TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM375FSDMG)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M375UE-01J

本製品取り扱い上のお願い

- ソフトウエア使用権許諾契約書の同意無しに使用しないで下さい。

目次

1 はしがき....1
2 概要....1
3 使用する機能....1
4 端子用途....3
5 開発環境....4
6 機能説明....5
6-1 動作モード選択....5
6-2 ADC....5
6-3 CG....6
6-4 ENC....6
6-5 FLASH....7
6-6 GPIO....7
6-7 VLTD....7
6-8 OFD....7
6-9 SBI....8
6-9-1 SBI スレーブ....8
6-9-2 SBI マスタ....8
6-10 TMRB....9
6-10-1 汎用タイマ....9
6-10-2 プログラマブル矩形波出力(PPG)....9
6-11 SIO/UART....9
6-11-1 リターゲット....9
6-11-2 UART FIFO....9
6-11-3 SIO....9
6-12 WDT....10
6-13 PMD....10
6-13-1 例: 位相出力....10
7 ソフトウェア....11
7-1 ADC....12
7-1-1 例: ADC データ読み出し....12
7-2 CG....14
7-2-1 例: パワーモード変更....14
7-3 ENC....17
7-3-1 例: 回転検出....17
7-4 FLASH....21
7-4-1 例: Flash ROM への書き込み....21
7-5 GPIO....24
7-5-1 例: GPIO データ読み出し....24
7-6 VLTD....25
7-6-1 例: VLTD リセット....25
7-7 OFD....27
7-7-1 例: OFD リセット....27
7-8 SBI....30

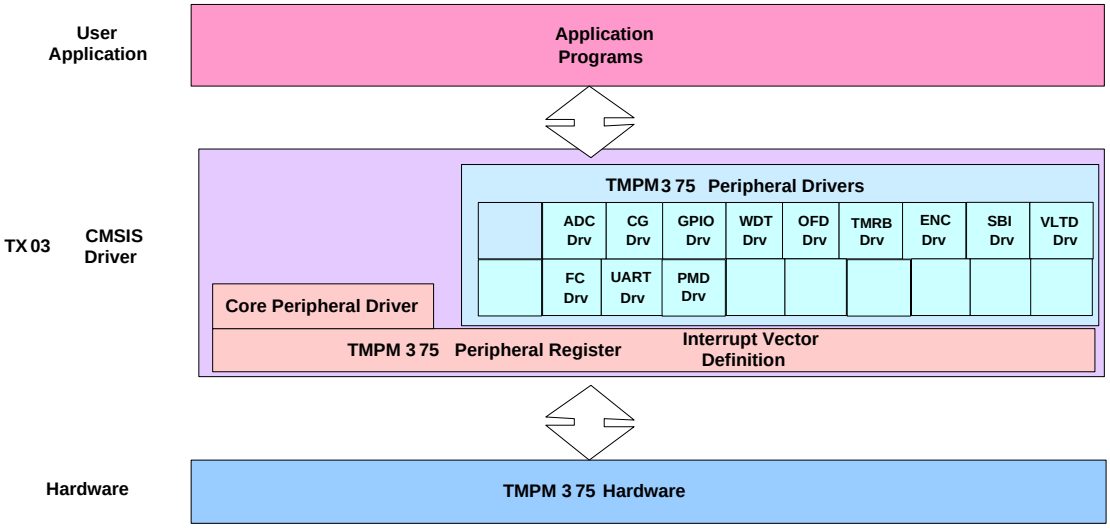
7-8-1 例: SBI スレーブ.....30
7-8-2 例: SBI マスタ.....34
7-9 16 ビットタイマ(TMRB).....38
7-9-1 例: 汎用タイマ.....38
7-9-2 例: プログラマブル矩形波出力(PPG).....41
7-10 SIO/UART.....44
7-10-1 例: リターゲット.....44
7-10-2 例: UART の FIFO デモ.....48
7-10-3 例: SIO マスタ.....51
7-10-4 例: SIO スレーブ.....53
7-11 ウォッチドッグタイマ(WDT).....55
7-11-1 例:ウォッチドッグタイマ.....55
7-12 PMD.....55
7-12-1 例: 位相出力.....55

1 はしがき

本アプリケーションノートのサンプルプログラムは、東芝製マイコンTMPM375FSDMG用です。各サンプルプログラムは、主なMCU内蔵機能を単独で実行するように出来ています。サンプルプログラム内の一部を取り出して再利用することで、希望する機能を動作させることができます。

2 概要

TX03ペリフェラルドライバを下記のように使用します。



3 使用する機能

機能	チャネル	使用／未使用
クロック／モード制御（CG）	クロックギア	使用
	PLL	PLL4 逓倍
低消費電力モード	-	使用: STOP モードのみ
SysTick	-	未使用
ウォッチドッグタイマ（WDT）	-	使用: WDT デモ
外部割り込み (INT)	INT6	未使用
	INT7	使用: CG デモにおける STOP モードからの復帰

機能	チャネル	使用／未使用
	INTC	未使用
シリアルチャネル (SIO/UART)	SIO0	使用: UART リターゲットデモ, UART FIFO デモ, SIO デモ
	SIO1	使用: UART FIFO デモ
16 ビットタイマ (TMRB)	TMRB0	使用: 汎用タイマ(TMRB)
	TMRB4	未使用
	TMRB5	未使用
	TMRB7	使用: PPG 出力(TMRB)
周波数検知回路(OFD)	-	使用: OFD デモ
パワーオンリセット回路 (POR)	-	未使用
電圧検出回路(VLTD)	-	使用: VLTD デモ
12ビットA/Dコンバータ (ADC)	AIN9	使用: ADC データ読み出し
	AIN10	未使用
	AIN11	未使用
	AIN12	未使用
シリアルバスインタフェース(SBI)	SBI	使用: SBI デモ
エンコーダ入力機能 (ENC)	ENC0	使用: ENC デモ
モータ制御回路(PMD)	PMD1	使用: PMD 位相出力

4 端子用途

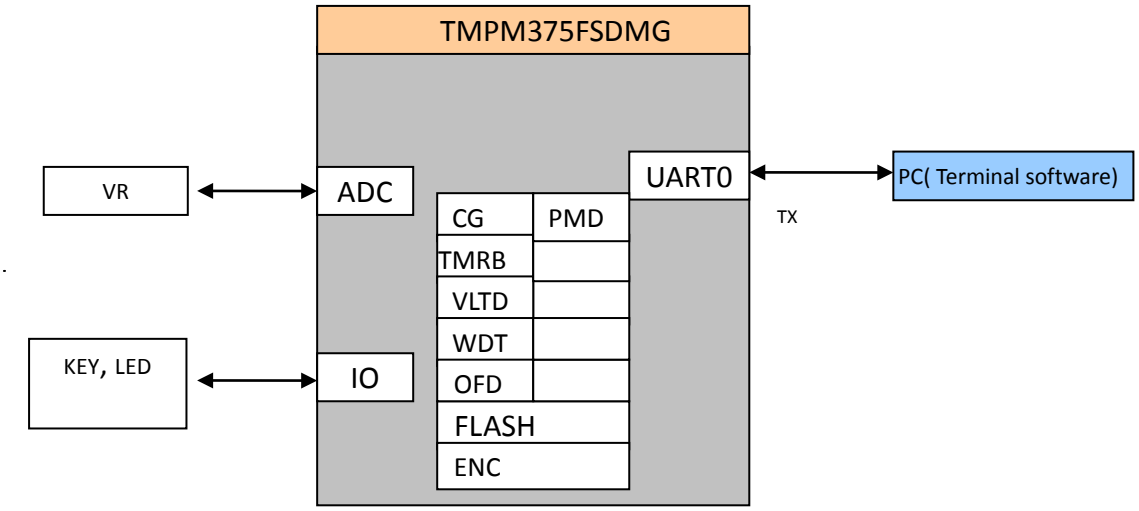
本サンプルプログラムは、TMPM375FSDMG評価ボードを用いてテストされています。以下に、端子用途を説明します。

下表はサンプルプログラムで使用する端子表です。

Pin No.	端子名	用途
1	AVSSB/VREFLB/AMPVSS	AD コンバータ用 GND
2	AINB9/PJ6	AINB9 入力
3	AINB10/PJ7	SW1/5
4	AINB11/PK0	SW2/6
5	AINB12/PK1	SW3/7
6	AVDD5B/VREFHB/AMPVDD5	ADコンバータ用基準電源
7	MODE	未使用
8	U0/PG0	LED0/ PMD UO 出力
9	X0/PG1	LED1/ PMD XO 出力
10	V0/PG2	LED2/ PMD VO 出力
11	Y0/PG3	LED3/ PMD YO 出力
12	W0/PG4	PMD WO 出力
13	Z0/PG5	PMD ZO 出力
14	EMG/OVV/PG6	PMD EMG 入力
15	RESET	リセット入力
16	PB3/TMS/SWDIO/(RXD1)	PMD サンプル用 LED4
17	PB4/TCK/SWCLK/(TXD1)	未使用
18	PB5TD0/SWV/SCK0/(SDA0)	SBI Tx/Rx
19	PB6/TB7OUT/TDI/SCL0/SI0/RXD1/ INT6	TMRB用PPG出力 / SBI SCL / UART1Rx
20	PF0/TB7IN/BOOT/SDA0/SO0/TXD1/ INTC	UART1Tx
21	PE2/ENCZ/SCLK0/CTS0/INT7/(SCL0)	SW0/4 / SBI SCL / SIO0 SCLK0
22	VOUT3	未使用
23	VINREG5	未使用
24	VOUT15	未使用
25	PM0/X1	高速クロック
26	DVSSB	GND
27	PM1/X2	高速クロック
28	DVDD5B	+5.0V
29	PE1/ENCB/RXD0/TB4IN	ENCB 入力 / SIO0 Rx
30	PE0/ENCA/TXD0	ENCA 入力 / SIO0Tx

5 開発環境

下記に開発環境の構成を示します:



1. ハードウェア::
   TMPM375FSDMG 評価ボード(非売品)+TMPM375FSDMG

2. 開発ツール:
   - IAR 社:
     1) J-Link: IAR 社製 J-Link-ARM 7.0 / J-Link 6.0
     2) IDE:IAR 社製 Embedded workbench 6.40.1 version
   - KEIL 社:
     1) IDE:KEIL uVision 4.60

補足: PMD サンプル動作環境は次の通りです。

- IAR:
  1) IDE: IAR Embedded workbench 6.50.1 version

6 機能説明

6-1 動作モード選択

TMPM375FSDMG には 3 つの動作モードがあります: NORMAL, IDLE, STOP モード

- **NORMAL モード:**

CPU コアおよび周辺ハードウェアを高速クロックで動作させるモードです。リセット解除後は、NORMALモードになります。

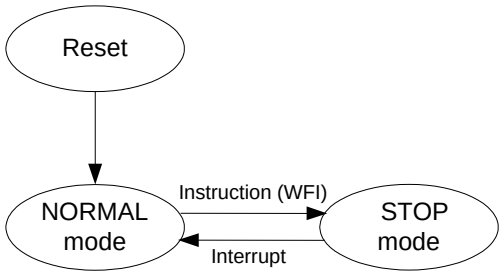
IDLE モードと STOP モードは低消費電力モードです。
低消費電力モードへ移行するには、システ制御レジスタ CGSTBYCR<STBY[2:0]>にて IDLE モードまたは STOP モードを選択し、WFI (Wait For Interrupt)命令を実行します。

注: STOP モードのみサンプルプログラムに含まれます。

- **STOP モード:**

STOPモードでは、すべての内蔵回路が停止します。
STOPモードが解除されると、STOPモードへ移行する直前の動作モードへ復帰し、動作を開始します。
STOPモードでは、CGSTBYCR<DRVE>の設定により端子の状態を設定することができます。

注: STOP モードのサンプルプログラムは CG の例に含まれています。



6-2 ADC

PJ6/AINB9 に接続された VR1(POT1)の値を変更します。測定された電圧値により、評価ボード上の 4 つの LED の点滅間隔を変更します。
電圧値が高いほど点滅間隔が短くなります。

6-3 CG

CPU の動作モードを変更します。NORMAL と STOP の 2 モードのみサポートします。 パワーモードを切り替えるにはキーを押してください。

現在のモード	アクション (Key)	動作	LED 表示
NORMAL	SW1 を押す	NORMAL→ STOP	UART 出力 “NORMAL MODE” →”STOP MODE” 全 LED 消灯
STOP	SW0 を押す	STOP →NORMAL	UART 出力 “STOP MODE” →”NORMAL MODE” 全 LED 点灯

6-4 ENC

この例は、TMPM375FSDMG の ENC ドライバによりホイールマウスの回転を検知します。

デモ手順:

1. ホイールマウスエンコーダーの端子 A(図 1 参照)と TMPM375FSDMG MCU ボードの端子 30(ENCA)を接続します。
2. ホイールマウスエンコーダーの端子 B(図 1 参照)と TMPM375FSDMG MCU ボードの端子 29(ENCB)を接続します。
3. マウスの USB と PC を接続し、ホイールマウスエンコーダーの Com(図 1)を 5V と接続します。
4. プログラムを実行します(LED1 が点滅します)。
5. マウスのホイールを前に回転すると LED1 はより短い間隔で点滅します。点滅間隔が最も短くなると、点滅間隔は初めの間隔に戻ります。
6. マウスのホイールを後ろに回転すると LED1 はより長い間隔で点滅します。点滅間隔が最も長くなると、点滅間隔は初めの間隔に戻ります。

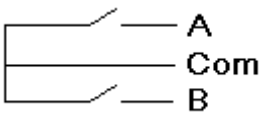


図 1

6-5 FLASH

このアプリケーションは、内蔵フラッシュメモリの消去及び再書き込み操作を行います。
—リセット動作: 書き込みルーチン(フラッシュ API)、転送ルーチンで構成されます。
▪転送ルーチン: 書き込みルーチンを Flash から RAM へ転送します。
▪書き込みルーチン: RAM 上で動作し、block1 にデータを書き込みます。

プログラムシーケンス:
(1) 電源 ON
評価ボードのリセット動作が行われます。
block0 のプログラムが実行を始めます。
SW0 が押されていなければ、LED1 は短い間隔で点滅します。
(2) まず SW0 が押されていると書き込みルーチンが転送ルーチンにより RAM に転送されます。
次に block1 を消去し、その後 block1 にデータを書き込みます。
データが書き込まれた block1 からデータをリードし、コンペアを行います。
同じ時は LED1 がゆっくり点滅し、LED2 は常に点灯します。
異なる時、または書き込みに失敗した時は LED2 が消灯し、LED1 は短い間隔で点滅します。
(3) ステップ(1)を繰り返します。

6-6 GPIO

この例では GPIO を LED に設定し、LED の点灯/消灯を行います。

6-7 VLTD

この例では、VLTDによって検出される電圧状態を実装しています。
電圧が検出電圧より低い場合はハード的にMCUリセットが行われ、ソフト的にPG0にHIGH レベルを出力します。

6-8 OFD

この例では、クロックが OFD に設定された周波数範囲を超えると、OFD リセットが行われ、OFD リセットフラグがセットされます。フラグがセットされると、OFD は無効化され、LED2 は点滅します。

6-9 SBI

6-9-1 SBI スレーブ

本機能は I2C バススレーブモードをサポートします。
評価ボード上の 2 本の I2C バス (SBI0 & SBI) を接続します。
I2C バス の 1 本は I2C スレーブとして機能します。

SBI 割り込みを使用して、I2C バスのリードを行います。
受信アドレスが SBII2CAR で指定されたスレーブアドレスとマッチする、またはゼネラルコールアドレスと等しい場合、SBI は 9 クロック目の SDA ラインを"Low"レベルにし、アクノレッジ信号を出力します。

I2C スレーブ(SBI)は、“TOSHIBA”の文字をマスタ(SBI)から受信し、UART 出力します。

SBI (スレーブ) デモを開始します。

```
K1: SBI SLAVE RECV
```

SW1 を押すと、SBI (スレーブ) 受信バッファから取得した文字列“TOSHIBA”が表示されます。

```
TOSHIBA
MASTER SBI to SLAVE SBI OK
```

6-9-2 SBI マスタ

本機能は I2C バスマスタモードをサポートします。
評価ボード上の 2 本の I2C バス(SBI & I2C)に接続します。
I2C バスの 1 本は I2C マスタとして機能します。

SBI 割り込みを使用して、I2C バスのライトを行います。
I2C のアドレス:任意
I2C マスタ(SBI) はスレーブ (SBI) へ“TOSHIBA”の文字を送信します。

SBI (マスタ) デモを開始します。
SW3 を押すと文字列“TOSHIBA”が SBI (マスタ) から SBI (スレーブ)へ送信されます。

6-10 TMRB

6-10-1 汎用タイマ

このサンプルは、MCU のタイマを使って、汎用タイマを実現します。
タイマ設定は 1ms 周期です。
このタイマを使い 1s(500ms でオン、500ms でオフ)間隔で LED 点滅を行います。

6-10-2 プログラマブル矩形波出力(PPG)

SW0 キーを使い、デューティ可変の PPG(プログラマブル矩形波)の波形を出力します。
デューティは 5 段階(10%, 25%, 50%, 75%, 90%)に変更できます。
電源投入すると PPG モードに入り、PPG 出力を開始します。
キーを押すことでデューティを変更します。
10% → 25% → 50% → 75% → 90% → 10%

6-11 SIO/UART

6-11-1 リターゲット

C 言語の標準入出力ライブラリの標準入力(stdin)、標準出力(stdout)をターゲットのハードウェアに合わせて変更することをリターゲットと呼びます。
サンプルプログラムでは、標準入力と標準出力を、共に UART に設定します。アプリケーションから printf()関数を使ってシリアルポートから出力を行います。

6-11-2 UART FIFO

UART0 から"TMPM3751"というデータを FIFO を使用して UART1 へ送信し、同時に UART1 から"TMPM3752"というデータを FIFO を使用して UART0 へ送信するサンプルプログラムです。
ResetIdx() 関数の前にブレークポイントを設定しておくと、RxBuffer=”TMPM3752” と RxBuffer1="TMPM3751"となった場合にブレークポイントで停止します。

6-11-3 SIO

この例では、SIO を利用した送受信を行います。

SIO のチャネル 0 とチャネル 1 を使用し、これらのチャネル間で同期式データ送信を行います。
(TXD0 と RXD1、TXD1 と RXD0、sclk0 と sclk1 を接続してください)
*補足: この例では、スレーブ側を最初に動作させ、その後、マスタ側を動作させてください。

6-12 WDT

ウォッチドッグタイマは高速クロックが停止する STOP モードでは使えません。リセットが行われ、その後ウォッチドッグタイマはが有効となり、SystemInit()関数で無効になります。

本ドライバサンプルの流れは以下です。

1. 検出時間の設定とカウンターオーバーフロー時の動作としての WDT 割り込み設定を行い、WDT を初期化します。
2. 2 つの WDT 用サンプルがあり、DEMO2 はマクロ定義にて切り替えられます。

DEMO1:

タイマーオーバーフロー時に NMI 割り込みを発生し、WDT をクリアします。

DEMO2:

ウォッチドッグタイマ制御レジスタにクリアコードを書き込み、ウォッチドッグタイマカウンタをクリアします。

6-13 PMD

6-13-1 例: 位相出力

この例は、PMD ドライバを使用して位相出力を行います。

動作シーケンス:

1. プロジェクトを開き、DEMO_U_PHASE、または DEMO_3_PHASE と定義されたサンプルを選択します。プログラムを実行します。
2. EMG (pin 14)を AVDSSB (pin 1)とプルダウン接続します。(LED4 が点灯します)これは EMG 保護状態であることを意味します。
3. EMG (pin 14)を ADVDD5B (pin 6)とプルアップ接続します。(LED1 が消灯します)これは PMD が通常動作であることを意味します。
4. UO1 (pin 13) をオシロスコープに接続し、波形を確認します。
5. XO1 (pin 12) をオシロスコープに接続し、波形を確認します。
6. VO1 (pin 11) をオシロスコープに接続し、波形を確認します。
7. YO1 (pin 10) をオシロスコープに接続し、波形を確認します。
8. WO1 (pin 9) をオシロスコープに接続し、波形を確認します。
9. ZO1 (pin 8) をオシロスコープに接続し、波形を確認します。

DEMO_U_PHASE 定義を選択した場合、位相波形は同じです。

UO1 のデューティは 25%、出力電圧は 5V、周期は 410us です。XO1 は UO1 を反転させた形です。VO1 と WO1 は UO1 と同じで、YO1 と ZO1 は XO1 と同じです。

DEMO_3_PHASE 定義を選択した場合、位相波形は異なります。
UO1 のデューティは 25%、出力電圧は 5V、周期は 410us です。XO1 は UO1 を反転させた形です。
VO1 のデューティは 50%、出力電圧は 5V、周期は 410us です。YO1 は VO1 を反転させた形です。
WO1 のデューティは 75%、出力電圧は 5V、周期は 410us です。ZO1 は WO1 を反転させた形です。

7 ソフトウェア

本ソフトウェアは、TMPM375FSDMG の主要機能を TMPM375FSDMG 評価ボード上でデモ、動作確認するためのサンプルプログラムです。

サンプルプログラムの実装には、最新のドライバーソフト、および IAR EWARM、または KEIL MDK をご使用ください。機能ごとにプロジェクトを作成してください。

ワークスペース構造とプロジェクト名は、以下の通りです:

IAR EWARM:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm375_wdt_int.c
        │   tmpm375_wdt_int.h
        │
        ├─TX03_CMSIS
        │ │ system_TMPM375.c
        │ │ system_TMPM375.h
        │ │ TMPM375.h
        │ │
        │ └─startup
              startup_TMPM375.s
        │
        ├─TX03_Periph_Driver
        │   ├─inc
        │   │   tmpm375_cg.h
        │   │   tmpm375_gpio.h
        │   │   tmpm375_wdt.h
```

```
│   │   tx03_common.h
│   │
│   └─src
│         tmpm375_cg.c
│         tmpm375_gpio.c
│         tmpm375_wdt.c
│
└─TMPM375-EVAL
        led.c
        led.h
        sw.c
        sw.h
        common_uart.c
        common_uart.h
```

KEIL MDK:

```
├─WDT_NMI
    ├─APP
    │   main.c
    │   tmpm375_wdt_int.c
    │
    ├─TX03_CMSIS
    │     system_TMPM375.c
    │     startup_TMPM375.s
    │
    ├─TX03_Periph_Driver
    │     tmpm375_cg.c
    │     tmpm375_gpio.c
    │     tmpm375_wdt.c
    │
    └─TMPM375-EVAL
          led.c
          sw.c
          common_uart.c
```

7-1 ADC

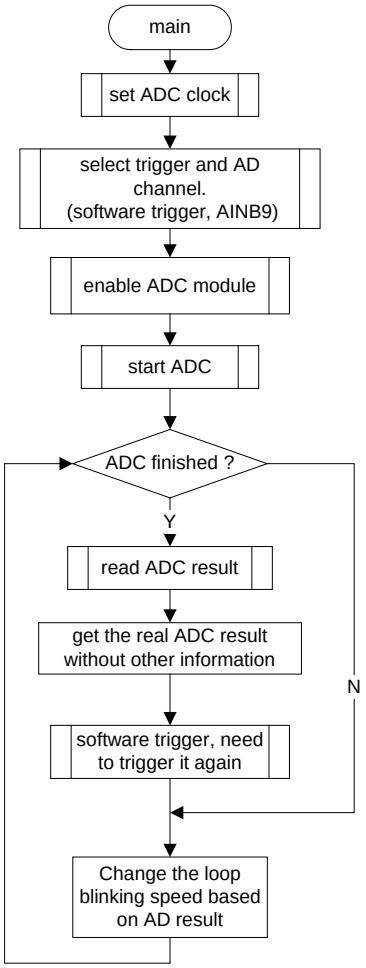
7-1-1 例: ADC データ読み出し

TX03 ペリフェラルドライバ(ADC, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. ADC 設定と初期化
2. ソフトウェアトリガによる AD 変換を開始し、AD 変換結果を読み出します。
3. AD 変換結果により LED 点滅間隔が変化します。

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず ADC のクロック設定を行い、トリガタイプや AD チャネルを選択し、ADC モジュールを有効に設定します。

```
    /* 1. set ADC clock */
    ADC_SetClk(TSB_ADB, ADC_HOLD_FIX, ADC_FC_DIVIDE_LEVEL_8);

    /* 2. select trigger and AD channel, this time we use sofeware trigger, */
    /* the VR1 is connected to ADC unit B channel 9, remember to input with macro
TRG_ENABLE() */
    ADC_SetSWTrg(TSB_ADB, ADC_REG0, TRG_ENABLE(ADC_AIN9));

    /* 3. enable ADC module */
    ADC_Enable(TSB_ADB);
```

AD 変換を開始します:

```
ADC_Start(TSB_ADB, ADC_TRG_SW);
```

AD 変換を行い、その後、ADC モジュールの状態を確認します。AD 変換終了時、他の AD 変換を開始します。

```
while (1U) {
    /* check ADC module state */
    adcState = ADC_GetConvertState(TSB_ADB, ADC_TRG_SW);

    if (adcState == DONE) {
        /* read ADC result when it is finished */
        result = ADC_GetConvertResult(TSB_ADB, ADC_REG0);

        /* get the real ADC result without other information */
        /* "/256" is to limit the range of AD value */
        myResult = 16U - result.Bit.ADResult / 256U;

        /* software trigger, need to trigger it again */
        ADC_Start(TSB_ADB, ADC_TRG_SW);
    }
```

“myResult”の AD 変換結果を確認後、4 つの LED の点滅間隔を変更します。

7-2 CG

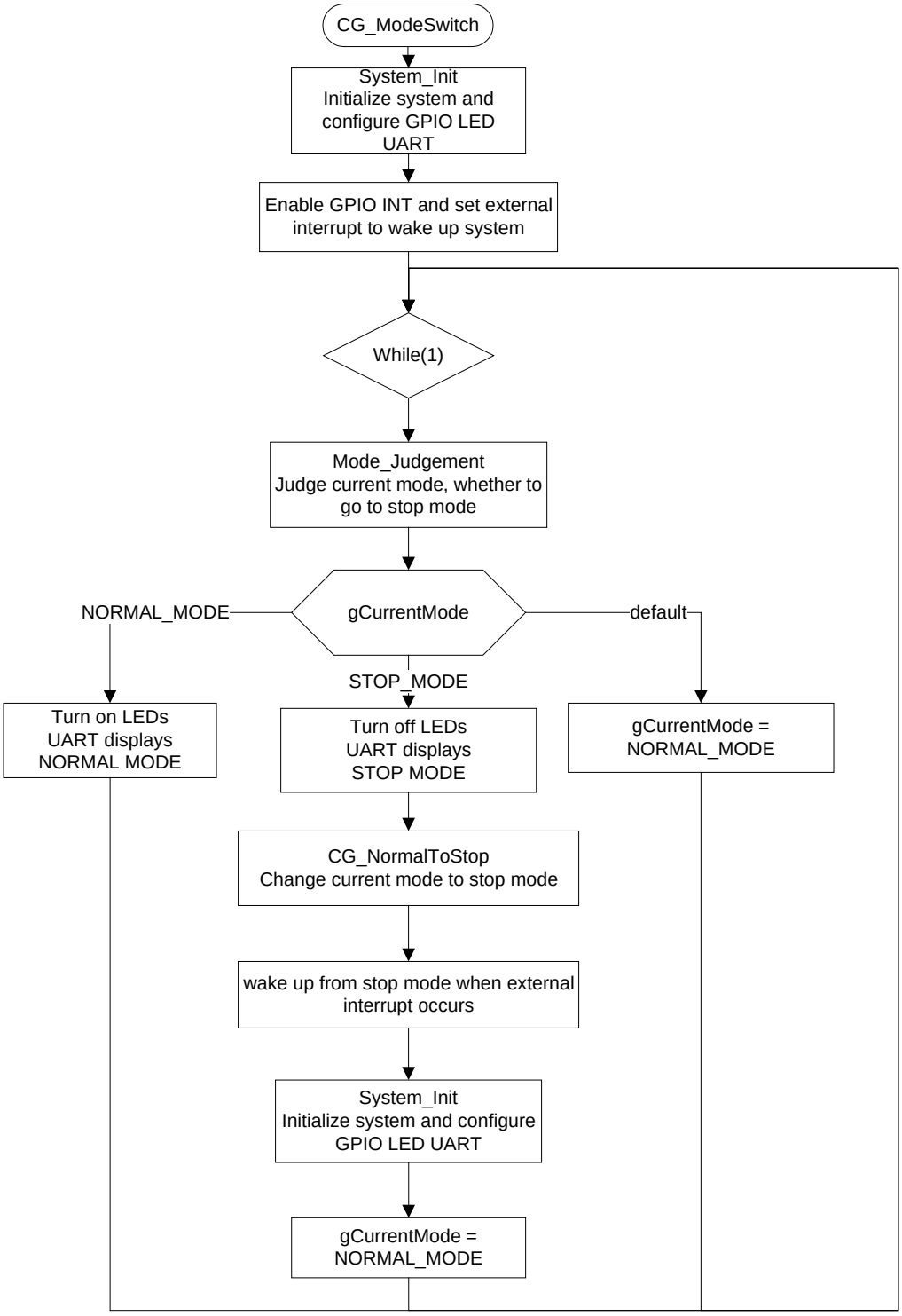
7-2-1 例: パワーモード変更

TX03 ペリフェラル・ドライバ(CG, GPIO,UART)を用いたサンプルプログラムです。

以下の例が含まれます:

1. 基本的な CG 動作の設定
2. normal モードと stop モードの切り替え方法

- **フローチャート**



- **サンプルプログラムのコードと説明**

以下は NORMAL モードと STOP モードの切り替えを行うプログラム例です。

通常の CG の初期化(リセット後)

以下は NORMAL モードで CG の設定を行うプログラム例です。(高速発振器 10MHz の場合)

サンプルプログラムのコード:

```
    /* Set fgear = fc/2 */
    CG_SetFgearLevel(CG_DIVIDE_2);
    /* Set fperiph to fgear */
    CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
    CG_SetPhiT0Level(CG_DIVIDE_64);
    TSB_CG->SYSCR &= 0x10000;
    /* Enable high-speed oscillator */
    CG_SetFosc(CG_FOSC_OSC1,ENABLE);
    /* select external oscillator */
    CG_SetFoscSrc(CG_FOSC_OSC1);
    /* Set low power consumption mode stop */
    CG_SetSTBYMode(CG_STBY_MODE_STOP);
    /* Set pin status in stop mode to "active" */
    CG_SetPinStateInStopMode(ENABLE);
    /* Set up pll and wait for pll to warm up, set fc source to fpll */
    CG_EnableClkMulCircuit();
```

スタンバイ解除のための外部割込みの設定

スタンバイ解除のために INT7 の設定を行います。割り込み保留要求レジスタをクリアし、INT7 を有効にします。

```
    __disable_irq();
    CG_ClearINTReq(CG_INT_SRC_7);
    /* Set external interrupt to wake up system */
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_7,
CG_INT_ACTIVE_STATE_RISING, ENABLE);
    NVIC_ClearPendingIRQ(INT7_IRQn);
    NVIC_EnableIRQ(INT7_IRQn);
    __enable_irq();
```

STOP モード設定

STOP モードに入る設定を行います。ウォームアップ時間を設定し、__WFI()命令を使用して STOP モードに入ります。

```
    /* Set CG module: Normal ->Stop mode */
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC1, CG_WUODR_EXT);
    /* Enter stop mode */
    __WFI();
```

マルチクロック回路の許可

まず PLL を設定します。そしてウォームアップ時間を設定し、ウォームアップ時間が完了するまで待ちます。その後、fPLL を fc ソースとして設定します。

```
    WorkState st = BUSY;
    CG_SetPLL(DISABLE);
    CG_SetFPLLValue(CG_FPLL_10M_MULTIPLY_4);
    retval = CG_SetPLL(ENABLE);
    if (retval == SUCCESS) {
        /* Set warm up time */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC1, CG_WUODR_PLL);
        CG_StartWarmUp();
```

```
        do {
            st = CG_GetWarmUpState();
        } while (st != DONE);

        retval = CG_SetFcSrc(CG_FC_SRC_FPLL);
    } else {
        /*Do nothing */
    }
```

7-3 ENC

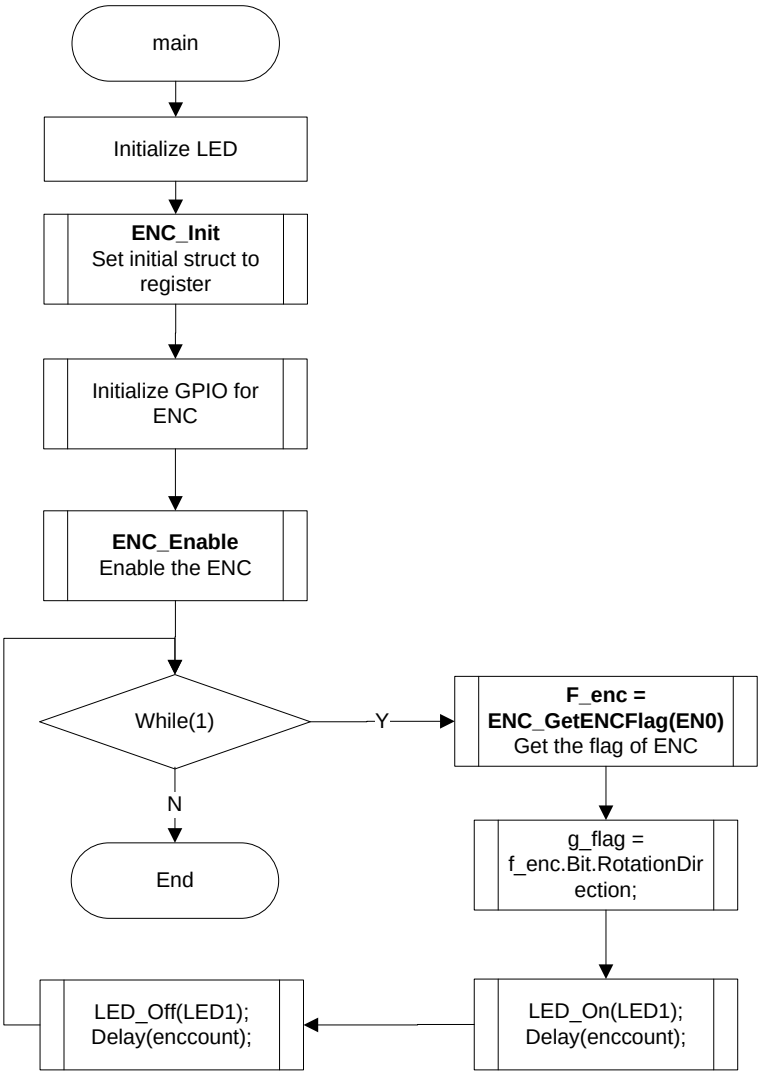
7-3-1 例: 回転検出

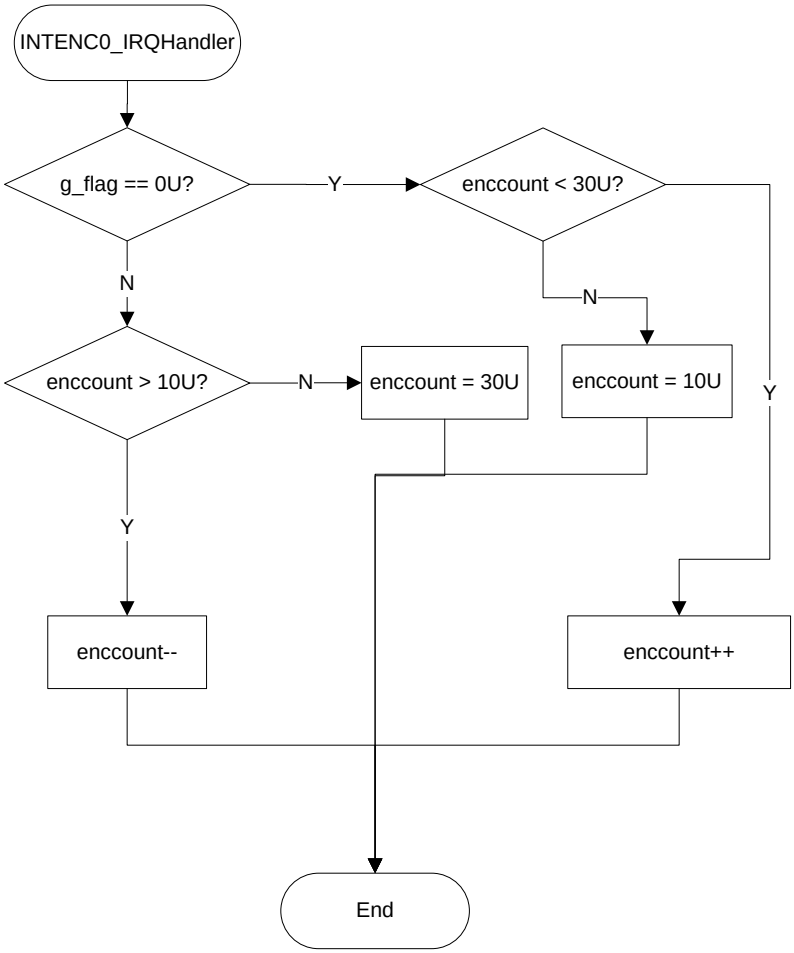
TX03 ペリフェラルドライバ(ENC, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. ENC0 の初期化
2. ホイールマウスの回転検出

- **フローチャート**


main
Initialize LED
ENC_Init
Set initial struct to register
Initialize GPIO for ENC
ENC_Enable
Enable the ENC
While(1)
Y
N
End
F_enc = ENC_GetENCFlag(EN0)
Get the flag of ENC
g_flag = f_enc.Bit.RotationDirection;
LED_On(LED1); Delay(enccount);
LED_Off(LED1); Delay(enccount);



- **サンプルプログラムのコードと説明**

TX03 ペリフェラルドライバ(ENC)を用いてホイールマウスの回転を検知するサンプルプログラムです。

まず、LED を初期化します。

```
/* LED initialization */
LED_Init();
```

ENC 初期化構造体を用意し、ENC0 を初期化します。

```
/* ENC0 initialization */
m_enc.ModeType = ENC_ENCODER_MODE;
m_enc.PhaseType = ENC_TWO_PHASE;
m_enc.CompareStatus = ENC_COMPARE_DISABLE;
m_enc.ZphaseStatus = ENC_ZPHASE_DISABLE;
m_enc.FilterValue = ENC_FILTER_VALUE31;
m_enc.IntEn = ENC_INTERRUPT_ENABLE;
m_enc.PulseDivFactor = ENC_PULSE_DIV1;
```

```
    ENC_Init(EN0,&m_enc);
    ENC_SetCounterReload(EN0,0xFFFU);
```

GPIO を ENC0 に設定します。PE0 と PE1 は ENC0 入力に設定します。

```
    /* GPIO initialization */
    GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_0, ENABLE);/*Set PE0 as
input */
    GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_3, GPIO_BIT_0);/*Set
PE0 as ENCA */
    GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_1, ENABLE);/*Set PE1 as
input */
    GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_3, GPIO_BIT_1);/*Set
PE1 as ENCB */
```

ENC0 と ENC0 割り込みを許可します。

```
    /* Enable ENC0 */
    ENC_Enable(EN0);
    /* Enable ENC0 interrupt */
    NVIC_EnableIRQ(INTENC0_IRQn);
```

ホイールマウスの回転検知のため ENC0 の回転方向検出状態を取得します。回転数に応じて LED1 を点滅します。

```
    /* LED1 blink speed based on the rolling of mouse wheel */
    while(1){
        f_enc = ENC_GetENCFlag(EN0);
        g_flag = f_enc.Bit.RotationDirection;
        LED_On(LED1);
        Delay2(enccount);
        LED_Off(LED1);
        Delay2(enccount);
    }
```

ENC カウントは、回転した数で、割り込みルーチンで変化します。
ホイールマウスが前方向に回転すると、ENC カウンタが 10 まで減ると ENC カウンタは 30 に戻り、ENC カウンタが 30 まで増えると ENC カウンタが 10 に戻ります。

```
    if(g_flag == 1U){
        if(enccount < 30U){
            enccount++;
        } else {
            enccount = 10U;
        }
     } else {
        if(enccount > 10U){
            enccount--;
        } else {
            enccount = 30;
        }
    }
```

7-4 FLASH

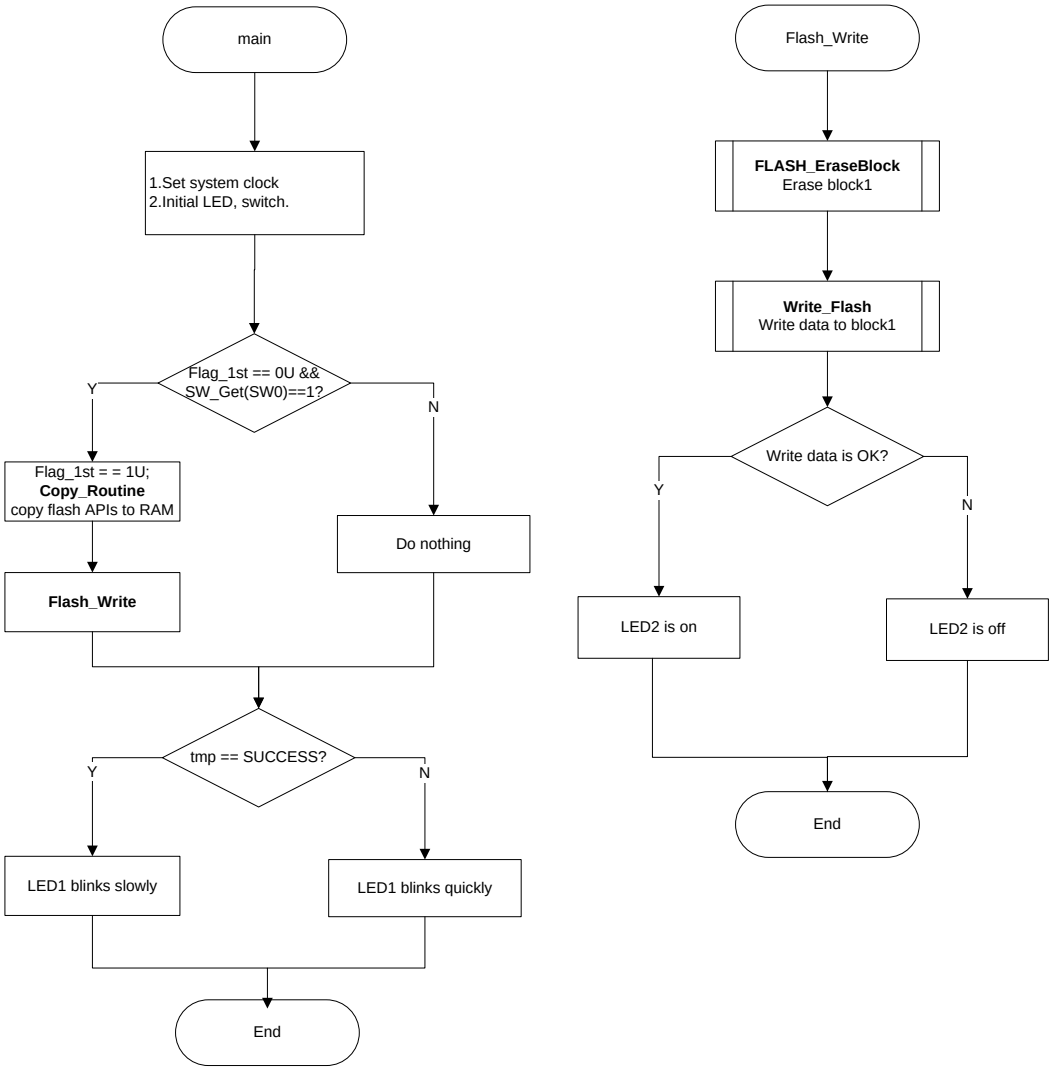
7-4-1 例: Flash ROM への書き込み

TX03 ペリフェラルドライバ(FLASH, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. FLASH メモリのオンボードプログラミング (書き込み/消去)

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず LED とスイッチを初期化します。

```
    LED_Init();
    SW_Init();
```

リセット時に SW0 が OFF の場合、LED1 は短い間隔で点滅します。SW0 が ON の場合、動作モードはユーザブートモードに移行します。フラッシュメモリ上で実行しながらフラッシュメモリ自身を消去/プログラムできないため、プログラムはフラッシュ操作ルーチンをアドレス“FLASH_API_ROM”からアドレス“FLASH_API_RAM”へ ROM->RAM 転送します。

```
    Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
```

Flash 動作用 API を RAM にコピー後、プログラムは Flash_Write()関数を実行します。書き込み動作が OK の場合、 LED1 は長い間隔で点滅します。

```
while (1U) {
        if(Flag_1st == 0U && SW_Get(SW0)) {     /* if SW0 is turned on 1st time*/
            Flag_1st = 1U;
            Copy_Routine(FLASH_API_RAM,                    FLASH_API_ROM,
SIZE_FLASH_API);
            tmp = Flash_Write();
        } else {
            /* Do nothing */
        }
        if(tmp== SUCCESS)
        {
            LED_On(LED1);
            delay(SUCCESS_DELAY);
            LED_Off(LED1);
            delay(SUCCESS_DELAY);
        } else {
            LED_On(LED1);
            delay(ERROR_DELAY);
            LED_Off(LED1);
            delay(ERROR_DELAY);
        }
    }
```

Flash_Write()関数では、まず消去するために FC_EraseBlock()をコールし、そして Flash メモリに書き込むために FC_WritePage()をコールします。データは block1 に書き込みます。
block1 からリードしたデータが書き込んだデータと同じ場合、LED2 は点灯し、異なる場合、LED2 は消灯します。

```
    int i = 0U;
    uint32_t pagedata[FC_PAGE_SIZE];
    Result tmp = ERROR;
    uint32_t buffer[FC_PAGE_SIZE] = {0U};
    if (FC_SUCCESS == FC_EraseBlock(FC_WRITE_ADDR)) {                    /*
erase this block which will be written */
        for (i = 0U; i < FC_PAGE_SIZE; i++) {
            pagedata[i] = 0x12345678U;
        }
        if  (FC_WritePage(FC_WRITE_ADDR,  pagedata)  ==  FC_SUCCESS)
/* write data to block which was erased */
```

```
        {
            Copy_Routine(buffer,                    (uint32_t*)FC_WRITE_ADDR,
FLASH_PAGE_SIZE);
            tmp = SUCCESS;
            LED_On(LED2);
            for (i = 0U; i < FC_PAGE_SIZE; i++) {
                if(buffer[i] != pagedata[i]) {
                    tmp = ERROR;
                    LED_Off(LED2);
                    break;
                }else {
                   /* Do nothing */
                }
            }
        }else {
            /* Do nothing */
        }
    } else {
       /* Do nothing */
    }
    return tmp;
```

Flash メモリ動作関数 FC_EraseBlock() は自動的に指定されたブロックを消去します。このブロックは最初に引数 “BlockAddr” で指定します。まず、この関数で引数 “BlockAddr” を確認します。次に、Flash ドライバー FC_GetBlockProtectState() を使用し、指定されたブロックがプロテクトされているか確認します。

```
    if (ENABLE == FC_GetBlockProtectState(BlockNum)) {
        retval = FC_ERROR_PROTECTED;
    }
```

ブロックにプロテクトがかかっている場合、“FC_ERROR_PROTECTED” を返します。それ以外の場合は、自動的ブロック削除命令を送り、ブロックを削除します。

```
        *addr1 = (uint32_t) 0x000000AA; /* bus cycle 1 */
        *addr2 = (uint32_t) 0x00000055; /* bus cycle 2 */
        *addr1 = (uint32_t) 0x00000080; /* bus cycle 3 */
        *addr1 = (uint32_t) 0x000000AA; /* bus cycle 4 */
        *addr2 = (uint32_t) 0x00000055; /* bus cycle 5 */
        *BA = (uint32_t) 0x00000030;       /* bus cycle 6 */
```

次に、削除が終了すると、Flash ドライバ FC_GetBusyState() を用いモニターリングを行います。同時に、タイムアウトカウンタを使用し、動作が指定時間を越えていないか確認します。

```
        while (BUSY == FC_GetBusyState()) {        /* check if FLASH is busy with
overtime counter */
            if (!(counter--)) { /* check overtime */
                retval = FC_ERROR_OVER_TIME;
                break;
            } else {
                /* Do nothing */
            }
        }
```

関数 Write_Flash() は FC_WritePage () を呼び出し、自動的に 1 ページにデータを書き込みます。この動作は、自動ページプログラム命令を除き、基本的に FC_EraseBlock

() と同じです。

```
*addr1 = (uint32_t) 0x000000AA; /* bus cycle 1 */
*addr2 = (uint32_t) 0x00000055; /* bus cycle 2 */
*addr1 = (uint32_t) 0x000000A0; /* bus cycle 3 */
for (i = 0U; i < FC_PAGE_SIZE; i++) {     /* bus cycle 4~67 */
      *addr3 = *source;
      source++;
}
```

7-5 GPIO

この機能は LED とスイッチを使うために GPIO の設定を行います。また、LED オンや LED オフを制御するためのスイッチ状態を読み出します。

7-5-1 例: GPIO データ読み出し

TX03 ペリフェラルドライバ GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. GPIO の初期化
2. GPIO へのデータ書き込み
3. GPIO からのデータ読み出し

- **サンプルプログラムのコードと説明**

まず GPIO_SetOutput(GPIO_Port GPIO_x, uint8_t Bit_x)を使用して GPIO を LED に設定します。そして GPIO_SetInput(GPIO_Port GPIO_x, uint8_t Bit_x)を使用して GPIO をキーに設定します。以下に例を示します。

```
GPIO_SetOutput(GPIO_PG, GPIO_BIT_0);
GPIO_SetOutput(GPIO_PG, GPIO_BIT_1);
GPIO_SetOutput(GPIO_PG, GPIO_BIT_2);
GPIO_SetOutput(GPIO_PG, GPIO_BIT_3);
GPIO_SetInput(GPIO_PE, GPIO_BIT_2);
GPIO_SetInput(GPIO_PJ, GPIO_BIT_7);
GPIO_SetInput(GPIO_PK, GPIO_BIT_0);
GPIO_SetInput(GPIO_PK, GPIO_BIT_1);
```

for(; ;)ループの中で、LED デモを実行します: スイッチ状態をリードし、LED オンや LED オフを制御します。

GPIO_ReadDataBit(GPIO_Port GPIO_x, uint8_t Bit_x)を使用してスイッチ状態をリードします。.

```
if (GPIO_ReadDataBit(GPIO_PE, GPIO_BIT_2) == 1U) {
            tmp = 1U;
} else {
            /*Do nothing */
}
```

GPIO_WriteDataBit(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t BitValue)を使用して

LED をオンします。

```
uint32_t tmp;
tmp = GPIO_ReadData(GPIO_PG);
tmp |= led;
GPIO_WriteData(GPIO_ PG, tmp);
```

GPIO_WriteDataBit(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t BitValue)を使って LED をオフします。

```
uint32_t tmp;
  tmp = GPIO_ReadData(GPIO_PG);
  tmp &= (uint8_t)(~led);
  GPIO_WriteData(GPIO_PG, tmp);
```

7-6 VLTD

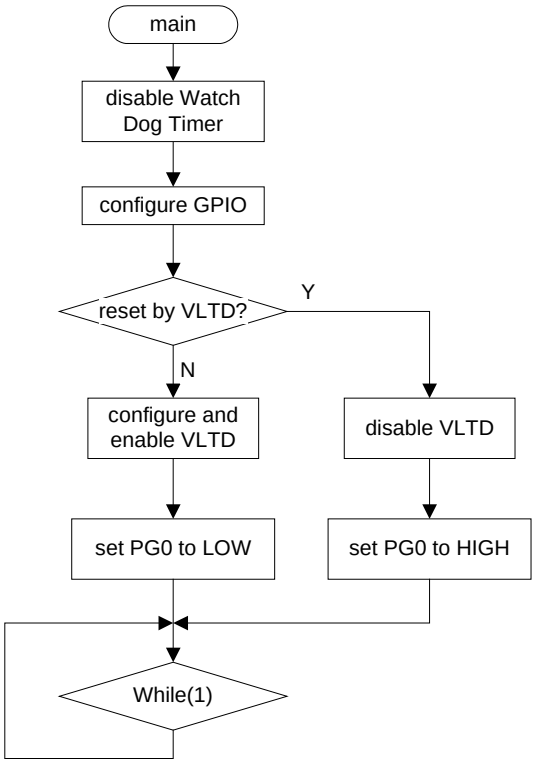
7-6-1 例: VLTD リセット

TX03 ペリフェラルドライバ(VLTD, CG, GPIO, WDT)を用いたサンプルプログラムです。

以下の例が含まれます:

1. VLTD 設定

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、ウォッチドッグタイマを禁止します。その後、PG0 を出力ポートに設定します。

```
WDT_Disable();
GPIO_SetOutput(GPIO_PG, GPIO_BIT_0);
```

その後、VLTD リセットフラグを読み出し、もし VLTD リセットされていなければ、VLTD の設定と有効化を行い、PG0 を LOW レベルにします。

```
if (CG_GetResetFlag().Bit.VLTDReset == 0U) {
        VLTD_SetVoltage(VLTD_DETECT_VOLTAGE_46);
        VLTD_Enable();
        GPIO_WriteData(GPIO_PG, 0x00U);
```

VLTD リセットされている場合は、VLTD を無効にし、PG0 を HIGH レベルにします。

```
} else {
    VLTD_Disable();
    GPIO_WriteData(GPIO_PG, 0x01U);
}
```

7-7 OFD

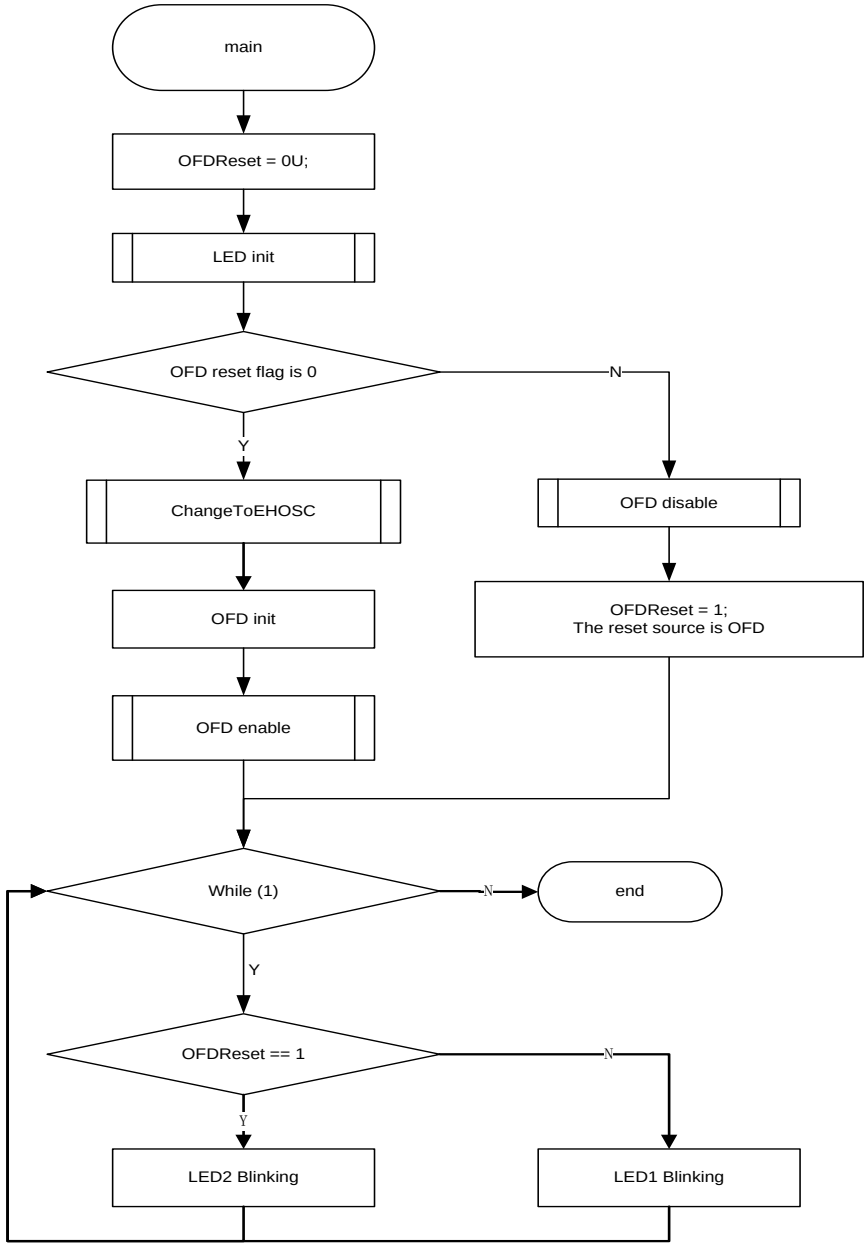
7-7-1 例: OFD リセット

TX03 ペリフェラルドライバ(OFD, CG, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. OFD の初期化(検知周波数範囲を OFD に設定します)
2. OFD リセットフラグの確認
3. OFD の有効／無効

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず、LED を初期化します。

```
    LED_Init ();
```

OFD のリセット flag を確認します。

```
    resetFlag = CG_GetResetFlag();
    if (resetFlag.Bit.OFDReset == 0U) {
      /* reset source isn't OFD */
    }
```

この OFD フラグが'0'の場合(通常リセットの場合、電源投入にて MCU がリセットされると、この OFD フラグは'0'になります)、プログラムにて外部発振器を選択、検知周波数範囲を OFD に設定し、OFD を初期化します。

下記コードは、外部発振器を使うための CG の設定を行います。

```
void ChangeToEHOSC(void)
{
    /* Use the external oscillation */
    TSB_CG->SYSCR = 0x00010000UL;
    TSB_CG->OSCCR &= 0x000FFFFFUL;
    TSB_CG->OSCCR |= 0xC3500000UL;              /*Set the warm up time WUODR[13:2] */
    TSB_CG_OSCCR_HOSCON = 1U;      /*Set port as X1/X2 for external oscillator */
    TSB_CG_OSCCR_XEN1 = 1U;        /*Enable external oscillator */
    TSB_CG_OSCCR_WUPSEL2 = 1U;
    TSB_CG_OSCCR_WUPSEL1 = 0U;   /*Select warm-up clock */
    TSB_CG_OSCCR_WUEON = 1U;       /*Start warm up */
    TSB_CG->PLLSEL = (0x0000680FUL<<1U);
    while (TSB_CG_OSCCR_WUEF) {
    }                                   /* Warm-up */
    TSB_CG_OSCCR_OSCSEL = 1U;     /* Use the external oscillation */
    TSB_CG_OSCCR_XEN2 = 1U;
    TSB_CG->OSCCR &= 0x000FFFFFUL;
    TSB_CG->OSCCR |= 0x03E80000UL;
    TSB_CG_OSCCR_PLLON = 1U;       /* PLL enabled */
    TSB_CG->STBYCR = 0x00000103UL;
    TSB_CG_OSCCR_WUEON = 1U;
    while (TSB_CG_OSCCR_WUEF) {
    }                                   /* Warm-up */

    TSB_CG->PLLSEL = ((0x0000680FUL<<1U)|1U);
}
```

OFD の設定を行うため、まずレジスタ書き込み許可コードを設定します。

```
    OFD_SetRegWriteMode(ENABLE);
```

PLL OFF 時の検知周波数の設定を行います。

```
    OFD_SetDetectionFrequency(OFD_PLL_OFF, 0X25, 0X1D);
```

PLL ON 時の検知周波数の設定を行います。

```
    OFD_SetDetectionFrequency(OFD_PLL_ON,OFD_HIGHER_COUNT, OFD_LOWER_COUNT);
```

DEMO2 を定義すると異常な周波数範囲が設定されます。

```
    OFD_SetDetectionFrequency(OFD_PLL_ON,OFD_HIGHER_COUNT_ABNO
RMAL, OFD_LOWER_COUNT_ABNORMAL);
```

初期化し、その後、OFD を有効にします。

```
    OFD_Enable();
```

上記の設定を行い、その後 OFD は外部クロックを検知します。このクロックが検知周波数範囲を超える(OSC が乱れる、DEMO2 が定義されている)と OFD は周波数検知リセットを発生します。これにより OFD のリセットフラグが設定されます。
このフラグを検知すると、デフォルト内部発振器で MCU が実行を始め、OFD は無効となり、OFD リセットフラグに'1'がセットされます。

```
    OFD_Disable();
    OFDReset = 1U;
```

LED1 と LED2 はシステムクロックの状態を表します。

LED の状態	意味
LED1 点滅	MCU は通常に動作しています。 外部発振器は正常に発振しています。 OFD 機能が有効となり、異常状態の発振を検知できる状態です。
LED2 点滅	外部発振器が異常のため、OFD リセットが発生し、MCU はデフォルト設定の内部発振器にて動作します。OFD 機能は無効の状態です。

```
    while (1U) {
        if (OFDReset == 1U) {
            LED_On(LED2);
            Delay();
            LED_Off(LED2);
            Delay();
        } else {
            LED_On(LED1);
            Delay();
            LED_Off(LED1);
            Delay();
        }
    }
```

7-8 SBI

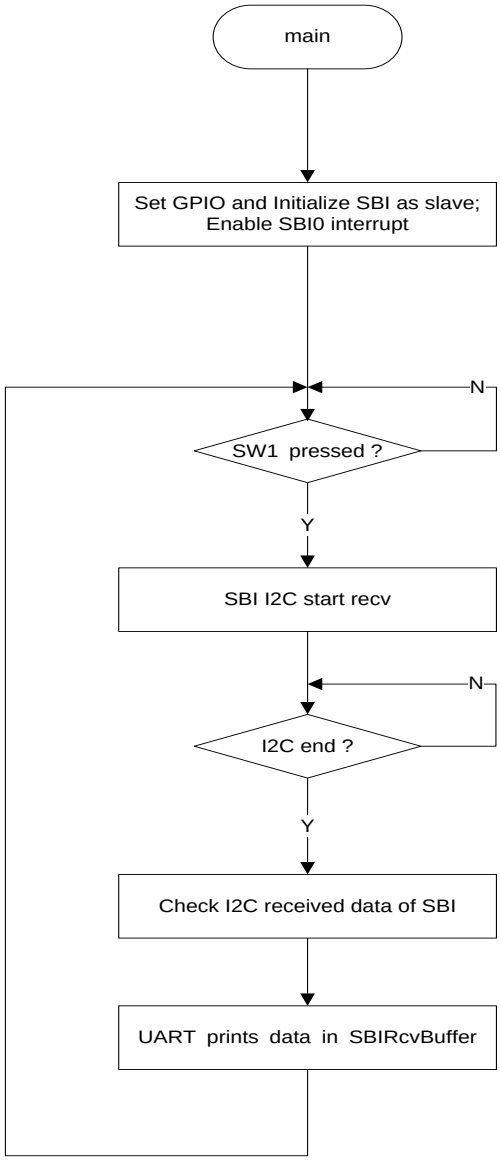
7-8-1 例: SBI スレーブ

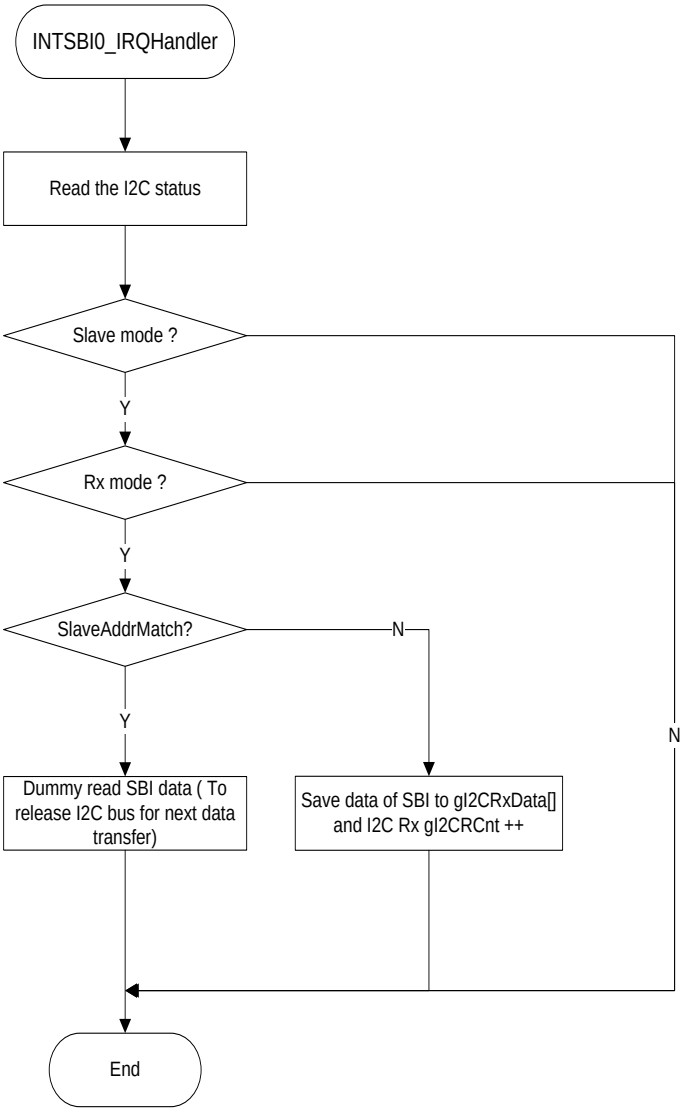
TX03 ペリフェラルドライバ (SBI, GPIO) のプログラムサンプルです。

以下の例が含まれます:

1. SBI 設定、I2C 初期化
2. SBI スレーブによるデータ受信

- **フローチャート**





- **サンプルプログラムのコードと説明**

まず、GPIO を I2C モードに設定します。

```
GPIO_EnableFuncReg(GPIO_PB, GPIO_FUNC_REG_5, GPIO_BIT_5);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_5, GPIO_BIT_2);
GPIO_SetOutputEnableReg(GPIO_PB, GPIO_BIT_5, ENABLE);
GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_2, ENABLE);
GPIO_SetInputEnableReg(GPIO_PB, GPIO_BIT_5, ENABLE);
GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_2, ENABLE);
```

```
    GPIO_SetOpenDrain(GPIO_PB, GPIO_BIT_5, ENABLE);
    GPIO_SetOpenDrain(GPIO_PE, GPIO_BIT_2, ENABLE);
    /* Pull up for SDA&SCL */
    GPIO_SetPullUp(GPIO_PB, GPIO_BIT_5, ENABLE);
    GPIO_SetPullUp(GPIO_PE, GPIO_BIT_2, ENABLE);
```

スレーブ SBI チャネルの初期設定、および INTSBI0 を許可します。

```
    myI2C.I2CSelfAddr = SLAVE_ADDR;
    myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
    myI2C.I2CACKState = ENABLE;
    myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
    SBI_Enable(TSB_SBI);
    SBI_SWReset(TSB_SBI);
    SBI_InitI2C(TSB_SBI, &myI2C);
    NVIC_EnableIRQ(INTSBI0_IRQn);
```

上記設定を行い、その後、I2C 受信を開始します。
I2C 受信バッファのクリア、SBI 受信バッファとバッファ長の初期設定を行います。

```
        /* Initialize TRx buffer and Tx length */
    case MODE_SBI_I2C_INITIAL:
        for (gICnt = 0U; gICnt < 8U; gICnt++) {
            gI2CRxData[gICnt] = 0U;
        }
        gSBIMode = MODE_SBI_I2C_RCV;
        break;
```

INTSBI0 でデータ受信を行います。

INTSBI0 関数内で、I2C バス状態を取得し、I2C スレーブ受信プロセスを決定します。
SBI_GetReceiveData()は SBI バッファの受信データをリードするために使い、I2C ストップコンディションはマスタにより制御されます。

```
void INTSBI0_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI_TypeDef *SBIy;
    SBI_I2CState sbi0_sr;

    SBIy = TSB_SBI;
    sbi0_sr = SBI_GetI2CState(SBIy);

    if (!sbi0_sr.Bit.MasterSlave) {        /* Slave mode */
        if (!sbi0_sr.Bit.TRx) { /* Rx Mode */
            if (sbi0_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRCnt = 0U;
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRxData[gI2CRCnt] = tmp;
                gI2CRCnt++;
            }
        } else {                        /* Tx Mode */
            /* Do nothing */
        }
    } else {                            /* Master mode */
```

```
            /* Do nothing */
        }
}
```

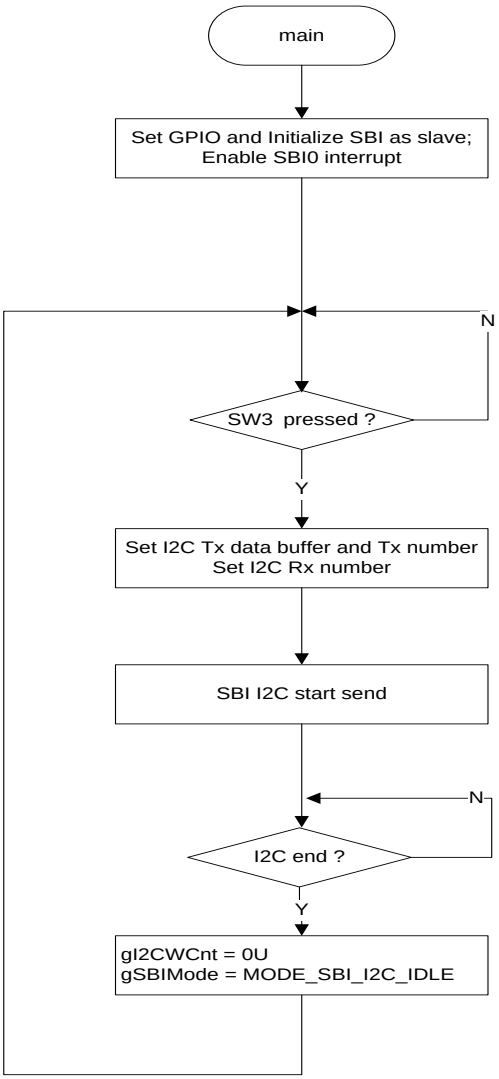
7-8-2　例: SBI マスタ

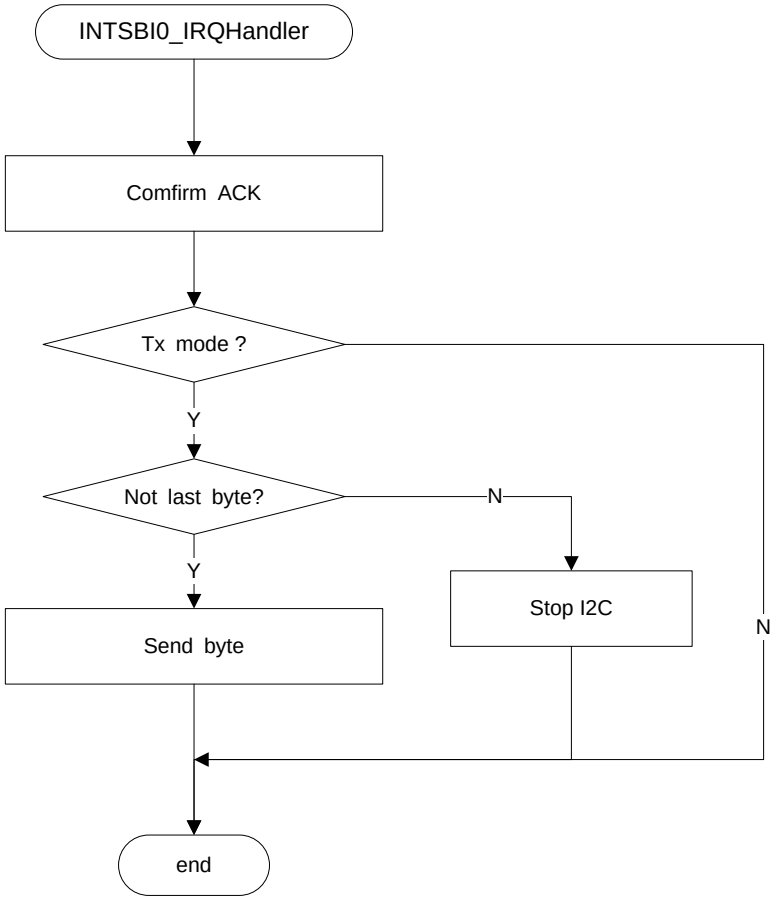
TX03 ペリフェラルドライバ (SBI, GPIO) のプログラムサンプルです。

以下の例が含まれます:

1　SBI 設定、I2C 初期化
2　SBI マスタによるデータ送信

- **フローチャート**





- **サンプルプログラムのコードと説明**

まず、GPIO を I2C モードに設定します。

```
GPIO_EnableFuncReg(GPIO_PB, GPIO_FUNC_REG_5, GPIO_BIT_5);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_5, GPIO_BIT_2);
GPIO_SetOutputEnableReg(GPIO_PB, GPIO_BIT_5, ENABLE);
GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_2, ENABLE);
GPIO_SetInputEnableReg(GPIO_PB, GPIO_BIT_5, ENABLE);
GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_2, ENABLE);
GPIO_SetOpenDrain(GPIO_PB, GPIO_BIT_5, ENABLE);
GPIO_SetOpenDrain(GPIO_PE, GPIO_BIT_2, ENABLE);
/* Pull up for SDA&SCL */
GPIO_SetPullUp(GPIO_PB, GPIO_BIT_5, ENABLE);
GPIO_SetPullUp(GPIO_PE, GPIO_BIT_2, ENABLE);
```

マスタ SBI チャネルの初期設定、および INTSBI0 を許可します。

```
myI2C.I2CSelfAddr = SELF_ADDR;
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2C.I2CACKState = ENABLE;
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
SBI_Enable(TSB_SBI);
```

```
    SBI_SWReset(TSB_SBI);
    SBI_InitI2C(TSB_SBI, &myI2C);
    NVIC_EnableIRQ(INTSBI0_IRQn);
```

上記設定を行い、その後、I2C 送信を開始します。
I2C 送信バッファに"TOSHIBA"を設定し、SBI 送信バッファとバッファ長の初期設定を行います。

```
        /* Initialize TRx buffer and Tx length */
    case MODE_SBI_I2C_INITIAL:
        gI2CTxDataLen = 7U;
        gI2CTxData[0] = gI2CTxDataLen;
        gI2CTxData[1] = 'T';
        gI2CTxData[2] = 'O';
        gI2CTxData[3] = 'S';
        gI2CTxData[4] = 'H';
        gI2CTxData[5] = 'I';
        gI2CTxData[6] = 'B';
        gI2CTxData[7] = 'A';

        gI2CWCnt = 0U;
 gSBIMode = MODE_SBI_I2C_START;
        break;
```

I2C バスが空いているかどうか、“SLAVE_ADDR”データを SBI_SetSendData()に設定します。そして、送信方向を “SBI_I2C_SEND”から SBI データバッファへ設定します。その後、SBI_GenerateI2CStart(TSB_SBI)を使用して、I2C 動作を開始します。

```
        /* Check I2C bus state and start TRx */
    case MODE_SBI_I2C_START:
        i2c_state = SBI_GetI2CState(TSB_SBI);
        if (!i2c_state.Bit.BusState) {
            SBI_SetSendData(TSB_SBI, SLAVE_ADDR | SBI_I2C_SEND);
            SBI_GenerateI2CStart(TSB_SBI);
            gSBIMode = MODE_SBI_I2C_TRX;
        } else {
            /* Do nothing */
        }
        break;
```

INTSBI0 でデータ送信を行います。

INTSBI0 関数内で、I2C バス状態を取得し、その値で I2C マスタ送信プロセスを決定します。I2C マスタ送信中は、SBI_SetSendData()で次のデータを送信し、I2C プロセス終了時には、SBI_GenerateI2CStop()で I2C を停止します。

```
void INTSBI0_IRQHandler(void)
{
    TSB_SBI_TypeDef *SBIx;
    SBI_I2CState sbi_sr;

    SBIx = TSB_SBI;
    sbi_sr = SBI_GetI2CState(SBIx);

    if (sbi_sr.Bit.MasterSlave) {           /* Master mode */
        if (sbi_sr.Bit.TRx) {     /* Tx mode */
            if (sbi_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further
data. */
                SBI_GenerateI2CStop(SBIx);
```

```
                } else {                /* LRB=0: the receiver requires further data. */
                    if (gI2CWCnt <= gI2CTxDataLen) {
                        SBI_SetSendData(SBIx, gI2CTxData[gI2CWCnt]);            /*
Send next data */
                        gI2CWCnt++;
                    } else {            /* I2C data send finished. */
                        SBI_GenerateI2CStop(SBIx);    /* Stop I2C */
                    }
                }
            } else {                    /* Rx Mode */
                /* Do nothing */
            }
        } else {                        /* Slave mode */
            /* Do nothing */
        }
}
```

7-9　16 ビットタイマ(TMRB)

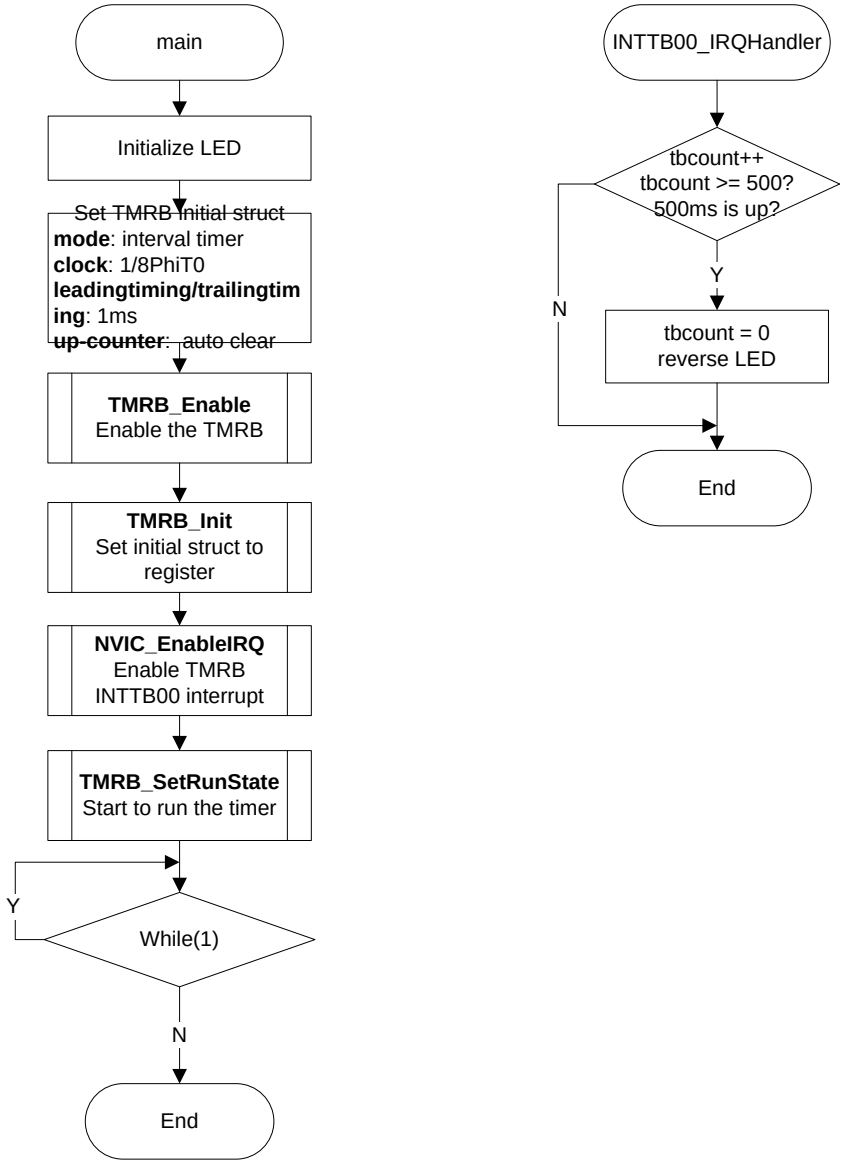
7-9-1　例: 汎用タイマ

TX03 ペリフェラルドライバ(TMRB, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. TMRB0 の初期化
2. 1ms の汎用タイマ

- **フローチャート**



- **サンプルプログラムのコードと説明**

最初に LED チャネルを初期化し、評価ボード上の LED をオンします。

```
LED_Init();                                    /* LED initialize */
LED_On(LED_ALL);                               /* Turn on LED_ALL */
```

TMRB 初期化構造体を用意し、TMRB モード、クロック、アップカウンタクリアメソッド、サイクル、デューティを設定します。本デモでは、サイクルとデューティーを 1ms に設定します。サイクルとデューティは Tmrb_Calculator()関数により算出します。

```
TMRB_InitTypeDef m_tmrb;

m_tmrb.Mode = TMRB_INTERVAL_TIMER;       /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;          /* 1/8PhiT0 */
/* periodic time is 1ms(require 1000us) */
m_tmrb.TrailingTiming = Tmrb_Calculator(1000U, m_tmrb.ClkDiv);
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;      /* up-counter auto clear */
/* periodic time is 1ms(require 1000us) */
m_tmrb.LeadingTiming = Tmrb_Calculator(1000U, m_tmrb.ClkDiv);
```

TMRB モジュールを有効にした後、初期化構造体を指定のレジスタに設定します。INTTB0 割り込み（1ms ごとにトリガ）を有効にします。最後に TMRB を動作させます。

```
TMRB_Enable(TSB_TB0);                    /* enable the TMRB0 */
TMRB_Init(TSB_TB0, &m_tmrb);             /* initial the TMRB0 */
NVIC_EnableIRQ(INTTB0_IRQn);             /* enable INTTB0 interrupt */
TMRB_SetRunState(TSB_TB0, TMRB_RUN);     /* run TMRB0*/
```

メインルーチンは、"While(1) "に入り、割り込みの発生を待ちます。割り込みルーチンでは、カウンタでカウントを行い、500ms をカウントすると、 LED を反転させカウントを再開します。

```
tbcount++;
if (tbcount >= 500U) {                   /* 500ms is up */
    tbcount = 0U;
    /* reverse LED output */
    ledon = (ledon == 0U) ? 1U : 0U;
    if (0U == ledon) {
        LED_Off(LED_ALL);
    } else {
        LED_On(LED_ALL);
    }
} else {
    /* Do nothing */
}
```

Tmrb_Calculator()関数:

```
uint16_t Tmrb_Calculator(uint16_t Tmrb_Require_us, uint32_t ClkDiv)
{
    uint32_t T0 = 0U;
    const uint16_t Div[8U] = {1U, 2U, 8U, 32U, 64U, 128U, 256U, 512U};

    SystemCoreClockUpdate();

    T0 = SystemCoreClock / (1U << ((TSB_CG->SYSCR >> 8U) & 7U));
    T0 = T0/((Div[ClkDiv])*1000000U);

    return(Tmrb_Require_us * T0);
}
```

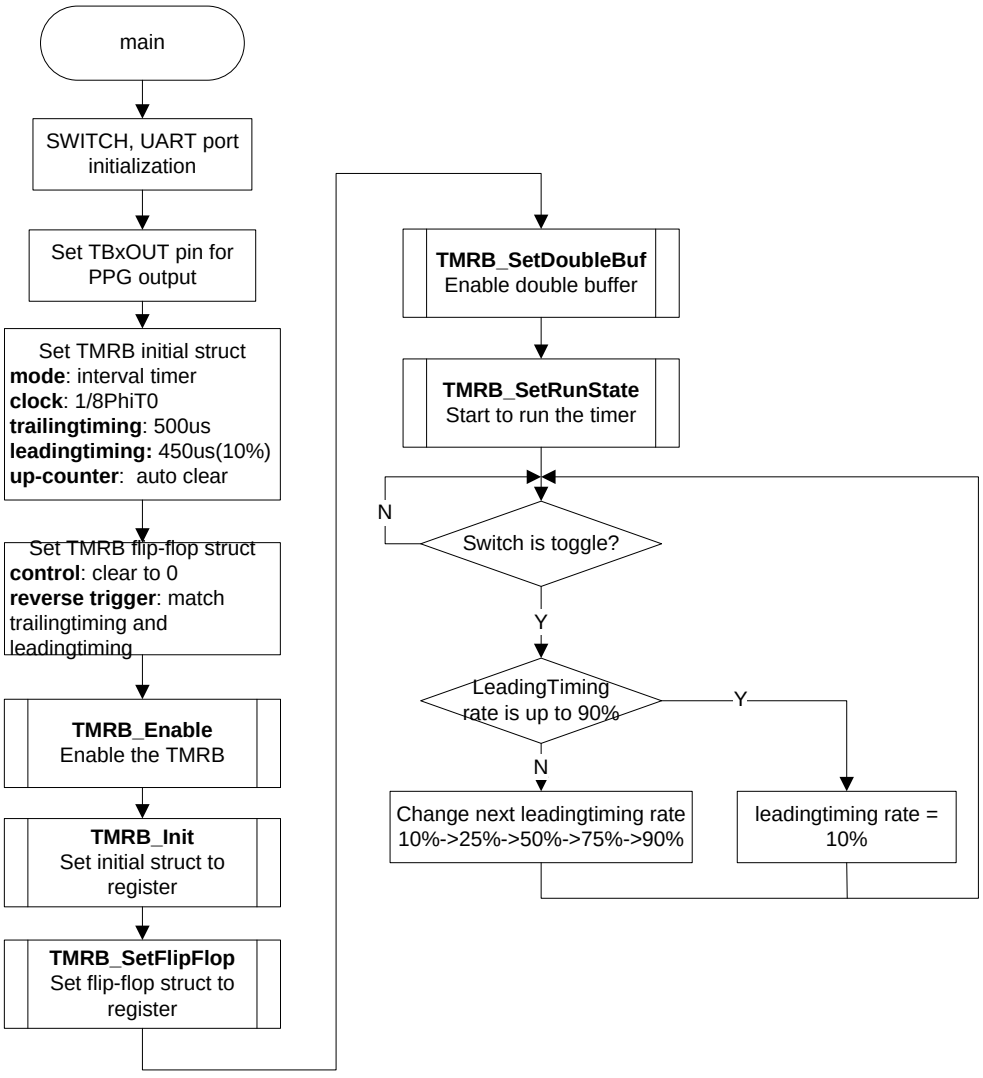
7-9-2 例: プログラマブル矩形波出力(PPG)

TX03 ペリフェラルドライバ(TMRB, GPIO, UART)を用いたサンプルプログラムです。

以下の例が含まれます:

1. TMRB7 の初期化
2. PPG 機能の設定と開始
3. PPG デューティの調整

- **フローチャート**



- **サンプルプログラムのコードと説明**

  最初に tgtLeadingTiming, LeadingTimingus, LeadingTiming の各配列を初期化します。
  次に SWITCH と UART チャネルを初期化し、PB6 を PPG 出力にするため TB7OUT に

設定します。

```
    TMRB_InitTypeDef m_tmrb;
    TMRB_FFOutputTypeDef PPGFFInital;
    uint8_t keyvalue;
    uint32_t i = 0U;
    uint32_t tgtLeadingTiming[5U] = { 10U, 25U, 50U, 75U, 90U }; /* leading timing:
10%, 25%, 50%, 75%, 90% */
    uint32_t LeadingTimingus[5U] = {0U};
    uint32_t LeadingTiming[5U] = {0U};

    /* LeadingTimingus: 50, 125, 250, 375, 450 */
    for (i=0U;i<=4U;i++) {
        LeadingTimingus[i] = tgtLeadingTiming[i] * 5U;
    }

    /* UART & switch initialization */
    hardware_init(UART_RETARGET);
    SW_Init();

    /* Set PB6 as TB7OUT for PPG output */
    GPIO_SetOutput(GPIO_PB, GPIO_BIT_6);
    GPIO_EnableFuncReg(GPIO_PB, GPIO_FUNC_REG_3, GPIO_BIT_6);
```

TMRB 初期化構造体を用意し、TMRB モード、クロック、アップカウンタクリアメソッド、サイクル、デューティを設定します。本デモでは500usの周期を設定します。サイクルとデューティは Tmrb_Calculator 関数により求めます。

```
    TMRB_InitTypeDef m_tmrb;

    m_tmrb.Mode = TMRB_INTERVAL_TIMER;        /* internal timer */
    m_tmrb.ClkDiv = TMRB_CLK_DIV_8;           /* 1/8PhiT0 */
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
    for(i=0U;i<=4U;i++) {
        LeadingTiming[i] = Tmrb_Calculator(LeadingTimingus[i], m_tmrb.ClkDiv);
    }
    m_tmrb.TrailingTiming = Tmrb_Calculator(500U, m_tmrb.ClkDiv);     /* trailing
timing is 500us */
    m_tmrb. LeadingTiming = LeadingTiming[Rate];                      /*
leading timing, initial value 10% */
```

フリッププロップ初期化構造体を用意し、フリップフロップ制御、反転トリガ引数を設定します。反転トリガは、デューティとサイクルを一致するように設定します。

```
    PPGFFInital.FlipflopCtrl = TMRB_FLIPFLOP_SET;
    PPGFFInital.FlipflopReverseTrg=TMRB_FLIPFLOP_MATCH_TRAILINGTIMIN
G | TMRB_FLIPFLOP_MATCH_LEADINGTIMING;
```

TMRB モジュールを許可し、指定レジスタに初期化構造体、フリップフロップ構造体を設定します。ダブルバッファを許可し、キャプチャ機能を禁止にします。最後に、TMRB を動作させます。

```
TMRB_Enable(TSB_TB7);
TMRB_Init(TSB_TB7, &m_tmrb);
TMRB_SetFlipFlop(TSB_TB7, &PPGFFInital);
/* enable double buffer */
TMRB_SetDoubleBuf(TSB_TB7,ENABLE, TMRB_WRITE_REG_SEPARATE);
TMRB_SetRunState(TSB_TB7, TMRB_RUN);
```

スイッチが Low から High にまるまで待ち、同時に UART に現在のデューティを表示します。

```
do {                                /* wait if switch is Low */
        keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_0);
        LeadingTiming_display(); /* display current leading timing */
    } while (GPIO_BIT_VALUE_0 == keyvalue);
```

スイッチが High の場合、下記のようにデューティを設定します。
10%->25%->50%->75% ->90%その後 再び 90%から 10%になります。

```
Rate++;
if (Rate >= LEADINGTIMINGMAX) {
    Rate = LEADINGTIMINGINIT;
    } else {
        /* Do nothing */
    }

TMRB_ChangeLeadingTiming(TSB_TB7, LeadingTiming[Rate]);     /* change
leading timing rate */
```

Tmrb_Calculator 関数:

```
uint16_t Tmrb_Calculator(uint16_t Tmrb_Require_us, uint32_t ClkDiv)
{
    uint32_t T0 = 0U;
    const uint16_t Div[8U] = {1U, 2U, 8U, 32U, 64U, 128U, 256U, 512U};

    SystemCoreClockUpdate();

    T0 = SystemCoreClock / (1U << ((TSB_CG->SYSCR >> 8U) & 7U));
    T0 = T0/((Div[ClkDiv])*1000000U);

    return(Tmrb_Require_us * T0);
}
```

7-10 SIO/UART

7-10-1 例: リターゲット

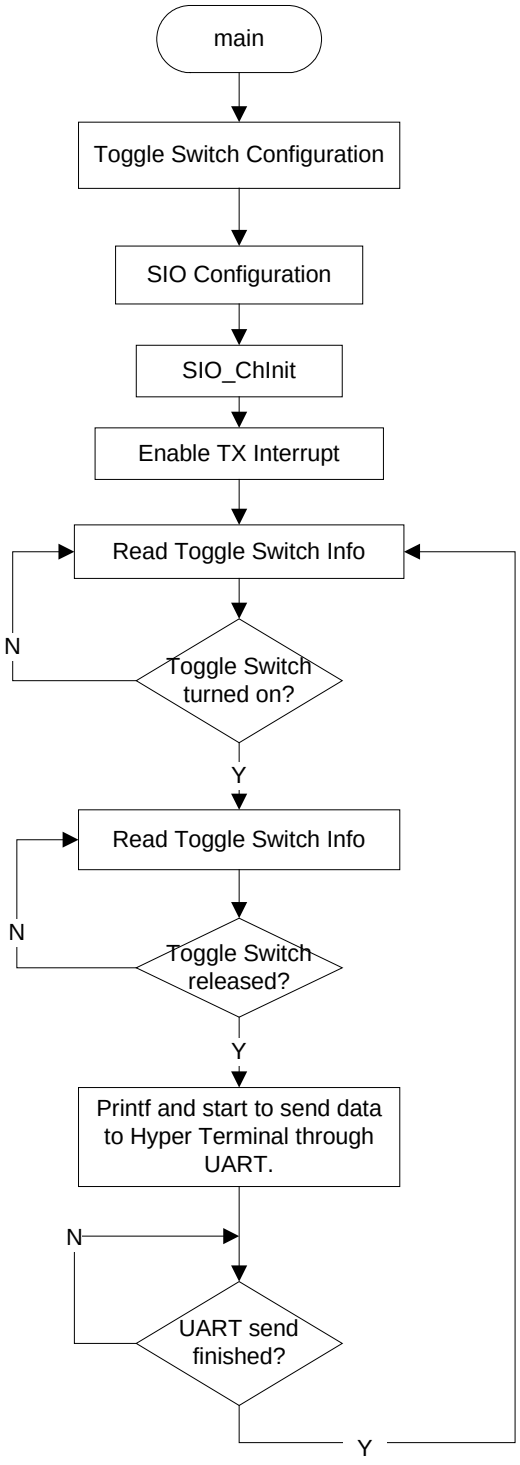
TX03 ペリフェラルドライバ (UART, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. UART 設定と初期化
2. UART 送信制御
3. データ送信に UART の TX 割り込みを使用
4. UART に printf()関数をリターゲット

この UART チャネルはハードウェアにより制限されています。下記の例では UART0 を使用しています。

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず、GPIO 設定と UART の初期化を行います。
GPIO ペリフェラルドライバを使い、GPIO をスイッチに設定します。

```
    GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_0, ENABLE);
```

```
    GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_0, DISABLE);
    GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_0);
```

UART_InitTypeDef 構造体を準備し、データを設定します。以下は設定例です。

```
    UART_InitTypeDef myUART;

    /* configure SIO0 for reception */
    UART_Enable(UART_RETARGET);
    myUART.BaudRate = 115200U;   /* baud rate = 115200 */
    myUART.DataBits = UART_DATA_BITS_8; /* no handshake, 8-bit data, clock
by baud rate generator */
    myUART.StopBits = UART_STOP_BITS_1; /* 1-bit stop, LSB, W-buff enable */
    myUART.Parity = UART_NO_PARITY;
    myUART.Mode = UART_ENABLE_TX;
    myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
    UART_Init(UART_RETARGET, &myUART);
```

上記設定を行い、その後、UART の送信割り込みを有効にします。

```
    NVIC_EnableIRQ(RETARGET_INT)
```

データ送信を開始します。TxBuffer は文字列配列です。

```
    printf("%s\r\n", TxBuffer);
```

データフローの残りのプロセスは UART0 送信割り込みルーチンの ISR にて終了します。
UART0 の送信割り込みルーチン:

```
void INTTX0_IRQHandler(void)
{
    if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
        UART_SetTxData(UART_RETARGET,     gSIOTxBuffer[gSIORdIndex++]);
/* send data */
        fSIO_INT = SET;              /* SIO0 INT is enable */
    } else {
        /* disable SIO0 INT */
        fSIO_INT = CLEAR;
        NVIC_DisableIRQ(RETARGET_INT);
        fSIOTxOK = YES;
    }
    if (gSIORdIndex >= gSIOWrIndex) {     /* reset buffer index */
        gSIOWrIndex = CLEAR;
        gSIORdIndex = CLEAR;
    } else {
        /* Do nothing */
    }
}
```

printf()関数は IAR コンパイラの putchar()を、RealView コンパイラの fputc()をコールしてデータ出力を行います。

```
#if defined ( __CC_ARM     )          /* RealView Compiler */
struct __FILE {
    int handle;                       /* Add whatever you need here */
};
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE * f)
#elif defined ( __ICCARM__ )       /*IAR Compiler */
```

```
int putchar(int ch)
#endif
{
	return (send_char(ch));
}

uint8_t send_char(uint8_t ch)
{
	while (gSIORdIndex != gSIOWrIndex) {		/* wait for finishing sending */
		/* Do nothing */
	}
	gSIOTxBuffer[gSIOWrIndex++] = ch;	/* fill TxBuffer */
	if (fSIO_INT == CLEAR) {	/* if SIO INT disable, enable it */
		fSIO_INT = SET;		/* set SIO INT flag */
		UART_SetTxData(UART_RETARGET, gSIOTxBuffer[gSIORdIndex++]);
		NVIC_EnableIRQ(RETARGET_INT);
	}

	return ch;
}
```

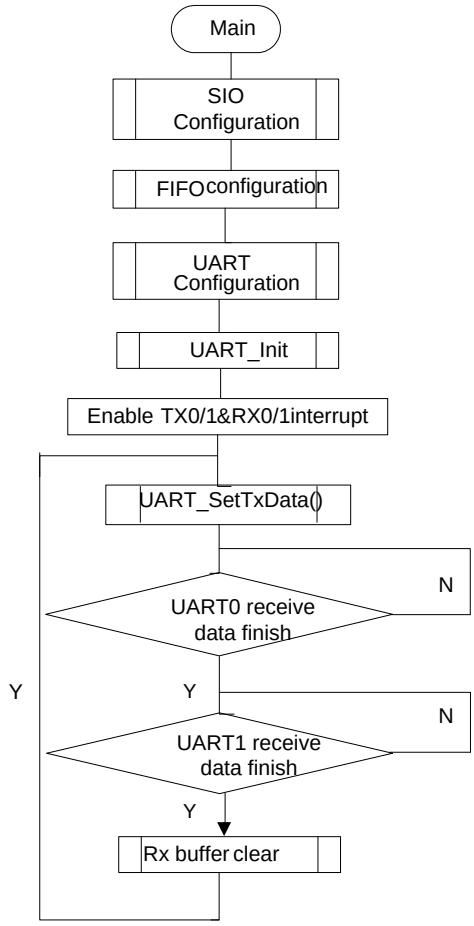
7-10-2　例: UART の FIFO デモ

UART と GPIO を使用した簡単なサンプルプログラムです。

以下の例が含まれます:

1. UART と FIFO の初期設定
2. FIFO を使用した UART の送受信

- **フローチャート**



- **サンプルプログラムのコードと説明**

最初に GPIO の設定と UART の初期化を行います。

GPIO ドライバを使用して、GPIO を UART0 と UART1 に設定します。

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
	if (SCx == TSB_SC0) {
		TSB_PE->CR |= GPIO_BIT_0;
		TSB_PE->FR1 |= GPIO_BIT_0;
		TSB_PE->FR1 |= GPIO_BIT_1;
		TSB_PE->IE |= GPIO_BIT_1;
	} else if (SCx == TSB_SC1) {
```

```
        TSB_PF->CR |= GPIO_BIT_0;
        TSB_PF->FR5 |= GPIO_BIT_0;
        TSB_PB->FR5 |= GPIO_BIT_6;
        TSB_PB->IE |= GPIO_BIT_6;
    }
}
```

UART_InitTypeDef 構造体を用意し、すべてのメンバを設定します。

```
    UART_InitTypeDef myUART;

    /* configure SIO0 for reception */
    UART_Enable(UART_RETARGET);
    myUART.BaudRate = 115200U;    /* baud rate = 115200 */
    myUART.DataBits = UART_DATA_BITS_8; /* no handshake, 8-bit data, clock by
baud rate generator */
    myUART.StopBits = UART_STOP_BITS_1; /* 1-bit stop, LSB,    W-buff enable */
    myUART.Parity = UART_NO_PARITY;
    myUART.Mode = UART_ENABLE_TX|UART_ENABLE_RX;
    myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
    UART_Init(UART_RETARGET, &myUART);
```

UART のドライバを使用して、UART0/1 の各チャネルの許可と初期設定を行います。

```
     UART_Enable(UART0);
     UART_Init(UART0, &myUART);

     UART_Enable(UART1);
     UART_Init(UART1, &myUART);
```

FIFO の設定を行います。

```
    UART_RxFIFOByteSel(UART0,UART_RXFIFO_RXFLEVEL);
    UART_RxFIFOByteSel(UART1,UART_RXFIFO_RXFLEVEL);

    UART_TxFIFOINTCtrl(UART0,ENABLE);
    UART_TxFIFOINTCtrl(UART1,ENABLE);

    UART_RxFIFOINTCtrl(UART0,ENABLE);
    UART_RxFIFOINTCtrl(UART1,ENABLE);

    UART_TRxAutoDisable(UART0,UART_RXTXCNT_AUTODISABLE);
    UART_TRxAutoDisable(UART1,UART_RXTXCNT_AUTODISABLE);

    UART_FIFOConfig(UART0,ENABLE);
    UART_FIFOConfig(UART1,ENABLE);

    UART_RxFIFOFillLevel(UART0, UART_RXFIFO4B_FLEVLE_4_2B);
    UART_RxFIFOFillLevel(UART1, UART_RXFIFO4B_FLEVLE_4_2B);

    UART_RxFIFOINTSel(UART0,UART_RFIS_REACH_EXCEED_FLEVEL);
    UART_RxFIFOINTSel(UART1,UART_RFIS_REACH_EXCEED_FLEVEL);

    UART_RxFIFOClear(UART0);
    UART_RxFIFOClear(UART1);

    UART_TxFIFOFillLevel(UART0, UART_TXFIFO4B_FLEVLE_0_0B);
    UART_TxFIFOFillLevel(UART1, UART_TXFIFO4B_FLEVLE_0_0B);

    UART_TxFIFOINTSel(UART0,UART_TFIS_REACH_NOREACH_FLEVEL);
```

```
    UART_TxFIFOINTSel(UART1,UART_TFIS_REACH_NOREACH_FLEVEL);

    UART_TxFIFOClear(UART0);
    UART_TxFIFOClear(UART1);
```

上記設定を行い、その後 UART0/1 の各割り込みを許可します。

```
    NVIC_EnableIRQ(INTTX0_IRQn);
    NVIC_EnableIRQ(INTRX1_IRQn);

    NVIC_EnableIRQ(INTTX1_IRQn);
    NVIC_EnableIRQ(INTRX0_IRQn);
```

最後に UART0/1 の送信割り込みと受信割り込みの割り込み処理ルーチンを準備します。
以下は UART0 の送信割り込み処理ルーチンです。

```
void INTTX0_IRQHandler(void)
{
    volatile UART_Err err;

    if (TxCounter < NumToBeTx) {
        UART_SetTxData(UART0, TxBuffer[TxCounter++]);
    } else {
        err = UART_GetErrState(UART0);
    }
}
```

以下は UART1 の送信割り込み処理ルーチンです。

```
void INTTX1_IRQHandler(void)
{
    volatile UART_Err err;

    if (TxCounter1 < NumToBeTx1) {
        UART_SetTxData(UART1, TxBuffer1[TxCounter1++]);
    } else {
        err = UART_GetErrState(UART1);
    }
}
```

以下は UART0 の受信割り込み処理ルーチンです。

```
void INTRX0_IRQHandler(void)
{
    volatile UART_Err err;

    err = UART_GetErrState(UART0);
    if (UART_NO_ERR == err) {
        RxBuffer[RxCounter++] = (uint8_t) UART_GetRxData(UART0);
    }
```

以下は UART1 の受信割り込み処理ルーチンです。

```
void INTRX1_IRQHandler(void)
{
    volatile UART_Err err;

    err = UART_GetErrState(UART1);
    if (UART_NO_ERR == err) {
        RxBuffer1[RxCounter1++] = (uint8_t) UART_GetRxData(UART1);
    }
```

```
}
```

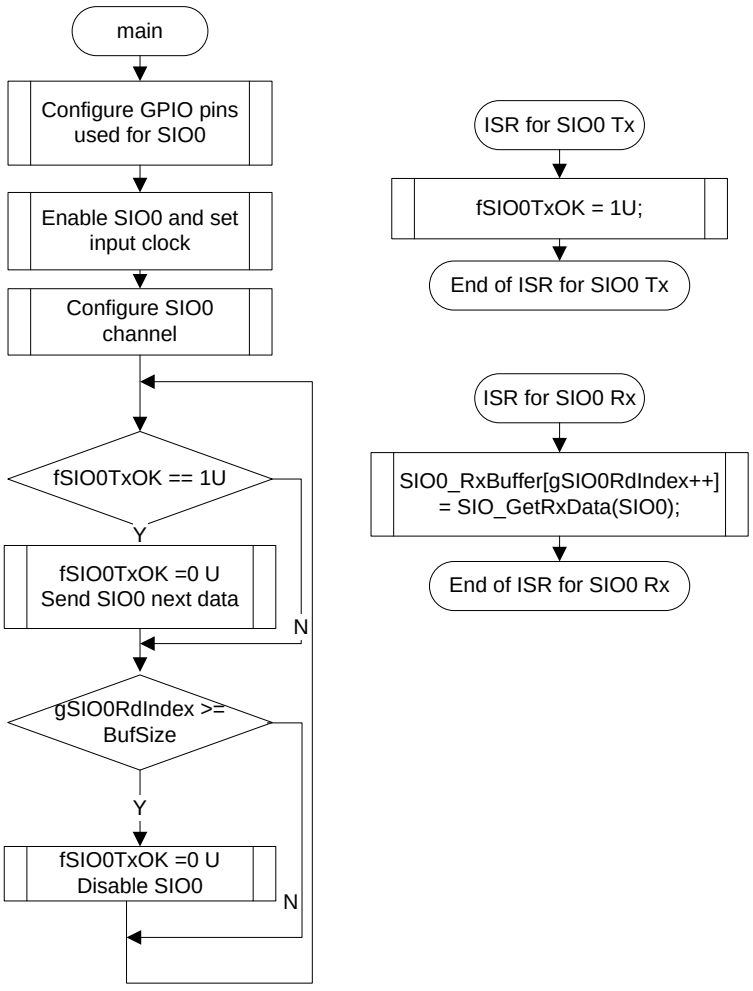
7-10-3　例: SIO マスタ

TX03 ペリフェラルドライバ (SIO, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます。

1. SIO の基本設定
2. マスタ SIO0⇔スレーブ SIO0 間の通信制御
3. 送受信に SIO 割り込みを使用

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず、GPIO 設定と SIO の初期化を行います。

GPIO ペリフェラルドライバを使い、GPIO を SIO0 に設定します。その後、SIO0 の初期化構造体を準備し、入力クロックの初期化を行います。

```
    /*Enable the SIO0 channel */
```

```
    SIO_Enable(SIO0);

    /* Selects input clock for prescaler as PhiT0. */
    SIO_SetInputClock(SIO0, SIO_CLOCK_T0);

    /*initialize the SIO0 struct */
        SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
        SIO0_Init.TIDLE = SIO_TIDLE_HIGH;
        SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_1;
        SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
        SIO0_Init.TransferDir = SIO_LSB_FRIST;
        SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
        SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
        SIO0_Init.BaudRateClock = SIO_BR_CLOCK_TS2;
        SIO0_Init.Divider = SIO_BR_DIVIDER_2;
        SIO_Init(SIO0, SIO_CLK_BAUDRATE, &SIO0_Init);
```

SIO の送受信割り込みを有効にします。

```
    /* Enable SIO0 Channel TX interrupt */
    NVIC_EnableIRQ(INTTX0_IRQn);
    /* Enable SIO0 Channel RX interrupt */
    NVIC_EnableIRQ(INTRX0_IRQn);
```

すべての基本設定を行い、その後、データ送信を行います。

```
        while (1) {
            /*SIO0 send data from TXD0 */
            if (fSIO0TxOK == 1U) {
                fSIO0TxOK = 0U;
                if (gSIO0WrIndex < BufSize) {
                SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
                }
            } else {
                /*Do Nothing */
            }

           /*SIO0 receive data end */
            if (gSIO0RdIndex >= BufSize) {
                SIO_Disable(SIO0);
            } else {
                /*Do Nothing */
            }
        }
```

SIO0 送信の ISR にて、転送設定を行います。

```
void INTTX0_IRQHandler (void)
{
    fSIO0TxOK = 1U;
}
```

SIO0 受信の ISR にて、受信バッファからデータを受信します。

```
void INTRX0_IRQHandler(void)
{
    SIO0_RxBuffer[gSIO0RdIndex++] = SIO_GetRxData(SIO0);
}
```

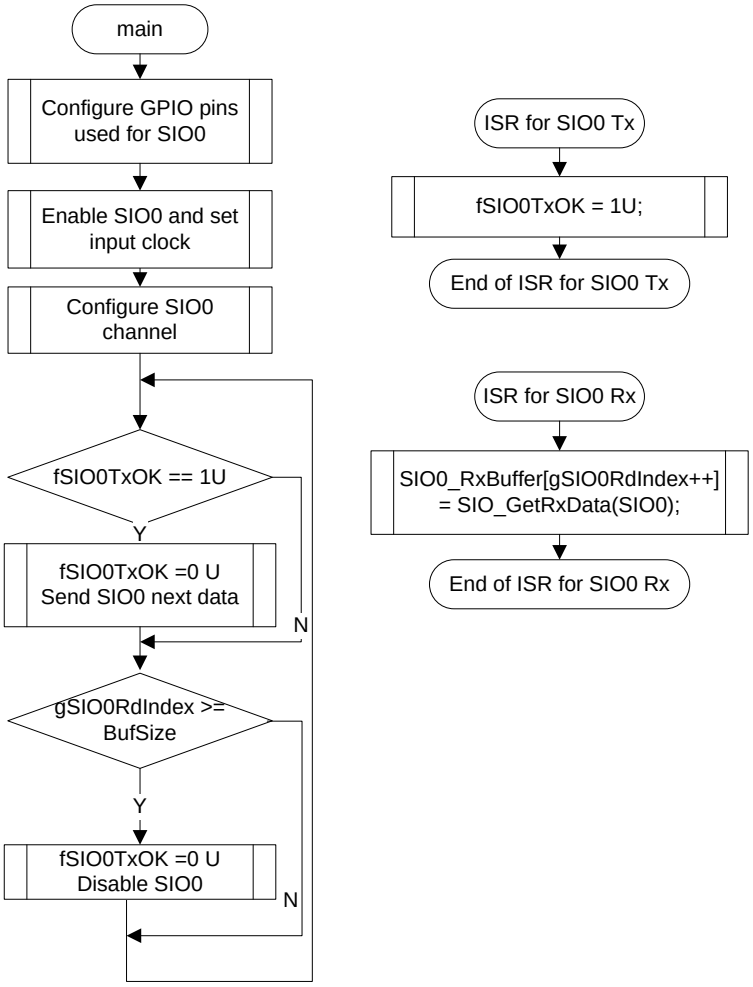
7-10-4　例: SIO スレーブ

TX03 ペリフェラルドライバ (SIO, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます。

1. SIO の基本設定
2. マスタ SIO0⇔スレーブ SIO0 間の通信制御
3. 送受信に SIO 割り込みを使用

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず、GPIO 設定と SIO の初期化を行います。

GPIO ペリフェラルドライバを使い、GPIO を SIO0 に設定します。

```
/*configure the SIO0 channel */
SIO_Enable(SIO0);
/* Selects input clock for prescaler as PhiT0. */
SIO_SetInputClock(SIO0, SIO_CLOCK_T0);

/*initialize the SIO0 struct */
```

```
    SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
    SIO0_Init.TIDLE = SIO_TIDLE_HIGH;
    SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
    SIO0_Init.TransferDir = SIO_LSB_FRIST;
    SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
    SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
    SIO0_Init.TXDEMP = SIO_TXDEMP_HIGH;
    SIO0_Init.EHOLDTime = SIO_EHOLD_FC_64;
    SIO_Init(SIO0, SIO_CLK_SCLKINPUT, &SIO0_Init);
```

SIO の送受信割り込みを有効にします。

```
  /* Enable SIO0 Channel TX interrupt */
    NVIC_EnableIRQ(INTTX0_IRQn);
    /* Enable SIO0 Channel RX interrupt */
    NVIC_EnableIRQ(INTRX0_IRQn);
```

すべての基本設定を行い、その後、データ送信を行います。

```
        while (1) {
            /*SIO0 send data from TXD0 */
            if (fSIO0TxOK == 1U) {
                fSIO0TxOK = 0U;
                if (gSIO0WrIndex < BufSize) {
                SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
                }
            } else {
                /*Do Nothing */
            }

           /*SIO0 receive data end */
            if (gSIO0RdIndex >= BufSize) {
                SIO_Disable(SIO0);
            } else {
                /*Do Nothing */
            }
        }
```

SIO0 送信の ISR にて、転送設定を行います。

```
void INTTX0_IRQHandler (void)
{
    fSIO0TxOK = 1U;
}
```

SIO0 受信の ISR にて、受信バッファからデータを受信します。

```
void INTRX0_IRQHandler(void)
{
    SIO0_RxBuffer[gSIO0RdIndex++] = SIO_GetRxData(SIO0);
}
```

7-11 ウォッチドッグタイマ(WDT)

7-11-1 例:ウォッチドッグタイマ

TX03 ペリフェラルドライバ (WDT, GPIO) のプログラムサンプルです。

以下の例が含まれます:

1. WDT の初期化
2. DEMO1 では、オーバーフロー前に WDT クリアを行わず、NMI 割り込みを発生させます。
3. DEMO2 では、オーバーフロー前にWDT クリアを行い、常時LED0 を点滅させます。

- **サンプルプログラムのコードと説明**

以下のコードは WDT の初期化の例です。検出時間が 2^25/fsys に設定されオーバーフロー時に NMI 割り込みを発生します。

```
WDT_InitTypeDef WDT_InitStruct;
WDT_InitStruct.DetectTime = WDT_DETECT_TIME_EXP_25;
WDT_InitStruct.OverflowOutput = WDT_NMIINT;
```

WDT を初期化し、その後 WDT を有効にします。

```
WDT_Init(&WDT_InitStruct);
WDT_Enable();
```

DEMO1 では、NMI 割り込みの発生を待ちます。

```
while(1)
{
}
```

DEMO1 では、NMI 割り込み発生時に WDT を禁止にし、LED1 の点滅を停止します。

```
WDT_Disable();
```

DEMO2 では、WDT クリアを行い、常に LED0 を点滅させます。

```
WDT_WriteClearCode();
```

7-12 PMD

7-12-1 例: 位相出力

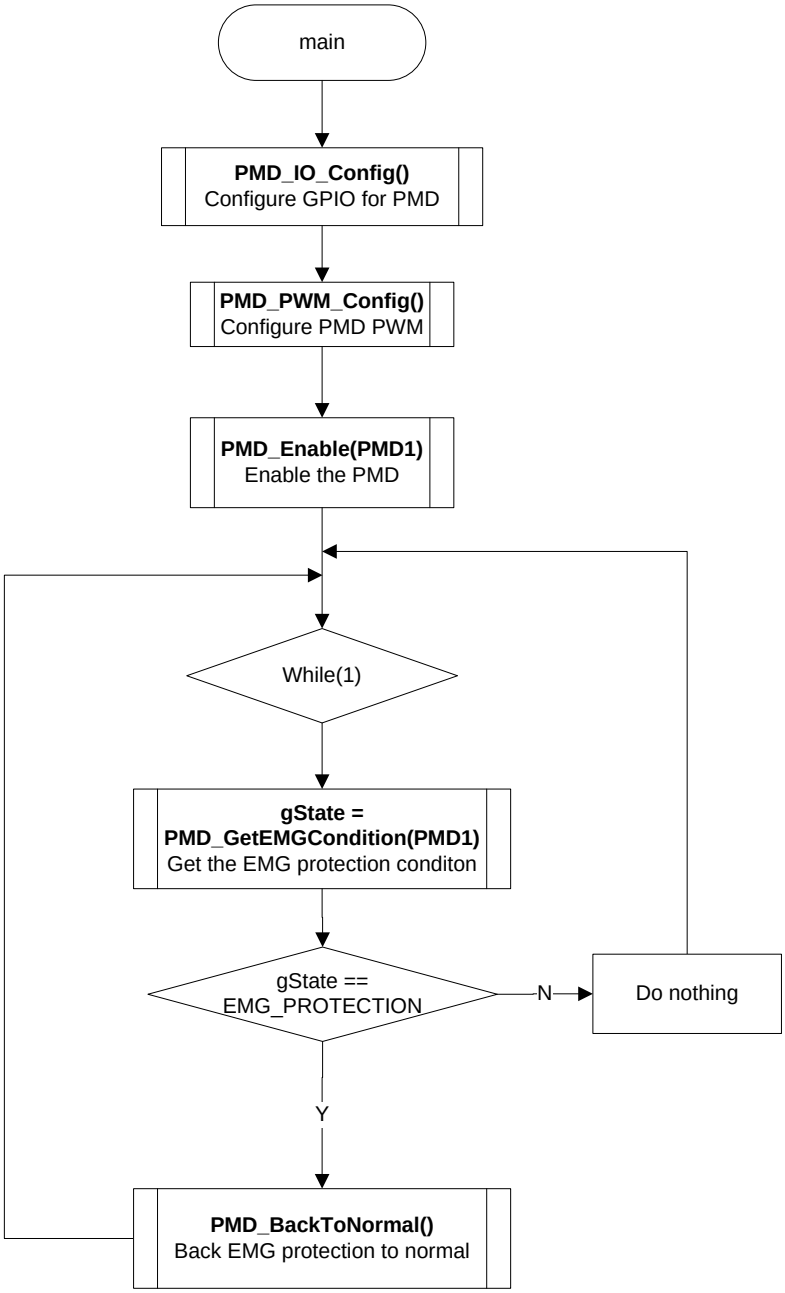
TX03 ペリフェラルドライバ (PMD, GPIO) のプログラムサンプルです。

この例では以下を行います。

1. PMD と GPIO の初期化

2. EMG 保護検出出力
3. EMG 保護検出出力から通常出力への状態変化

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず LED、PMD、GPIO の初期化を行います。

```
/* LED initialization */
LED4_Init();

/* GPIO configuration*/
PMD_IO_Config();
```

```
    /* PMD PWM configuration*/
    PMD_PWM_Config();
```

DEMO_U_PHASE 定義を行う場合、DUTY モードは U 相共通です。
DEMO_3_PHASE 定義を行う場合、DUTY モードは 3 相独立です。

```
/* PMD1 initialization */
    m_pmd.CycleMode = PMD_PWM_NORMAL_CYCLE;
#ifdef DEMO_U_PHASE
    m_pmd.DutyMode = PMD_DUTY_MODE_U_PHASE;          /* U-phase in
common */
#endif
#ifdef DEMO_3_PHASE
    m_pmd.DutyMode = PMD_DUTY_MODE_3_PHASE;          /* 3-phase
independent */
#endif
    m_pmd.IntTiming = PMD_PWM_INT_TIMING_MINIMUM;
    m_pmd.IntCycle = PMD_PWM_INT_CYCLE_1;
    m_pmd.CarrierMode = PMD_CARRIER_WAVE_MODE_1;     /* PWM mode 1
(center PWM, triangle wave) */
    m_pmd.CycleTiming = 0x3FFFU;

    PMD_Init(PMD1,&m_pmd);
```

3 相のコンペア値を設定します。
DUTY モードが U 相共通の場合、U 相と同じ出力です。
DUTY モードが 3 相独立の場合、Y/V/W それぞれの出力です。

```
    PMD_SetAllPhaseCompareValue(PMD1,0x0FFFU,0x1FFFU,0x2FFFU);
```

PMD 動作を許可します。

```
    PMD_Enable(PMD1);
```

EMG 保護検出の判定を行います。保護検出状態の場合は LED4 を点灯し、EMG 保護検出出力から通常出力へ戻します。保護検出状態ではない場合は LED4 を消灯します。

```
    while(1){
        /* Get the EMG protection condition*/
        gState = PMD_GetEMGCondition(PMD1);
        /* Judge the EMG protection condition */
        if (gState == EMG_PROTECTION) {
            /* LED4 will light on if the condition is protection */
            LED4_On();
            /* Back EMG protection to normal */
            PMD_BackToNormal();
            Delay(1000U);
        } else {
            /* LED4 will light off if the condition is not protection */
            LED4_Off();
        }    }
```