CClkDiv*:** I2C 転送のソースクロックを選択します。
- **SBI_I2C_CLK_DIV_104:** fsys/104
- **SBI_I2C_CLK_DIV_136:** fsys/136
- **SBI_I2C_CLK_DIV_200:** fsys/200
- **SBI_I2C_CLK_DIV_328:** fsys/328
- **SBI_I2C_CLK_DIV_584:** fsys/584
- **SBI_I2C_CLK_DIV_1096:** fsys/1096
- **SBI_I2C_CLK_DIV_2120:** fsys/2120

FunctionalState
***I2CACKState*:** ACK の有効/無効を選択します。
- **ENABLE:** 有効。
- **DISABLE:** 無効。

7.2.4.2 SBI_I2CState

メンバ:

uint32_t
***All*:** I2C モードの全ての状態

Bit Fields:

uint32_t
***LastRxBit*:** 最終受信ビットモニタ

uint32_t
***GeneralCall*:** ゼネラルコール検出モニタ

uint32_t
***SlaveAddrMatch*:** スレーブアドレス一致モニタ

uint32_t
***ArbitrationLost*:** アービトレーションロスト検出モニタ

uint32_t
***INTReq*:** 割り込み要求状態モニタ

uint32_t
***BusState*:** バス状態モニタ

uint32_t
***TRx*:** 送信/受信選択状態モニタ

uint32_t
***MasterSlave*:** マスタ/スレーブ選択状態モニタ

8. OFD

8.1 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/tmpM37A_ofd.c
/Libraries/TX03_Periph_Driver/inc/tmpM37A_ofd.h

8.2 API 関数

8.2.1 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);

8.2.2 関数の種類

OFD 回路の初期化と設定:
OFD_SetRegWriteMode(), OFD_SetDetectionFrequency (), OFD_Enable (),
OFD_Disable ()

8.2.3 関数仕様

8.2.3.1 OFD_SetRegWriteMode

レジスタ書き込み制御

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

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:
本関数は、***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

8.2.3.2 OFD_Enable

OFD 動作の許可

関数のプロトタイプ宣言:
void
OFD_Enable(void)

引数:
なし

機能:
OFD 動作を許可します。

戻り値:
なし

8.2.3.3 OFD_Disable

OFD 動作の禁止

関数のプロトタイプ宣言:
void
OFD_Disable(void)

引数:
なし

機能:
OFD 動作を禁止します。

戻り値:
なし

8.2.3.4 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: PLL の状態を下記から選択します。

- **OFD_PLL_ON:** PLL ON 時
- **OFD_PLL_OFF:** PLL OFF 時

HigherDetectionCount: 検出周波数上限値

LowerDetectionCount: 検出周波数下限値

機能:
本関数は、検出周波数上限値、検出周波数下限値、検出周波数カウント値、PLL OFF, PLL ON を設定します。

戻り値:
なし

8.2.4 データ構造:

なし

9. TMRB

9.1 概要

本デバイスは、4 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, TMRB4, TMRB5, 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-bit interval timer mode
- 16-bit event counter mode
- 16-bit programmable pulse generation mode (PPG)
- External trigger Programmable pulse generation mode (PPG)
- 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
- 外部トリガ用プログラマブル矩形波出力(PPG)モード

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。

- パルス幅測定
- 外部トリガパルスからのワンショットパルス出力

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

全ドライバ API は、マクロ、データタイプ、構造、API 定義を格納する以下のファイルで構成されています。
/Libraries/TX03_Periph_Driver/src/tmpM37A_tmrb.c
/Libraries/TX03_Periph_Driver/inc/tmpM37A_tmrb.h

9.2 API 関数

9.2.1 関数一覧

- void TMRB_Enable(TSB_TB_TypeDef * ***TBx***);
- void TMRB_Disable(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetRunState(TSB_TB_TypeDef * ***TBx***, uint32_t ***Cmd***);
- void TMRB_Init(TSB_TB_TypeDef * ***TBx,*** TMRB_InitTypeDef * ***InitStruct***);
- void TMRB_SetCaptureTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***CaptureTiming***);
- void TMRB_SetFlipFlop(TSB_TB_TypeDef * ***TBx***, TMRB_FFOutputTypeDef * ***FFStruct***);
- TMRB_INTFactor TMRB_GetINTFactor(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetINTMask(TSB_TB_TypeDef * ***TBx***, uint32_t ***INTMask***);
- void TMRB_ChangeLeadingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***LeadingTiming***);
- void TMRB_ChangeTrailingTiming(TSB_TB_TypeDef * ***TBx***, uint32_t ***TrailingTiming***);
- uint16_t TMRB_GetUpCntValue(TSB_TB_TypeDef * ***TBx***);

- uint16_t TMRB_GetCaptureValue(TSB_TB_TypeDef * ***TBx***, uint8_t ***CapReg***);
- void TMRB_ExecuteSWCapture(TSB_TB_TypeDef * ***TBx***);
- void TMRB_SetIdleMode(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***);
- void TMRB_SetDoubleBuf(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***WriteRegMode***);
- void TMRB_SetExtStartTrg(TSB_TB_TypeDef * ***TBx***, FunctionalState ***NewState***, uint8_t ***TrgMode***);
- void TMRB_SetClkInCoreHalt(TSB_TB_TypeDef * ***TBx***, uint8_t ***ClkState***);

9.2.2 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(), TMRB_SetClkInCoreHalt ()

9.2.3 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_TB4, TSB_TB5, TSB_TB7.

9.2.3.1 TMRB_Enable

TMRB 動作の許可

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

引数:
***TBx*:** TMRB チャネルを指定します。

機能:
TMRB 動作を有効にします。

戻り値:
なし

9.2.3.2 TMRB_Disable

TMRB 動作の禁止

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

引数:

***TBx*:** TMRB チャネルを指定します。

機能:
TMRB 動作を無効にします。

戻り値:
なし

9.2.3.3 TMRB_SetRunState

カウンタ動作の設定

関数のプロトタイプ宣言:
void
TMRB_SetRunState(TSB_TB_TypeDef* ***TBx***,
uint32_t ***Cmd***)

引数:
***TBx*:** TMRB チャネルを指定します。

Cmd: カウンタ動作を選択します。
- **TMRB_RUN:** カウント
- **TMRB_STOP:** 停止&クリア

機能:
Cmd が **TMRB_RUN** の場合、アップカウンタがカウントを開始します。
Cmd が **TMRB_STOP** の場合、アップカウンタはカウントを停止し、同時にカウンタをクリアします。

戻り値:
なし

9.2.3.4 TMRB_Init

TMRB チャネルの初期化

関数のプロトタイプ宣言:
void
TMRB_Init(TSB_TB_TypeDef* ***TBx***,
TMRB_InitTypeDef* ***InitStruct***)

引数:
***TBx*:** TMRB チャネルを指定します。

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

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

戻り値:

なし

補足:
指定されたチャネルが TBxIN を持たない場合、***InitStruct->mode*** は **TMRB_INTERVAL_TIMER** しか選択できません。

9.2.3.5 TMRB_SetCaptureTiming

キャプチャタイミングの設定

関数のプロトタイプ宣言:
void
TMRB_SetCaptureTiming(TSB_TB_TypeDef* ***TBx***,
uint32_t ***CaptureTiming***)

引数:
***TBx*:** 以下から、TMRB チャネルを指定します。

TSB_TB0, TSB_TB4, TSB_TB7

CaptureTiming: キャプチャタイミングを選択します。
- **TMRB_DISABLE_CAPTURE**: キャプチャ機能を無効にします。
- **TMRB_CAPTURE_IN_RISING**: TBxIN↑
  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む
- **TMRB_CAPTURE_IN_RISING_FALLING**: TBxIN↑
  TBxIN↓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む
- **TMRB_CAPTURE_TIMPLS_RISING_FALLING**: TIMPLS↑ TIMPLS↓
  TIMPLS 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、
  TIMPLS 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む

補足:

TSB_TB0 の場合、キャプチャタイミングに **TMRB_CAPTURE_IN_RISING** と **TMRB_CAPTURE_IN_RISING_FALLING** を選択できません。

TSB_TB4、または TSB_TB7 の場合、キャプチャタイミングに **TMRB_CAPTURE_TIMPLS_RISING_FALLING** を選択できません。

機能:
CaptureTiming が **TMRB_CAPTURE_IN_RISING** の場合、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ます。
CaptureTiming が **TMRB_CAPTURE_IN_RISING_FALLING** の場合、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。
CaptureTiming が **TMRB_CAPTURE_TIMPLS_RISING_FALLING** の場合、TIMPLS 入力の立ち上がりエッジでキャプチャレジスタ(TBxCP0)にカウント値を取り

込み、TIMPLS 入力の立ち下がりエッジでキャプチャレジスタ(TBxCP1)にカウント値を取り込みます。

戻り値:
なし

9.2.3.6 TMRB_SetFlipFlop

フリップフロップ機能の設定

関数のプロトタイプ宣言:
void
TMRB_SetFlipFlop(TSB_TB_TypeDef* ***TBx***,
TMRB_FFOutputTypeDef* ***FFStruct***)

引数:
***TBx*:** TMRB チャネルを指定します。

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

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

戻り値:
なし

9.2.3.7 TMRB_GetINTFactor

割り込み要因の取得

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

引数:
***TBx*:** TMRB チャネルを指定します。

機能:
割り込み要因を取得します。

戻り値:
TMRB の割り込み要因:
MatchLeadintTiming(Bit0): 一致フラグ(TBxRG0)
MatchTrailingTiming(Bit1): 一致フラグ(TBxRG1)
OverFlow(Bit2): オーバーフローフラグ

補足:
異なる割り込み要因を処理する場合は、以下のように記述してください。

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

```
}

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

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

9.2.3.8 TMRB_SetINTMask

割り込みマスクの設定

関数のプロトタイプ宣言:
void
TMRB_SetINTMask(TSB_TB_TypeDef* ***TBx***,
uint32_t ***INTMask***)

引数:
***TBx*:** TMRB チャネルを指定します。

***INTMask*:** マスクする割り込みを選択します。
- **TMRB_MASK_MATCH_LEADINGTIMING_INT**: 一致フラグ(TBxRG0)
- **TMRB_MASK_MATCH_TRAILINGTIMING_INT**: 一致フラグ(TBxRG1)
- **TMRB_MASK_OVERFLOW_INT**: オーバーフロー割り込み。
- **TMRB_NO_INT_MASK**: マスクしない。

機能:
TMRB_MASK_MATCH_ TRAILINGTIMING _INT 選択時、アップカウンタ値と TBxRG1 が一致した場合、割り込みは発生しません。
TMRB_MASK_MATCH_ LEADINGTIMING _INT 選択時、アップカウンタ値と TBxRG0 が一致した場合、割り込みは発生しません。
TMRB_MASK_OVERFLOW_INT 選択時、オーバフロー発生時の割り込みは発生しません。
TMRB_NO_INT_MASK 選択時、割り込みマスクはすべてクリアされます。

戻り値:
なし

9.2.3.9 TMRB_ChangeLeadintTiming

デューティの設定

関数のプロトタイプ宣言:
void
TMRB_ChangeLeadingTiming(TSB_TB_TypeDef* ***TBx***,
uint32_t ***LeadingTiming***)

引数:
***TBx*:** TMRB チャネルを指定します。

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

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

戻り値:
なし。

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

9.2.3.10 TMRB_ChangeTrailingTiming

周期の設定。

関数のプロトタイプ宣言:
void
TMRB_ChangeTrailingTiming(TSB_TB_TypeDef* ***TBx***,
uint32_t ***TrailingTiming***)

引数:
***TBx*:** TMRB チャネルを指定します。

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

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

戻り値:
なし。

Note:
TrailingTiming は ***LeadingTiming*** より小さくすることはできません。 また TBxRG0/1 の値は PPG モードの TBxRG0<TBxRG1 を満たすように設定してください。

9.2.3.11 TMRB_GetUpCntValue

アップカウンタ値の読み込み

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

引数:
***TBx*:** TMRB チャネルを指定します。

機能:
アップカウンタ値の読み込みを行います。

戻り値:

アップカウンタ値

9.2.3.12 TMRB_GetCaptureValue

キャプチャレジスタの読み込み

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

引数:
***TBx*:** TMRB チャネルを指定します。

CapReg: キャプチャレジスタを選択します。
- **TMRB_CAPTURE_0:** キャプチャレジスタ 0。
- **TMRB_CAPTURE_1:** キャプチャレジスタ 1。***TBx*** は以下のチャネルのみ指定可能です。
  TSB_TB0, TSB_TB4, TSB_TB7.

機能:
CapReg が **TMRB_CAPTURE_0** の場合、キャプチャレジスタ 0 の値を読み込み、***CapReg*** が **TMRB_CAPTURE_1** の場合、キャプチャレジスタ 1 の値を読み込みます。

戻り値:
キャプチャされた値。

9.2.3.13 TMRB_ExecuteSWCapture

ソフトウェアキャプチャの実行

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

引数:
***TBx*:** TMRB チャネルを指定します。

機能:
キャプチャレジスタ 0 (TBxCP0)にカウント値を取り込みます。

戻り値:
なし

9.2.3.14 TMRB_SetIdleMode

IDLE 時の動作設定

関数のプロトタイプ宣言:
void

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

引数:
***TBx*:** TMRB チャネルを指定します。

NewState: IDLE 時の動作を指定します。
- **ENABLE:** 動作
- **DISABLE:** 停止

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

戻り値:
なし

9.2.3.15 TMRB_SetDoubleBuf

ダブルバッファ動作の制御

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

引数:
***TBx*:** TMRB チャネルを指定します。

***NewState*:** ダブルバッファの有効/無効を選択します。
- **ENABLE:** 許可。
- **DISABLE:** 禁止。

***WriteRegMode*:** ダブルバッファがイネーブルの場合のタイマレジスタ 0 および 1 への書き込みタイミングを指定します。
- **TMRB_WRITE_REG_SEPARATE:** タイマレジスタ 0 および 1 は個別に書き込みが可能です。一方のレジスタのみ書き込み準備が完了した場合も同様です。
- **TMRB_WRITE_REG_SIMULTANEOUS:** 両方のレジスタの書き込み準備が完了していない場合、タイマレジスタ 0 および 1 への書き込みはできません。

機能:
TBxRG0 レジスタ(***LeadingTiming***)と TBxRG1 (***TrailingTiming***)およびこれらのバッファは、同一アドレスへ割り付けられます。ダブルバッファがディセーブルの場合、同一の値はレジスタとそのバッファに書き込まれます。
ダブルバッファがイネーブルの場合、その値は各レジスタのバッファのみに書き込まれます。そのため初期値をレジスタ(TBxRG0 (***LeadingTiming***) および TBxRG1 (***TrailingTiming***))へ書き込むためには、ダブルバッファは **DISABLE** に設定してください。その後、イネーブルのダブルバッファには、レジスタへ書き込む次のデータが書き込まれます。データは対応する割り込みが発生した場合に自動的にロードされます。

戻り値:
なし

9.2.3.16 TMRB_SetExtStartTrg

外部トリガの設定

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

引数:
***TBx*:** TMRB チャネルを指定します。

***NewState*:** カウントスタート方法を選択します。
- **ENABLE:** 外部トリガ
- **DISABLE:** ソフトスタート

***TrgMode*:** 外部トリガのアクティブエッジを選択します。
- **TMRB_TRG_EDGE_RISING:** 立ち上がりエッジ
- **TMRB_TRG_EDGE_FALLING:** 立ち下りエッジ

機能:
外部トリガによる変換開始の有無とアクティブエッジの設定を行います。

戻り値:
なし

9.2.3.17 TMRB_SetClkInCoreHalt

デバッグ HALT 中のクロック動作

関数のプロトタイプ宣言:
void
TMRB_SetClkInCoreHalt (TSB_TB_TypeDef* ***TBx***, uint8_t ***ClkState***)

引数:
***TBx*:** TMRB チャネルを指定します。

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

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

戻り値:
なし

9.2.4 データ構造:

9.2.4.1 TMRB_InitTypeDef

メンバ:

uint32_t
Mode: タイマモードを選択します。

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

補足: TBxIN を持たないチャネルは ***InitStruct->mode*** に **TMRB_INTERVAL_TIMER** のみ指定可能です。

uint32_t
ClkDiv: インターバルタイマのソースクロックの分周を選択します。

- **TMRB_CLK_DIV_2**: fperiph / 2
- **TMRB_CLK_DIV_8**: fperiph / 8
- **TMRB_CLK_DIV_32**: fperiph / 32
- **TMRB_CLK_DIV_64**: fperiph / 64
- **TMRB_CLK_DIV_128**: fperiph / 128
- **TMRB_CLK_DIV_256**: fperiph / 256
- **TMRB_CLK_DIV_512**: fperiph / 512

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

uint32_t
UpCntCtrl: アップカウンタの動作を選択します。

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

uint32_t
LeadingTiming: TBnRG0 に書き込むデューティ (最大 0xFFFF)。**TrailingTiming** 以上の値を設定できません。

9.2.4.2 TMRB_FFOutputTypeDef

メンバ:

FlipflopCtrl: フリップフロップのレベルを選択します。

- **TMRB_FLIPFLOP_INVERT:** TBxFF0 の値を反転(ソフト反転)します。
- **TMRB_FLIPFLOP_SET:** TBxFF0 を"1"にセットします。
- **TMRB_FLIPFLOP_CLEAR:** TBxFF0 を"0"にクリアします。

uint32_t
FlipflopReverseTrg: 以下から、フリップフロップの反転トリガを選択します。

- **TMRB_DISALBE_FLIPFLOP**: 反転トリガを無効にします。
- **TMRB_FLIPFLOP_TAKE_CATPURE_0**: アップカウンタの値がキャプチャレジスタ 0 に取り込まれた時にタイマフリップフロップを反転します。
- **TMRB_FLIPFLOP_TAKE_CATPURE_1**: アップカウンタの値がキャプチャレジスタ 1 に取り込まれた時にタイマフリップフロップを反転します。
- **TMRB_FLIPFLOP_MATCH_ 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。

9.2.4.3 TMRB_INTFactor

メンバ:

uint32_t
All: TMRB 割り込み要因
Bit
uint32_t
MatchLeadingTiming: 1 デューティとの一致検出

uint32_t
MatchTrailingTiming : 1　周期との一致検出

uint32_t
OverFlow : 1 オーバーフロー

uint32_t
Reserverd : 29　-

10. SIO/UART

10.1 概要

本デバイスは 2 つの動作モードを持っています。チャネル0は UART モード(非同期通信)と SIO モード(同期通信)を選択することができ、チャネル 1 は UART モードのみ設定可能です。
UART モードでは、7, 8, 9 ビット長のデータを選択可能です。

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/tmpM37A_uart.c
/Libraries/TX03_Periph_Driver/inc/tmpM37A_uart.h

10.2 API 関数

10.2.1 関数一覧

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

- void UART_TxFIFOINTSel(TSB_SC_TypeDef * UARTx, uint32_t TxINTCondition);
- void UART_TxFIFOClear(TSB_SC_TypeDef * UARTx);
- void UART_TxBufferClear(TSB_SC_TypeDef * UARTx);
- uint32_t UART_GetRxFIFOFillLevelStatus(TSB_SC_TypeDef * UARTx);
- uint32_t UART_GetRxFIFOOverRunStatus(TSB_SC_TypeDef * UARTx);
- uint32_t UART_GetTxFIFOFillLevelStatus(TSB_SC_TypeDef * UARTx);
- uint32_t UART_GetTxFIFOUnderRunStatus(TSB_SC_TypeDef * UARTx);
- void SIO_SetInputClock(TSB_SC_TypeDef * SIOx, uint32_t Clock)
- void SIO_Enable(TSB_SC_TypeDef* SIOx)
- void SIO_Disable(TSB_SC_TypeDef* SIOx)
- void SIO_Init(TSB_SC_TypeDef* SIOx, uint32_t IOClkSel,
  UART_InitTypeDef* InitStruct)
- uint8_t SIO_GetRxData(TSB_SC_TypeDef* SIOx)
- void SIO_SetTxData(TSB_SC_TypeDef* SIOx, uint8_t Data)

10.2.2 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(),
   UART_SetInputClock, SIO_Enable(), SIO_Disable(), SIO_SetInuptClock(), SIO_Init()
2) 送受信設定とエラー確認:
   UART_GetBufState(), UART_GetRxData(), UART_SetTxData(),
   UART_GetErrState(), SIO_GetRxData() and SIO_SetTxData
3) その他:
   UART_SWReset(), UART_SetWakeUpFunc(), UART_SetIdleMode()
4) FIFO モードの設定:
   UART_FIFOConfig(), UART_SetFIFOTransferMode(), UART_RxFIFOINTCtrl(),
   UART_TxFIFOINTCtrl(), UART_RxFIFOByteSel(), UART_RxFIFOFillLevel(),
   UART_RxFIFOINTSel(), UART_RxFIFOClear(), UART_TxFIFOFillLevel(),
   UART_TxFIFOINTSel(), UART_TxFIFOClear(), UART_TxBufferClear(),
   UART_GetRxFIFOFillLevelStatus(), UART_GetRxFIFOOverRunStatus(),
   UART_GetTxFIFOFillLevelStatus(), UART_GetTxFIFOUnderRunStatus()

10.2.3 関数仕様

補足: 引数に記述している “TSB_SC_TypeDef* ***UARTx***” は、以下から選択してください。
UART0, **UART1**.
また、“TSB_SC_TypeDef* ***SIOx***” は、以下から選択してください。
SIO0

10.2.3.1 UART_Enable

UART 動作の許可

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

引数:
UARTx: UART チャネルを指定します。

機能:
UART 動作を許可します。

戻り値:
なし

10.2.3.2 UART_Disable

UART 動作の禁止

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

引数:
UARTx: UART チャネルを指定します。

機能:
UART 動作を禁止します。

戻り値:
なし

10.2.3.3 UART_GetBufState

送受信バッファ状態の読み込み

関数のプロトタイプ宣言:
WorkState
UART_GetBufState(TSB_SC_TypeDef* ***UARTx***,
uint8_t ***Direction***)

引数:
UARTx: UART チャネルを指定します。

Direction: 送信/受信を選択します。
- **UART_RX**: 受信
- **UART_TX**: 送信

機能:
Direction が **UART_RX** の場合、以下の受信バッファの状態を返します。
DONE:受信データはバッファに保存済み
BUSY:データ受信中
Direction が **UART_TX** の場合、以下の送信バッファの状態を返します。
DONE:バッファ中のデータは送信済み
BUSY:データ送信中

戻り値:
DONE: バッファリード/ライト可能状態
BUSY: 送受信中。

10.2.3.4 UART_SWReset

ソフトウェアリセットの実行

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

引数:
UARTx: UART チャネルを指定します。

機能:
ソフトウェアリセットを実行します。

戻り値:
なし

10.2.3.5 UART_Init

UART チャネルの初期化

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

引数:
UARTx: UART チャネルを指定します。

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

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

戻り値:
なし

10.2.3.6 UART_GetRxData

受信データの読み込み

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

引数:
UARTx: UART チャネルを指定します。

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

戻り値:
受信データです。データ範囲は 0x00～0x1FF です。

10.2.3.7 UART_SetTxData

送信データの設定

関数のプロトタイプ宣言:
void
UART_SetTxData(TSB_SC_TypeDef* ***UARTx***,
uint32_t ***Data***)

引数:
UARTx: UART チャネルを指定します。

Data: 送信データ(7 ビット、8 ビット、9 ビット)

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

戻り値:
なし

10.2.3.8 UART_DefaultConfig

デフォルト構成での初期化

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

引数:
UARTx: UART チャネルを指定します。

機能:
以下の構成で初期化します:
ボーレート: 115200 bps
データ長: 8 ビット
ストップビット: 1 ビット
パリティ: なし
フローコントロール: なし
送受信有効。ボーレートジェネレータはソースクロックとして使用。

戻り値:
なし

10.2.3.9 UART_GetErrState

転送エラーフラグの読み出し

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

引数:
UARTx: UART チャネルを指定します。

機能:
転送エラーフラグを読み出します。

戻り値:
UART_NO_ERR: エラーなし
UART_OVERRUN: オーバーランエラー
UART_PARITY_ERR: パリティエラー
UART_FRAMING_ERR: フレーミングエラー
UART_ERRS: 上記の 2 つ以上のエラーが発生している

10.2.3.10 UART_SetWakeUpFunc

9 ビットモード時のウェイクアップ機能の設定

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

引数:
UARTx: UART チャネルを指定します。

NewState:ウェイクアップ機能の有効/無効を選択します。

- **ENABLE**: 有効
- **DISABLE**: 無効

機能:
9 ビットモード時のウェイクアップ機能を設定します。
NewState が **ENABLE** の場合、ウェイクアップ機能を有効に、
NewState が **DISABLE** の場合、ウェイクアップ機能を無効に設定します。
ウェイクアップ機能は、9 ビットモード時のみ機能します。

戻り値:
なし

10.2.3.11 UART_SetIdleMode

IDLE 時の動作

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

引数:
UARTx: UART チャネルを指定します。

NewState: IDLE 時の動作を選択します。

- **ENABLE**: 動作
- **DISABLE**: 停止

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

戻り値:
なし

10.2.3.12 UART_SetInputClock

入力クロックの設定

関数のプロトタイプ宣言:
void
UART_SetInputClock(TSB_SC_TypeDef * ***UARTx***, uint8_t ***ClkDivider***)

引数:
UARTx: UART チャネルを指定します。

ClkDivider: 以下から、プリスケーラの入力クロックを選択します。

- **UART_DIVIDE_1_1**: ΦT0
- **UART_DIVIDE_1_2**: ΦT0/2

機能:
プリスケーラの入力クロックを選択します。

戻り値:
なし

10.2.3.13 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

10.2.3.14 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

10.2.3.15 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

10.2.3.16 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

10.2.3.17 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

10.2.3.18 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

10.2.3.19 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

10.2.3.20 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

10.2.3.21 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

10.2.3.22 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

10.2.3.23 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

10.2.3.24 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

10.2.3.25 UART_TxBufferClear

送信バッファクリア

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

引数:
UARTx: UART チャネルを指定します。

機能:
送信バッファをクリアします。

戻り値:
なし

10.2.3.26 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

10.2.3.27 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

10.2.3.28 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

10.2.3.29 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

10.2.3.30 UART_SetInputClock

入力クロックの設定

関数のプロトタイプ宣言:
void
UART_SetInputClock (TSB_SC_TypeDef * UARTx,
uint32_t clock)

引数:
UARTx: UART チャネルを指定します。

Clock : 以下から、プリスケーラの入力クロックを選択します。
0 : ΦT0/2
1 :ΦT0

機能:
プリスケーラの入力クロックを選択します。

戻り値:

なし

10.2.3.31 SIO_SetInputClock

入力クロックの設定

関数のプロトタイプ宣言:
void
SIO_SetInputClock (TSB_SC_TypeDef * SIOx,
uint32_t Clock)

引数:
SIOx: SIO チャネルを指定します。

Clock : 以下から、プリスケーラの入力クロックを選択します。
SIO_CLOCK_T0_HALF : ΦT0/2
SIO_CLOCK_T0 :ΦT0

機能:
プリスケーラの入力クロックを選択します。

戻り値:
なし

10.2.3.32 SIO_Enable

SIO 動作の許可

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

引数:
SIOx: SIO チャネルを指定します。

機能:
SIO 動作を許可します。

戻り値:
なし

10.2.3.33 SIO_Disable

SIO 動作の禁止

関数のプロトタイプ宣言:
void
SIO _Disable(TSB_SC_TypeDef* ***SIOx***)

引数:
SIOx: SIO チャネルを指定します。

機能:
SIO 動作を禁止します。

戻り値:
なし

10.2.3.34 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

10.2.3.35 SIO_GetRxData

受信用バッファの取得

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

引数:
SIOx: SIO チャネルを指定します。

機能:
受信用バッファを取得します。

戻り値:
受信用バッファ(値の範囲は 0x00 ~ 0xFF です)

10.2.3.36 SIO_SetTxData

送信用バッファの設定

関数のプロトタイプ宣言:

void
SIO_SetTxData(TSB_SC_TypeDef* ***SIOx***,
uint8_t ***Data***)

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

機能:
送信用バッファを指定します。

戻り値:
なし

10.2.4 データ構造:

10.2.4.1 UART_InitTypeDef

メンバ:

uint32_t
BaudRate : UART 通信ボーレートを 2400(bps) から 115200(bps) に設定。(*)

uint32_t
DataBits : 転送ビット数を選択します。
- **UART_DATA_BITS_7** : 7 ビットモード
- **UART_DATA_BITS_8** : 8 ビットモード
- **UART_DATA_BITS_9** :9 ビットモード

uint32_t
StopBits : ストップビット長を選択します。
- **UART_STOP_BITS_1:** 1 ビット
- **UART_STOP_BITS_2 :** 2 ビット

uint32_t
Parity : パリティを選択します。
- **UART_NO_PARITY :** パリティなし
- **UART_EVEN_PARITY**: 偶数(Even) パリティ
- **UART_ODD_PARITY**: 偶数(Even) パリティ

uint32_t
Mode: 転送モードを選択します。送受信の場合は、送信と受信を OR 演算子によって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 :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。

10.2.4.2 SIO_InitTypeDef

メンバ:
uint32_t
InputClkEdge: 入力クロックエッジを選択します。
- **SIO_SCLKS_TXDF_RXDR**: SCLKx の立ち下がりエッジで送信バッファのデータを 1bit ずつ TXDx 端子へ出力します。SCLKx の立ち上がりエッジで RXDx 端子のデータを 1bit ずつ受信バッファに取り込みます。この時、SCLKx は High レベルからスタートします。
- **SIO_SCLKS_TXDR_RXDF**: SCLKx の立ち上がりエッジで送信バッファのデータを 1bit ずつ TXDx 端子へ出力します。SCLKx の立ち下がりエッジで RXDx 端子のデータを 1bit ずつ受信バッファに取り込みます。この時、SCLKx は Low レベルからスタートします。

TIDLE: 最終ビット出力後の TXDx 端子の状態を選択します。
- **SIO_TIDLE_LOW**: "Low"出力保持
- **SIO_TIDLE_HIGH**: "High"出力保持
- **SIO_TIDLE_LAST**: 最終ビット保持

uint32_t
TXDEMP: 以下から、クロック入力モード時、アンダーランエラーが発生したときの TXDx 端子の状態を選択します。
- **SIO_TXDEMP_LOW**: "Low"出力
- **SIO_TXDEMP_HIGH**: "High"出力

uint32_t
EHOLDTime: 以下から、クロック入力モードの TXDx 端子の最終ビットホールド時間を選択します。
- **SIO_EHOLD_FC_2**: 2/fc
- **SIO_EHOLD_FC_4**: 4/fc
- **SIO_EHOLD_FC_8**: 8/fc
- **SIO_EHOLD_FC_16**: 16/fc
- **SIO_EHOLD_FC_32**: 32/fc
- **SIO_EHOLD_FC_64**: 64/fc
- **SIO_EHOLD_FC_128**: 128/fc

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_TS0**: φTS0
- **SIO_BR_CLOCK_TS2**: φTS2
- **SIO_BR_CLOCK_TS8**: φTS8
- **SIO_BR_CLOCK_TS32**: φTS32

uint32_t
Divider: 分周値"N"を選択します。
- **SIO_BR_DIVIDER_16**: 16 分周
- **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

11. VLTD

11.1 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 機能の許可/禁止、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/tmpM37A_vltd.c
/Libraries/TX03_Periph_Driver/inc/tmpM37A_vltd.h

11.2 API 関数

11.2.1 関数一覧

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

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) VLTD の許可/禁止:
   VLTD_Enable()、VLTD_Disable()
2) 検出電圧の設定:
   VLTD_SetVoltage()

11.2.3 関数仕様

11.2.3.1 VLTD_Enable

電圧検出の許可

関数のプロトタイプ宣言:
void
VLTD_Enable(void)

引数:
なし

機能:
電圧検出を許可します。

戻り値:
なし

11.2.3.2 VLTD_Disable

電圧検出の禁止

関数のプロトタイプ宣言:
void
VLTD_Disable(void)

引数:
なし

機能:
電圧検出を禁止します。

戻り値:
なし

11.2.3.3 VLTD_SetVoltage

検出電圧レベルの選択

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

引数:
***Voltage*:** 以下から検出電圧レベルを選択します。
- **VLTD_DETECT_VOLTAGE_41:** 4.1V ± 0.2V
- **VLTD_DETECT_VOLTAGE_44:** 4.4V ± 0.2V
- **VLTD_DETECT_VOLTAGE_46:** 4.6V ± 0.2V

機能:
検出電圧レベルを選択します。

戻り値:
なし

11.2.4 データ構造:

なし

12. WDT

12.1 概要

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

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

本ドライバは、以下のファイルで構成されています。
\Libraries\TX03_Periph_Driver\src\tmpM37A_wdt.c
\Libraries/TX03_Periph_Driver\inc\tmpM37A_wdt.h

12.2 API 関数

12.2.1 関数一覧

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

12.2.2 関数の種類

関数は、主に以下の 2 種類に分かれています。
1) ウォッチドッグタイマ設定:
WDT_SetDetectTime(),WDT_SetOverflowOutput(), WDT_Init(), WDT_Enable(), WDT_Disable(), WDT_WriteClearCode()
2) IDLE モード時の開始・停止:
WDT_SetIdleMode()

12.2.3 関数仕様

12.2.3.1 WDT_SetDetectTime

WDT 検出時間の設定

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

引数:
***DetectTime*:** 以下から検出時間を選択します。
- **WDT_DETECT_TIME_EXP_15:** 2^15/fsys

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

機能:
WDT の検出時間を設定します。

戻り値:
なし

12.2.3.2 WDT_SetIdleMode

IDLE 時の動作選択

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

引数:
NewState: 以下から IDLE 時の WDT 動作を選択します。
- **ENABLE**: 動作
- **DISABLE**. 停止

機能:
本関数は、IDLE モード時の WDT カウンタの動作を設定します。
NewState が **ENABLE** の時は WDT カウンタ停止
NewState が **DISABLE** の時は WDt カウンタ作動

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

戻り値:
なし

12.2.3.3 WDT_SetOverflowOutput

暴走検出後の動作選択

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

引数:
OverflowOutput: 以下から暴走検出後の動作を選択します。
- **WDT_NMIINT**: INTWDT 割り込み要求を発生します。
- **WDT_WDOUT**: マイコンをリセットします。

機能:

カウンタオーバーフロー時の NMI 割り込み/リセットの設定を行います。***OverflowOutput***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

12.2.3.4 WDT_Init

WDT の初期化

関数のプロトタイプ宣言:
void
WDT_Init (WDT_InitTypeDef* ***InitStruct***)

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

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

戻り値:
なし

12.2.3.5 WDT_Enable

WDT 動作の許可

関数のプロトタイプ宣言:
void
WDT_Enable(void)

引数:
なし

機能:
WDT 動作を許可します。

戻り値:
なし

12.2.3.6 WDT_Disable

WDT 動作の禁止

関数のプロトタイプ宣言:
void
WDT_Disable(void)

引数:
なし

機能:
WDT 動作を禁止します。

戻り値:
なし

12.2.3.7 WDT_WriteClearCode

クリアコードの書き込み

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

引数:
なし

機能:
クリアコードをライトします。

戻り値:
なし

12.2.4 データ構造:

12.2.4.1 WDT_InitTypeDef

メンバ:

uint32_t
DetectTime 以下から検出時間を選択します。

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

uint32_t
OverflowOutput 以下から、カウンタオーバーフロー時の動作を選択します。

- **WDT_WDOUT:** マイコンをリセットします。
- **WDT_NMIINT:** INTNMI 割り込み要求を発生します。

13. PMD

13.1 概要

本デバイスはモータ制御回路(PMD)を1 チャネル内蔵しています。本デバイスのPMD はベクトルエンジン(VE+)やアナログ/デジタルコンバータ(ADC)と連携動作してベクトル制御などの3 相モータ制御を実現します。パルス幅変調回路、通電制御および同期トリガ生成回路はVE からの指令で動作可能で、同期トリガ生成回路はADC に変換開始指令ができます。

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 キャリアが共通で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。
- 保護回路ではEMG 入力、OVV 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\tmpM37A_pmd.c
/Libraries/TX03_Periph_Driver\inc\tmpM37A_pmd.h

13.2 API 関数

13.2.1 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_EnableOVV(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_DisableOVV(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetOVVNoiseElimination(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***NoiseElimination***);
- void PMD_SetADCMonitorInput(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Monitor***,
  FunctionalState ***NewState***);
- void PMD_SetOVVMode(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Mode***);
- void PMD_SetOVVInputSrc(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***Source***);
- void PMD_SetOVVAutoRelease(TSB_PMD_TypeDef * ***PMDx***,
  FunctionalState ***NewState***);
- uint32_t PMD_GetOVVAbnormalLevel(TSB_PMD_TypeDef * ***PMDx***);
- void PMD_SetAutoSwitchCtrl(TSB_PMD_TypeDef * ***PMDx***,
  FunctionalState ***NewState***);
- void PMD_SetPWMEdge(TSB_PMD_TypeDef * ***PMDx***,
  uint32_t ***PMDPhase***,
  uint32_t ***Edge***);

◆ void PMD_SetBufferUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***UpdateTime***);
◆ void PMD_SetTrgSyncTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***SyncTime***);
◆ void PMD_SetTrgUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***UpdateTime***);

13.2.2 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(), PMD_SetAutoSwitchCtrl(), PMD_SetPWMEdge(),
PMD_SetBufferUpdateTime()
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()
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(), PMD_SetTrgSyncTime(),
PMD_SetTrgUpdateTime()
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()

13.2.3 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。
PMD1

13.2.3.1 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

13.2.3.2 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

13.2.3.3 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

13.2.3.4 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

13.2.3.5 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)

13.2.3.6 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

13.2.3.7 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

13.2.3.8 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

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

13.2.3.10 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

13.2.3.11 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

13.2.3.12 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

13.2.3.13 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

13.2.3.14 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

13.2.3.15 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

13.2.3.16 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

13.2.3.17 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

13.2.3.18 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"

13.2.3.19 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

13.2.3.20 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

13.2.3.21 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

13.2.3.22 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

13.2.3.23 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。

13.2.3.24 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。

13.2.3.25 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

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

13.2.3.26 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

13.2.3.27 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

13.2.3.28 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

13.2.3.29 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

13.2.3.30 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

13.2.3.31 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

13.2.3.32 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

13.2.3.33 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

13.2.3.34 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

13.2.3.35 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

13.2.3.36 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

13.2.3.37 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"

13.2.3.38 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

13.2.3.39 PMD_SetAutoSwitchCtrl

VE レジスタと PMD レジスタの自動切り替え許可/禁止

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

引数:
PMDx: PMD チャネルを指定します。

NewState: 自動切り替え許可/禁止を選択します。
- **ENABLE:** 2 レジスタ切り替え許可
- **DIABLE:** 2 レジスタ切り替え禁止

機能:
VE レジスタと PMD レジスタの自動切り替えを許可/禁止します。

戻り値:
なし

13.2.3.40 PMD_SetPWMEdge

PWM エッジ設定

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

引数:
PMDx: PMD チャネルを指定します。

PMDPhase: 3 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:** すべての相

Edge: エッジを選択します。
- **PMD_PWM_EDGE_UNFIXED:** エッジ固定解除
- **PMD_PWM_RISING_EDGE_FIXED:** PWM 立ち上がりエッジ固定
- **PMD_PWM_FALLING_EDGE_FIXED:** PWM 立ち下りエッジ固定

機能:
PWM エッジを設定します。

戻り値:
なし

13.2.3.41 PMD_SetBufferUpdateTime

Duty コンペアレジスタと PWM 周期レジスタのダブルバッファ更新タイミングの設定

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

引数:
PMDx: PMD チャネルを指定します。

UpdateTime: 更新タイミングを選択します。
- **PMD_UPDATE_TIME_WRITE:** 割り込み周期設定(INTPRD)によります。
- **PMD_UPDATE_TIME_MIN:** PWM キャリアボトムで更新
- **PMD_UPDATE_TIME_MAX:** PWM キャリアピークで更新
- **PMD_UPDATE_TIME_MIN_MAX:** PWM キャリアのピークとボトムで更新

機能:
Duty コンペアレジスタと PWM 周期レジスタのダブルバッファ更新タイミングを設定します。

戻り値:
なし

13.2.3.42 PMD_SetTrgSyncTime

トリガ同期設定

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

引数:
PMDx: PMD チャネルを指定します。

SyncTime: 同期タイミングを選択します。
- **PMD_TRGGER_TIME_ASYNC:** 非同期
- **PMD_TRGGER_TIME_ENC:** INTENC (ENC 割り込み要求)発生時
- **PMD_TRGGER_TIME_TMRB:** INTTB00 (TMRB 割り込み要求)発生時

機能:
トリガ同期設定を行います。

戻り値:
なし

13.2.3.43 PMD_SetTrgUpdateTime

通電制御レジスタの更新タイミング選択

関数のプロトタイプ宣言:
void

PMD_SetTrgUpdateTime(TSB_PMD_TypeDef * ***PMDx***,
uint32_t ***UpdateTime***);

引数:
PMDx: PMD チャネルを指定します。

UpdateTime: 通電制御レジスタの更新タイミングを選択します。
- **PMD_UPDATE_TIME_WRITE:** PWM 非同期
- **PMD_UPDATE_TIME_MIN:** PWM キャリアボトム
- **PMD_UPDATE_TIME_MAX:** PWM キャリアピーク
- **PMD_UPDATE_TIME_MIN_MAX:** キャリアピークおよびキャリアボトムを選択します。

機能:
通電制御レジスタの更新タイミングを選択します。

戻り値:
なし

13.2.4 データ構造

13.2.4.1 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。