

**TOSHIBA**

# **TX03 ペリフェラルドライバ使用例 (TMPM365)**

第一版

2017 年 9 月

**東芝デバイス&ストレージ株式会社**

## **本製品取り扱い上のお願い**

- ソフトウェア使用権許諾契約書の同意無しに使用しないで下さい。

## 目次

|       |                      |    |
|-------|----------------------|----|
| 1     | はしがき .....           | 1  |
| 2     | 概要 .....             | 1  |
| 3     | 使用する機能 .....         | 2  |
| 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 .....          | 5  |
| 6-2   | ADC .....            | 5  |
| 6-3   | CG .....             | 6  |
| 6-3-1 | Power モード変更 .....    | 6  |
| 6-4   | DMAC .....           | 6  |
| 6-4-1 | メモリから周辺回路 .....      | 6  |
| 6-5   | Flash .....          | 6  |
| 6-6   | GPIO .....           | 8  |
| 6-7   | SBI .....            | 8  |
| 6-8   | TMRB .....           | 9  |
| 6-8-1 | 汎用タイマ .....          | 9  |
| 6-8-2 | PPG 出力 .....         | 9  |
| 6-9   | SIO/UART .....       | 9  |
| 6-9-1 | リターゲット .....         | 9  |
| 6-9-2 | UART FIFO .....      | 9  |
| 6-9-3 | SIO .....            | 10 |
| 6-10  | WDT .....            | 10 |
| 7     | ソフトウェア .....         | 11 |
| 7-1   | ADC .....            | 13 |
| 7-1-1 | 例: ADC データリード .....  | 13 |
| 7-2   | CG .....             | 16 |
| 7-2-1 | 例: Power モード変更 ..... | 16 |
| 7-3   | DMAC .....           | 20 |
| 7-3-1 | 例: メモリから周辺回路 .....   | 20 |
| 7-4   | FLASH .....          | 22 |
| 7-5   | GPIO .....           | 25 |
| 7-6   | SBI .....            | 26 |

---

|                         |    |
|-------------------------|----|
| 7-7 TMRB .....          | 30 |
| 7-7-1例: 汎用タイマ .....     | 30 |
| 7-7-2例: PPG 出力 .....    | 31 |
| 7-8 SIO/UART.....       | 34 |
| 7-8-1例: リターゲット.....     | 34 |
| 7-8-2例: UART FIFO ..... | 37 |
| 7-8-3例: SIO.....        | 40 |
| 7-9 WDT .....           | 43 |

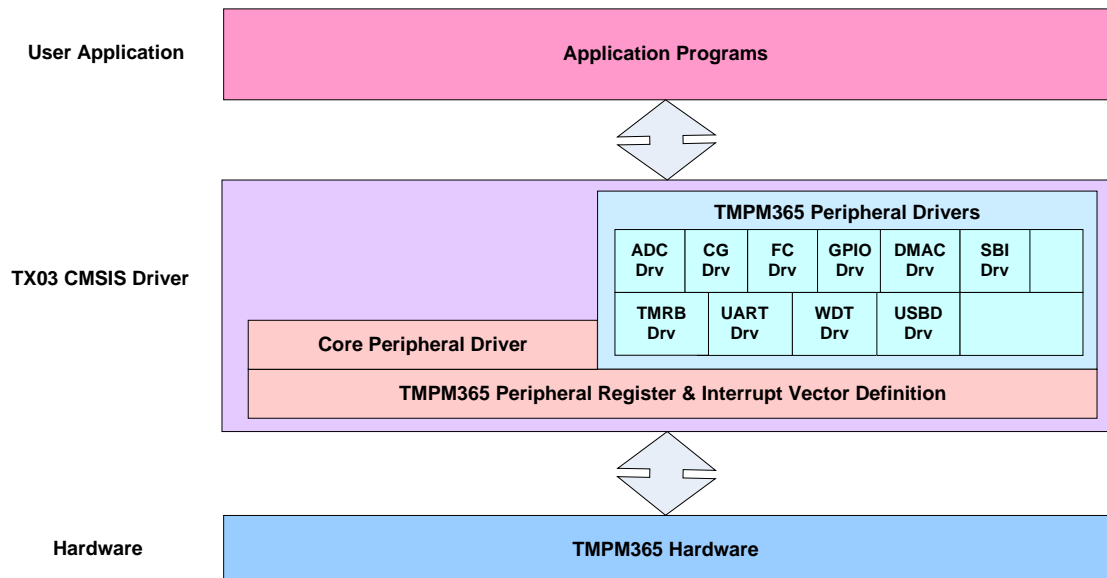
---

## 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TMPM365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                           | ブロック/ユニット/チャンネル | 使用／未使用                              |
|------------------------------|-----------------|-------------------------------------|
| ADC                          | AIN0            | 使用(ADC サンプル)                        |
|                              | 上記以外のチャンネル      | 未使用                                 |
| CG                           | -               | 使用(CG サンプル)                         |
| 低消費電力モード                     | -               | STOP1 モード                           |
| 外部割込み<br>( INT )             | INT0            | 使用(CG サンプル)                         |
|                              | 上記以外のチャンネル      | 未使用                                 |
| DMAC                         | -               | 使用(DMAC サンプル)                       |
| シリアルチャンネル<br>(SIO/UART)      | SIO0            | 使用(UART リターゲットサンプル)                 |
|                              | SIO1            | 使用( UART_FIFO サンプル、SIO サンプル)        |
| 16 ビットタイマ/イベント<br>カウンタ(TMRB) | TMRB0           | 使用(TMRB: 汎用タイマサンプル、PPG<br>波形出力サンプル) |
|                              | 上記以外のチャンネル      | 未使用                                 |
| シリアルバスインタフェース(SBI)           | SBI0            | 使用(SBI サンプル: マスタ Tx)                |
|                              | SBI1            | 使用(SBI サンプル: スレーブ Rx)               |
| ウォッチドッグタイマ<br>( WDT )        | WDT0            | 使用(WDT サンプル)                        |

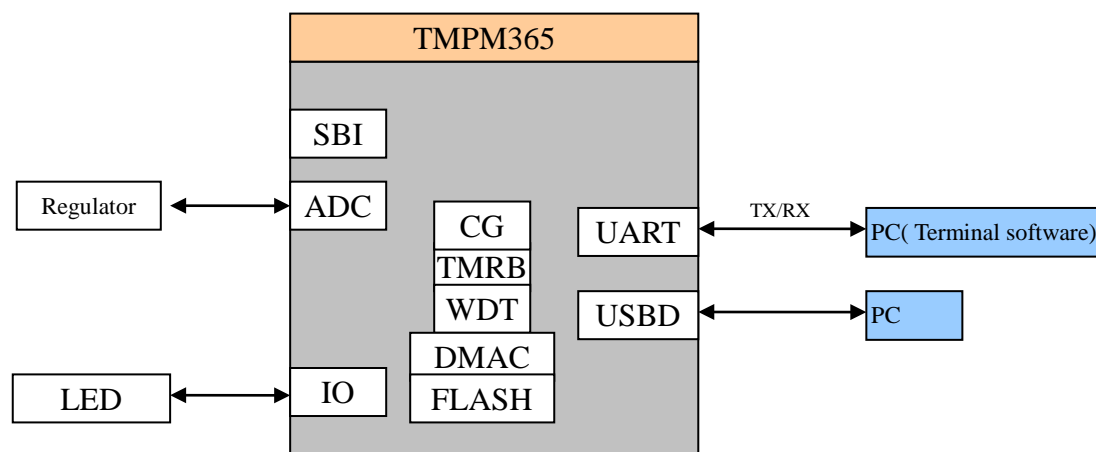
## 4 端子用途

本サンプルプログラムは、TMPM365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。

| No  | Name     | Usage                  |
|---|----------|------------------------|
| R10   | PA0      | LED0, USBドサンプルのプルアップ制御 |
| R11   | PA1      | LED1                   |
| P11   | PA2      | LED2                   |
| R12   | PA3      | LED3                   |
| N15   | PB4      | SW0                    |
| M14   | PB5      | SW1                    |
| R2  | PC0/Txd1 | UART1_Tx               |
| R3  | PC1/Rxd1 | UART1_Rx               |
| D15   | PD1      | TMRBサンプルのスイッチ          |
| A15   | PE0/Txd0 | UART0_Tx               |
| C14   | PE1/Rxd0 | UART0_Rx               |
| B10   | PE4/SDA1 | SBIサンプル, SDA1          |
| B9  | PE5/SCL1 | SBIサンプル, SCL1          |
| P5  | PG0/SDA0 | SBIサンプル, SDA0          |
| P4  | PG1/SCL0 | SBIサンプル, SCL0          |
| R1  | PG3/INT0 | For CGサンプル, INT0       |
| L2  | PI3      | SWD デバッグ用 SWCLK        |
| K2  | PI4      | SWD デバッグ用 SWDIO        |
| G15   | D+       | USBド サンプル用 D+端子        |
| H15   | D-       | USBド サンプル用 D-端子        |
| F1<br>R8<br>J14<br>B13<br>H14<br>A7<br>E1<br>E2 | GND      | GND                    |

## 5 開発環境

以下に開発環境の構成を示します。



1. ハードウェア:  
TMPM365 評価ボード (非売品)
2. 開発ツール:
  - IAR:
    - 1) J-Link: IAR J-Link-7.0/8.0
    - 2) IDE: IAR Embed Workbench for ARM version 6.40
  - KEIL:
    - 1) u-Link: Realview ULINK2
    - 2) IDE: KEIL uVision MDK version 4.21



## 6 機能説明

### 6-1 動作モード

本デバイスには3つの動作モードがあります: NORMAL, IDLE, STOP1 モード

IDLE と STOP1 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、CGSTBYCR<STBY2:0>にて IDLE、STOP1 のいずれかのモードを選択し、WFI (Wait For Interrupt) 命令を実行します。

#### ➤ NORMAL モード:

このモードは、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。

補足: STOP1 モードのサンプルが含まれています。

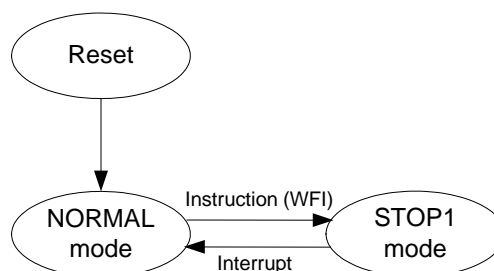
#### ➤ STOP1 モード:

STOP1 モードは、内蔵発振器も含めてすべての内蔵回路が停止するモードです。STOP1 モードが解除されると内蔵発振器が発振を開始し、NORMAL モードへ復帰します。CGSTBYCR<DRVE>を設定することにより、STOP1 モード中は端子のドライブ状態を保持することができます。

STOP1 モードの解除要因により、STOP1 モードから NORMAL モードへ復帰する場合、ウォーミングアップカウンタは自動的に起動します。この場合のウォーミングアップ時間は内蔵 Flash の安定時間として、450  $\mu$ s 以上を設定してください。

リセットで NORMAL モードへ復帰する場合、ウォーミングアップは行われませんので、発振動作が安定するまでのリセット信号を有効に保ってください。

補足: STOP1 モードのサンプルが含まれています。



### 6-2 ADC

AIN0 に接続された VR3 の値を変更します。測定された電圧値は標準出力ライブラリ(stdout)を経由で出力します。stdout は UART にリターゲットされますので、PC と UART0 を接続してください。

AD 変換結果は、ターミナル出力ソフトに表示されます。

UART の設定: ボーレート=115200bps, 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

## 6-3 CG

### 6-3-1 Power モード変更

CPU 動作モードを変更するシンプルなデモです。NORMAL モードと STOP1 モードの 2 つのモードを使用します。

| 現在のモード | アクション (Key)             | 動作            | ターミナル表示             | LED 表示  |
|--------|-------------------------|---------------|---------------------|---------|
| NORMAL | SW1を押す (Lowアクティブ)       | NORMAL→ STOP1 | Now, Going to Stop1 | LED0 オフ |
| STOP1  | PG3(INT0)-VCC(3.3V)ショート | STOP1 →NORMAL | Wakeup from Stop1   | LED0 オン |

## 6-4 DMAC

### 6-4-1 メモリから周辺回路

UART0 から UART1 へのデータ転送を行う処理が実装されています。“TOSHIBA” の文字列データを UART0 から UART1 へ送信されます。

DMAC により、RAM から UART0 データレジスタにデータが送信され、文字列の各文字データは UART0 経由で送信され、UART1 で受信します。UART1 のデータ受信後、データは文字配列に保存されます。

デバッガの変数ウィンドウに文字配列変数を登録して、受信されたデータを確認します。

**補足:** サンプルソフトウェアを使用するには、TMPM365 評価ボードの設定変更を行います。  
UART0(TX)と UART1(RX)を接続します。

## 6-5 Flash

Flash のドライバ API を使用して内蔵フラッシュメモリの消去及び再書き込みを行うサンプルプログラムです。

このプログラムは、シングルチップモードのノーマルモードとユーザブートモードで実行します。

ーリセット動作: モード判定、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。

- ・モード判定ルーチン: ユーザブートモードまたはノーマルモードの判定を行います。
- ・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、フラッシュ上のプログラム A とプログラム B のコードを入れ替えます。

ープログラム A/B (リセット動作)は事前にフラッシュメモリに書き込まれています。

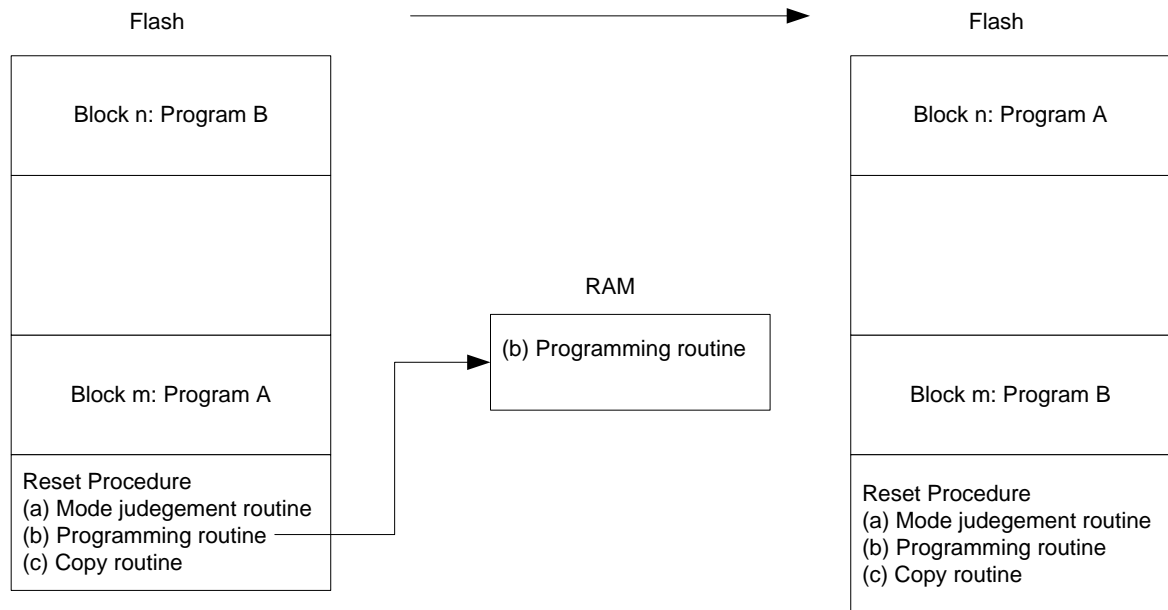
ープログラム A/B (A: LED 0 点滅, B: LED 1 点滅)

初期状態は、プログラム A が最初に動作します。

ーSW0 キーはリセット動作のモード判定に使用します。

SW0 キーが押されていない場合 → Normal モードです。

SW0 キーが押された場合 → ユーザブートモードです。



## 動作シーケンス

### (1) 電源投入

SW0 を OFF した状態で電源投入またはリセット端子を ON してください。まずプログラム A が実行され、LED0 が点滅します。

### (2) SW0 を ON している間にリセット端子を ON し、LED3 が点灯中に SW0 を OFF します。

プログラムルーチンを RAM に転送します。その後、プログラム A と B のスワップを行います。

### (3) スワップ完了後、システムリセットを行います。

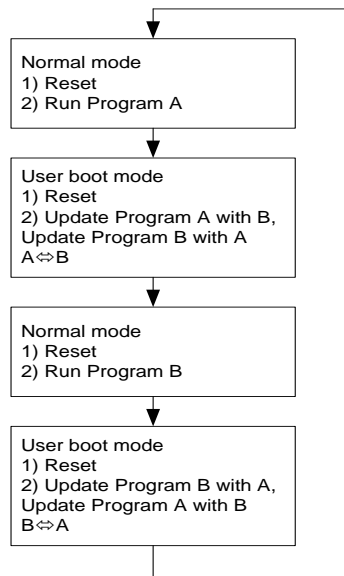
LED1 を点滅し、プログラム B が動作します。

### (4) 再度 SW0 を ON した状態でリセット端子を ON し、LED3 が点灯中に SW0 を OFF します。

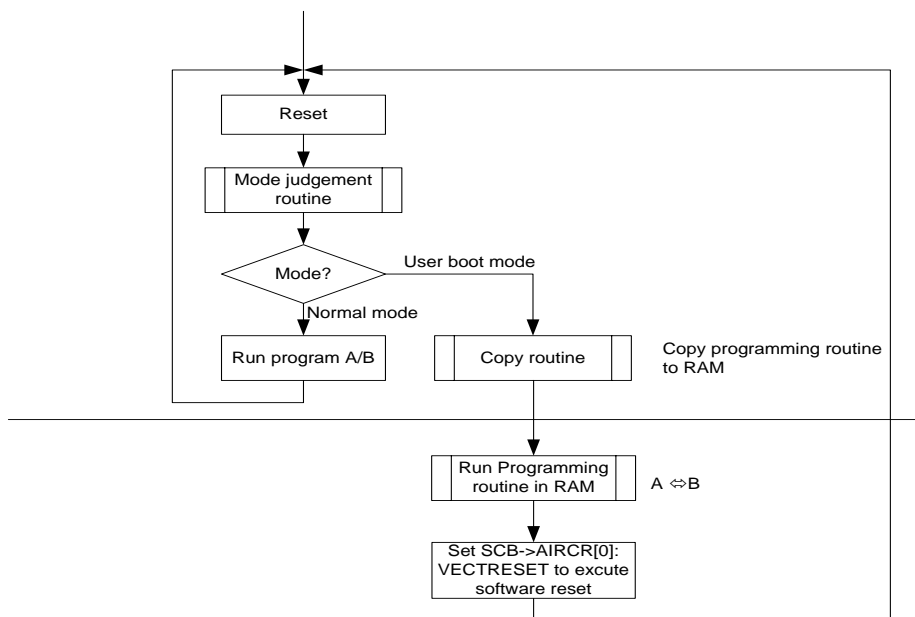
手順(2)と同じ動作を行います。

### (5) 再実行後、システムリセットを行います。

LED0 を点滅し、プログラム A が動作します。



## ・サンプルプログラムのフローチャート



## 6-6 GPIO

ペリフェラルドライバ(GPIO)を使用した簡単なサンプルプログラムです。GPIOポートをLEDポートとして使用します。

## 6-7 SBI

ペリフェラルドライバの I2C を使用し、I2C バスのスレーブモード処理を行うサンプルプログラムです。

評価ボードの 2 本の I2C バス (SBI0 & SBI1) を接続します。

片方は I2C バスのマスタモードで動作し、片方は I2C バスのスレーブモードで動作します。

I2C 割り込みを使用して I2C バスのリード/ライトを行います。

I2C のスレーブアドレス: 0xB0.

I2C スレーブ (SBI1) は I2C マスタ (SBI0) から "TOSHIBA" を受信します。

受信結果はデバッガにて RAM 変数 gl2CrxData[] にて確認できます。

**補足:** 本サンプルソフトを実行するためには、以下の接続を行った TMPM365 評価ボードが必要です。

PG1 (SCL0) 端子と PE5 (SCL1) 端子を接続します。

PG0 (SDA0) 端子と PE4 (SDA1) 端子を接続します。

## 6-8 TMRB

### 6-8-1 汎用タイマ

TMRB を使って汎用タイマを実現するサンプルプログラムです。

時間周期は 1ms 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。

### 6-8-2 PPG 出力

1 つのスイッチを使い、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。

スイッチの ON/OFF を切り替えることでデューティを変更します。

10% → 25% → 50% → 75% → 90% → 10%

## 6-9 SIO/UART

### 6-9-1 リターゲット

この例では、C 言語の標準入出力ライブラリの stdin、stdout のリターゲットを行います。

stdin、stdout を共に UART0 へ設定します。アプリケーションから printf() と getchar() 関数を用いて、シリアルポートからデータを入出力します。

### 6-9-2 UART FIFO

UART0 から "TMPM3651" というデータを FIFO を使用して UART1 へ送信し、同時に UART1 から

"TMPM3652"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、RxBuffer="TMPM3652"と  
RxBuffer1="TMPM3651"となった場合にブレークポイントで停止します。

## 6-9-3 SIO

SIO 機能を使用して同期式の送受信を行うサンプルプログラムです。  
SIO のチャンネル 0 とチャンネル 1 を使用し、これらのチャン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)

## 6-10 WDT

リセットが行われるとウォッチドッグタイマは有効となるので、ウォッチドッグタイマを使用しない場合は無効にしてください。ウォッチドッグタイマは、高周波クロックが停止している場合、使用できません。下記の動作モードへ移行する前に、ウォッチドッグタイマを無効にしてください。IDLEモードでは、ウォッチドッグタイマの動作はWDMOD<I2WDT> 設定に依存します。

### ドライバ使用方法ステップ:

1. 検出時間の設定とカウンタ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。

## 7 ソフトウェア

本ソフトウェアは、TMPM365 MCU の主要機能を TMPM365 評価ボード上で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:

```
\---TMPM365
  +---Libraries
  |   +---TX03_CMSIS
  |   |   |   system_TPM365.c
  |   |   |   system_TPM365.h
  |   |   |   TPM365.h
  |   |   \---startup
  |   |       +---arm
  |   |       |   startup_TPM365.s
  |   |       \---iar
  |   |           startup_TPM365.s
  |   \---TX03_Periph_Driver
  |       +---inc
  |       |   tmpm365_adc.h
  |       |   tmpm365_cg.h
  |       |   tmpm365_dmac.h
  |       |   tmpm365_fc.h
  |       |   tmpm365_gpio.h
  |       |   tmpm365_sbi.h
  |       |   tmpm365_tmrb.h
  |       |   tmpm365_uart.h
  |       |   tmpm365_wdt.h
  |       |   tx03_common.h
  |       \---src
  |       |   tmpm365_adc.c
  |       |   tmpm365_cg.c
  |       |   tmpm365_dmac.c
  |       |   tmpm365_fc.c
  |       |   tmpm365_gpio.c
  |       |   tmpm365_sbi.c
  |       |   tmpm365_tmrb.c
  |       |   tmpm365_uart.c
```

```
|      |      tmpm365_wdt.c
\---Project
    +---Examples          //only 1 examples listed here
    |   +---ADC
    |   |   \---ADC_Data_Read
    |   |       +---App
    |   |       |       main.c
    |   |       +---IAR
    |   |       |       ADC_Data_Read.ewd
    |   |       |       ADC_Data_Read.ewp
    |   |       |       ADC_Data_Read.eww
    |   |       \---KEIL
    |   |           ADC_Data_Read.uvopt
    |   |           ADC_Data_Read.uvproj
    |   \---Workspace
    |       +---IAR
    |       |       Examples_for_M365_Driver.eww
    |       \---KEIL
    |           Examples_for_M365_Driver.uvmpw
\---Template
    +---IAR
    |       TPM365_flash.icf
    \---KEIL
        tmpm365.sct
```



## 7-1 ADC

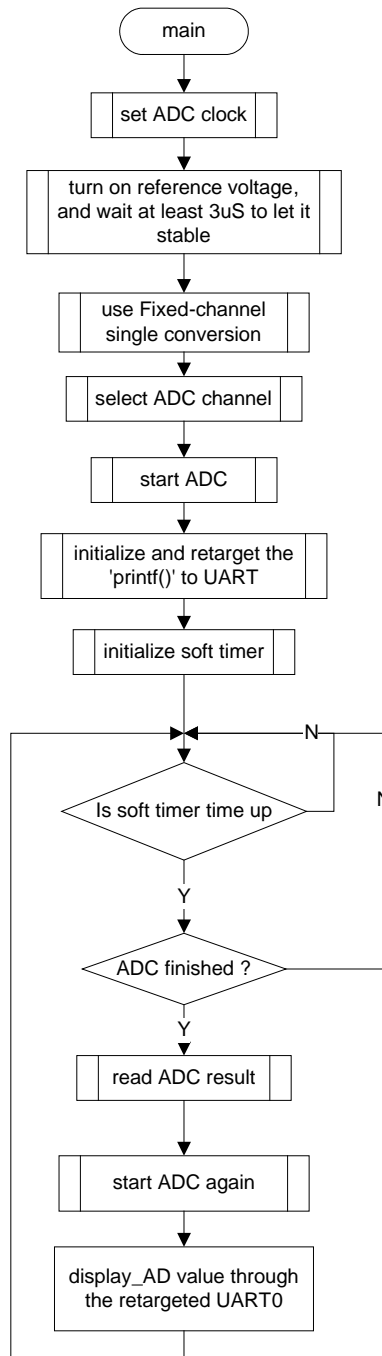
### 7-1-1 例: ADC データリード

ペリフェラルドライバ(ADC, GPIO, UART)を使用し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. チャンネル固定リピートモードによる AD 変換を開始し、AD 変換結果を読み出します。

- 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、VREF を ON します。

```
/* 1. set ADC clock */
ADC_SetClk(ADC_CONVERSION_81_CLOCK,
            ADC_FC_DIVIDE_LEVEL_16);

/* 2. turn on reference voltage, and wait at least 3uS to let it stable */
ADC_SetVref(ENABLE);
timeUp = 300U;
```

```
while( timeUp ) {  
    timeUp -- ;  
}
```

その後、チャンネル固定シングル変換の設定と AD チャンネルを選択します。

```
/* 3. use Fixed-channel single conversion */  
ADC_SetScanMode(DISABLE);  
ADC_SetRepeatMode(DISABLE);  
  
/* 4. select ADC channel */  
ADC_SetInputChannel(ADC_CHANNEL);
```

AD 変換を開始します。

```
/* 5. now start ADC */  
ADC_Start();
```

標準関数の“printf()”を使い、ソフトウェアタイマの初期設定を行います。

```
/* initialize and retarget the 'printf()' to UART */  
Retarget_Init();  
  
/* initialize soft timer */  
softTimer_Init();
```

AD 変換開始後、AD 変換状態を確認します。AD 変換後、ADC\_Display()をコールして別の AD 変換を開始します。

```
while (1U) {  
    softTimer_Run();  
    if(softT.flag_TimeUp) {  
        softT.flag_TimeUp = 0U;  
  
        /* check ADC module state */  
        adcState = ADC_GetConvertState();  
        if (adcState.Bit.NormalComplete ) {  
            /* read ADC result when it is finished */  
            adResult = ADC_GetConvertResult(ADC_REG_RESULT);  
            myResult = (uint32_t)adResult.Bit.ADResult ;  
            ADC_Start();  
            display_AD(myResult);  
        }  
    } else {  
        /* Do nothing */  
    }  
}
```

display\_AD()関数では、標準関数の“printf()”を使用し、AD 変換結果を PC へ送ります。

```
printf("ADC Value is %4d\n", myResult);
```

## 7-2 CG

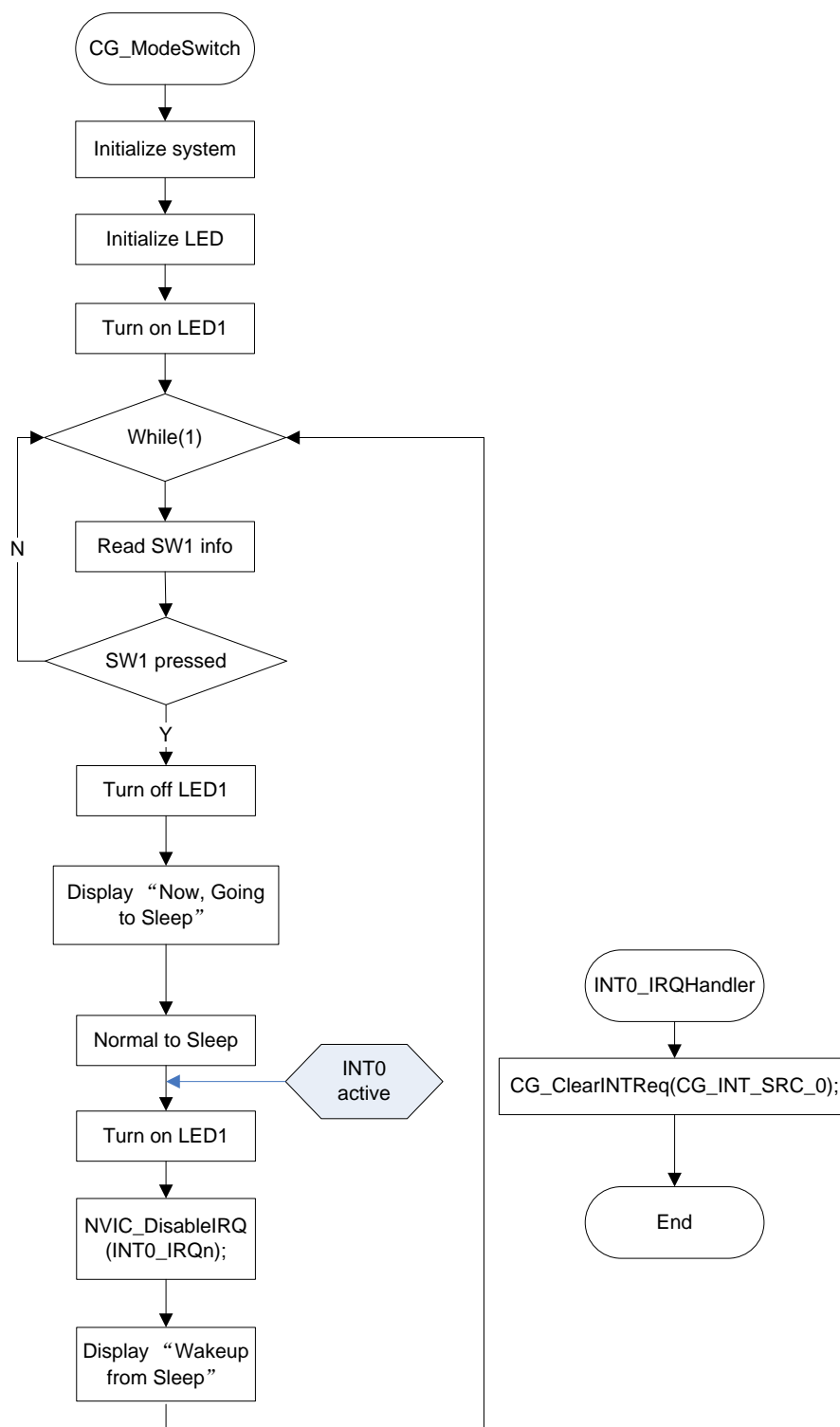
### 7-2-1 例: Power モード変更

ペリフェラルドライバ(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. 基本的な CG 動作の設定
2. NORMAL モードと STOP1 モードの切り替え方法
3. マルチクロック回路の許可方法

- フローチャート:



- サンプルプログラムのコードと説明

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。

```
/* Set SCOUT source */
CG_SetSCOUTSrc(CG_SCOUT_SRC_PHIT0);
if (CG_GetFoscSrc()==CG_FOSC_OSC_INT){
    /* Switch over from IHOSC to EHOSC*/
    switchFromIHOSCtoEHOSC();
}
/* Set up pll and wait 200us for pll to warm up , set fc source to fpll */
CG_EnableClkMulCircuit();
/* Set fgear = fc/2 */
CG_SetFgearLevel(CG_DIVIDE_2);
/* Set fperiph to fgear */
CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
CG_SetPhiT0Level(CG_DIVIDE_64);
/* Set low power consumption mode stop1 */
CG_SetSTBYMode(CG_STBY_MODE_STOP1);
/* Set pin status in stop mode to "active" */
CG_SetPinStateInStop1Mode(ENABLE);
```

## STOP1 モードへの遷移設定

STOP1 モードへの遷移設定を行います。ウォームアップ時間を設定します。  
解除要因として INT0 の設定を行います。INT0 割り込みを許可し、割り込み要求をクリアします。最後に \_\_WFI() 命令を実行して STOP1 モードへ遷移します。

```
/* Set CG module: Normal ->STOP1 mode */
/* after exiting STOP mode, internal oscillator starts up */
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT, CG_WUODR_INT);
/* Set RMC wake up system */
CG_SetSTBYReleaseINTSrc(CG_INT_SRC_0, CG_INT_ACTIVE_STATE_H,
ENABLE);

/* Disable interrupts */
__disable_irq();
NVIC_ClearPendingIRQ(INT0_IRQn);
NVIC_EnableIRQ(INT0_IRQn);
CG_ClearINTReq(CG_INT_SRC_0);
__enable_irq();

__DSB();
/* Enter stop1 mode */
__WFI();
```

## マルチクロック回路の許可

PLL を設定し、fc ソースを設定します。まず、PLL 値をセットし、PLL を許可します。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。

```
Result retval = ERROR;
WorkState st = BUSY;
CG_SetPLL(DISABLE);
CG_SetFPLLValue(CG_FPLL_MULTIPLY_8);
retval = CG_SetPLL(ENABLE);
if (retval == SUCCESS) {
    /* Set warm up time */
    CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT, CG_WUODR_PLL);
    CG_StartWarmUp();

    do {
        st = CG_GetWarmUpState();
```

```
    } while (st != DONE);  
  
    retval = CG_SetFcSrc(CG_FC_SRC_HALF_FPLL);  
} else {  
    /*Do nothing */  
}  
  
return retval;
```

## 7-3 DMAC

### 7-3-1 例: メモリから周辺回路

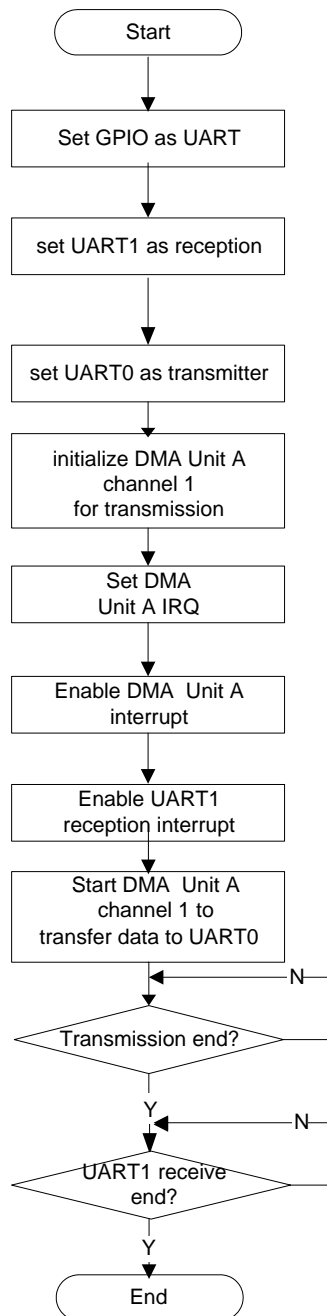
ペリフェラルドライバ (DMAC, GPIO, SIO) を使用したサンプルプログラムです。

以下の例が含まれます。

1. UART0/1 の初期化と DMAC の初期化を行います。
2. メモリから DMAC 経由で UART0 へデータを転送します。



- フローチャート:



- サンプルプログラムのコードと説明

以下は DMAC のドライバを使用して、メモリから UART0 へデータ転送を行うサンプルプログラムです。

まず、データ転送のために UART0 の設定と許可を行います。

```
UART_InitStruct.Mode = UART_ENABLE_TX;  
UART_Enable(UART0);  
UART_Init(UART0, &UART_InitStruct);  
UART_SetTxDMAReq(UART0, ENABLE);
```

データ転送のために DMAC ユニット A のチャンネル 1 の初期化と設定を行います。

```
DMAC_InitStruct.SrcAddr = (uint32_t) SRC_Buffer;
DMAC_InitStruct.SrcIncrementState = ENABLE;
DMAC_InitStruct.SrcBitWidth = DMAC_BYTE;
DMAC_InitStruct.SrcBurstSize = DMAC_1_BEAT;
DMAC_InitStruct.DstAddr = (uint32_t) (UART0_BASE_ADDR + UART0_BUF_OFFSET);
DMAC_InitStruct.DstIncrementState = DISABLE;
DMAC_InitStruct.DstBitWidth = DMAC_BYTE;
DMAC_InitStruct.DstBurstSize = DMAC_1_BEAT;
DMAC_InitStruct.TxSize = size;
DMAC_InitStruct.TxDirection = DMAC_MEMORY_TO_PERIPH;
DMAC_InitStruct.A_TxDstPeriph = DMACA_SIO0_UART0_TX;
DMAC_InitStruct.TxINT = ENABLE;

DMAC_Init(DMAC_UNIT_A, myChannel, &DMAC_InitStruct);
```

DMAC ユニット A の割り込みを許可します。

```
NVIC_ClearPendingIRQ(INTDMAC0TC_IRQn);
NVIC_EnableIRQ(INTDMAC0TC_IRQn);
```

DMAC ユニット A の許可、送信終了割り込みの設定、UART0 へのバースト転送設定を行い、DMAC ユニット A のチャンネル 1 から転送を開始します。

```
DMAC_Enable(DMAC_UNIT_A);
DMAC_SetTxINTConfig(DMAC_UNIT_A, myChannel, DMAC_INT_TX_END, ENABLE);
DMACA_SetSWBurstReq(DMACA_SIO0_UART0_TX);
DMAC_SetDMACChannel(DMAC_UNIT_A, myChannel, ENABLE);
```

DMAC 転送が終わるまで待ちます。

```
while (TxEndFlag != DONE) {
    /* Do nothing */
}
```

DMAC 転送終了時に ISR の DMAC ユニット A にフラグがセットされます。

```
state = DMAC_GetINTReq(DMAC_UNIT_A);
if (state.Bit.CH1_INTReq == 1U) {
    tmp = DMAC_GetTxINTReq(DMAC_UNIT_A, DMAC_CHANNEL_1);
    if (tmp == DMAC_TX_END_REQ) {
        TxEndFlag = DMAC_GetChannelTxState(DMAC_UNIT_A, DMAC_CHANNEL_1);
        DMAC_ClearTxINTReq(DMAC_UNIT_A, DMAC_CHANNEL_1, DMAC_INT_TX_END);
    } else {
        /* Do nothing */
    }
} else {
    /* Do nothing */
}
```

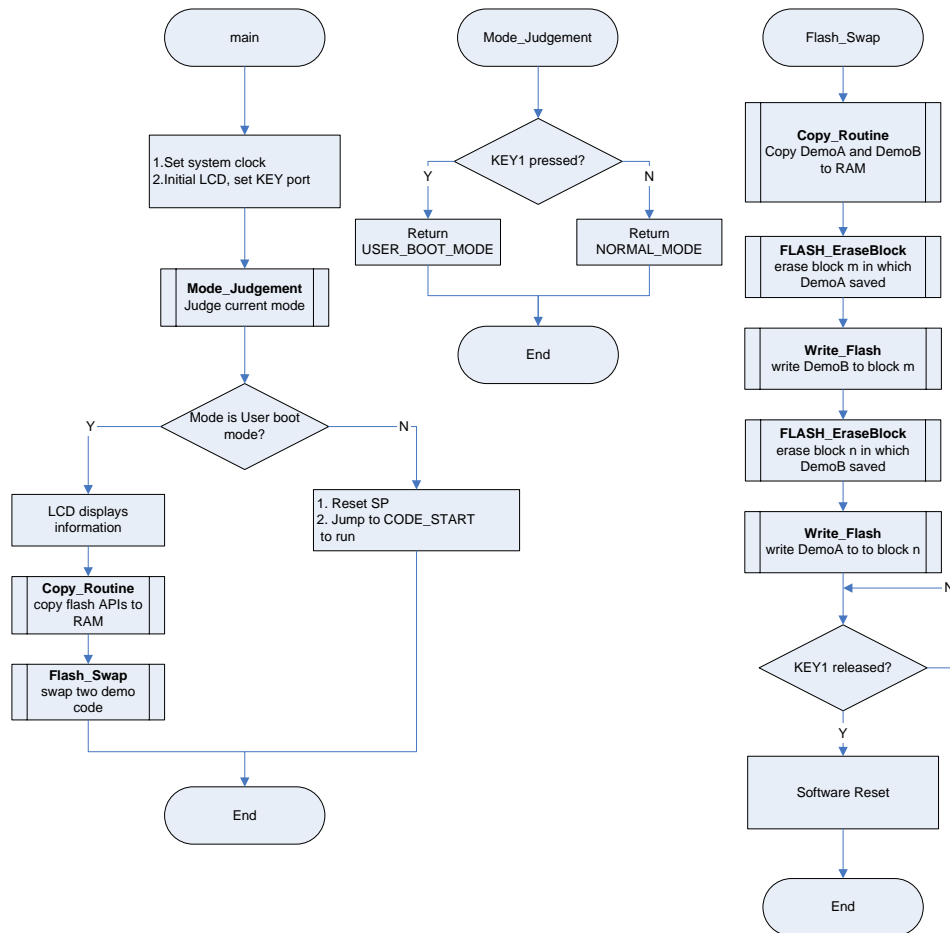
## 7-4 FLASH

ペリフェラルドライバ(FLASH, GPIO, CG)を使用したサンプルプログラムです。

以下の例が含まれています。

1. FLASH メモリのオンボードプログラミング (書き込み/消去)
2. 動作モード: シングルチップモード (ノーマルモード、ユーザブートモード)
3. ユーザブートモードを使用し、Flash メモリのコードを更新。

• 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SWを初期化します。リセット時に現在のモード判定をSWで、現在の状態表示をLEDで行います。

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

リセット後にどのモードになっているかの判断を行うために、Mode\_Judgement()関数をコールします。

```
uint8_t Mode_Judgement(void)
{
    return (SW_Get(SW0) == 1U) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

リセット時にSWが離されている場合、ノーマルモードになります。SPをリセットし、“CODE\_START”にジャンプします。プログラムAはブロックm内の“CODE\_START”に保存されているため、プログラムAが動作します(LEDが点滅)。

```
#if defined ( __CC_ARM ) /* RealView Compiler */
```

```
ResetSP(); /* reset SP */
#elif defined ( __ICCARM__ ) /* IAR Compiler */
asm("MOV R0, #0"); /* reset SP */
asm("LDR SP, [r0]");
#endif
SCB->VTOR = DEMO_START_ADDR; /* redirect vector table */
startup = CODE_START;
startup(); /* jump to code start address to run */
```

リセット時に SW0 が押されている場合、ユーザブートモードになります(LED3 が点灯)。Flash メモリは自分自身で消去/書き込みを実行できないため、Flash 動作用 API を、Flash メモリ内のアドレス “FLASH\_API\_ROM” から RAM の “FLASH\_API\_RAM” にコピーします。

```
Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
/* copy flash API to RAM */
```

Flash 動作用 API を RAM にコピー後、ルーチンプログラムは Flash\_Swap()関数にジャンプします。この関数は、上記の Copy\_Routine() を使用し、Flash メモリから RAM にコピーされます。

Flash\_Swap() 関数では、まずサンプル A、サンプル B を Flash メモリから RAM にコピーします。次に、Flash メモリの消去/書き込みを行うため、関数 FC\_EraseBlock () と Write\_Flash() を呼び出します。サンプル A とサンプル B の各プログラムは Flash メモリ内にて差し替えられます。

```
Copy_Routine(DEMO_A_RAM, DEMO_A_FLASH, SIZE_DEMO_A); /*
copy A to RAM */
Copy_Routine(DEMO_B_RAM, DEMO_B_FLASH, SIZE_DEMO_B); /*
copy B to RAM */

if (FC_SUCCESS == FC_EraseBlock((uint32_t) DEMO_A_FLASH)) { /* erase A */
    /* Do nothing */
} else {
    return ERROR;
}

if (FC_SUCCESS == Write_Flash(DEMO_A_FLASH, DEMO_B_RAM,
SIZE_DEMO_B)) { /* write B to A */
    /* Do nothing */
} else {
    return ERROR;
}

if (FC_SUCCESS == FC_EraseBlock((uint32_t) DEMO_B_FLASH)) { /* erase B */
    /* Do nothing */
} else {
    return ERROR;
}

if (FC_SUCCESS == Write_Flash(DEMO_B_FLASH, DEMO_A_RAM,
SIZE_DEMO_A)) { /* write A to B */
    /* Do nothing */
} else {
    return ERROR;
}
```

SW が離されると、SCB->AIRC\_R レジスタを用いてソフトリセットを行います。その後、ノーマルモードになります。サンプル A とサンプル B が差し替えられているため、アドレス

“CODE\_START” はサンプル B のスタートアドレスになり、サンプル B が動作します(LED1 が点滅)。

```
while (SW_Get(SW0) == 1U) {  
    }  
  
    NVIC_SystemReset();    /* software reset */
```

Flash メモリ動作関数 FC\_EraseBlock() は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 “block\_addr” で指定します。まず、この関数で引数 “Block\_addr” を確認します。次に、Flash ドライバ 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。プロテクトされていない場合は、ブロック消去コマンドにて、ブロックを消去します。

```
*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、消去が完了すると、Flash ドライバ FC\_GetBusyState() を用いビジーチェックを行います。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 FLASH\_WritePage() を呼び出し、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 FLASH\_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++;  
}
```

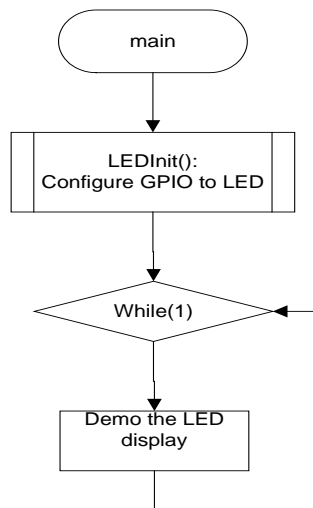
## 7-5 GPIO

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。

この例は以下を行います。

1. GPIO の初期化
2. GPIO へのデータ書き込み

- 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\_SetOutput()関数を用いて GPIO を LED に設定します。以下は GPIO\_WriteData()関数を用いて LED を OFF する例です。

```
GPIO_SetOutput(LED_DATA_PORT,LED_ALL);  
GPIO_WriteData(LED_DATA_PORT,led_off);
```

サンプルでは、while ループにて LED を順番に On します。

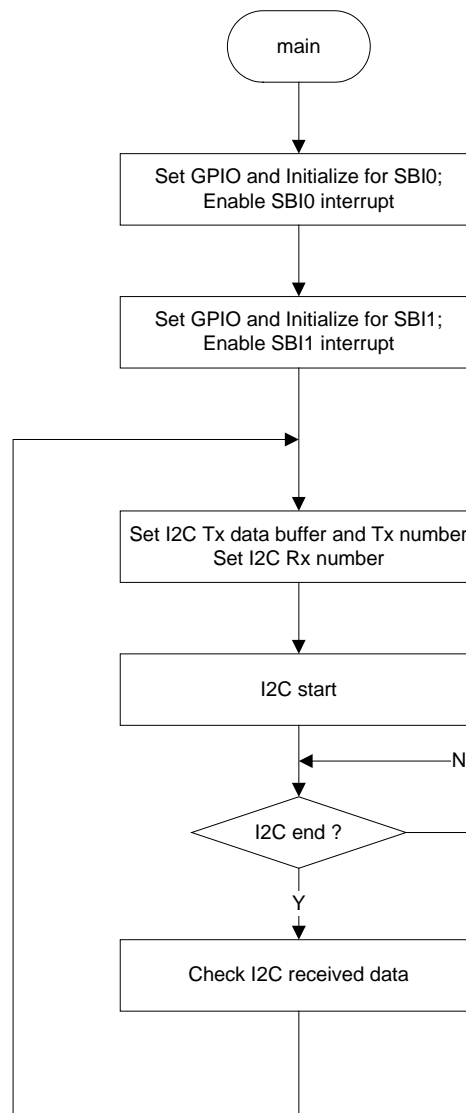
## 7-6 SBI

ペリフェラルドライバ(SBI, GPIO)を使用したサンプルプログラムです。

以下の例が含まれます。

1. SBI 設定、I2C 初期化
2. I2C マスタによるデータプロセスの送信
3. I2C スレーブによるデータプロセスの受信

- 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 SBI0 と SBI1 に設定します。

```
SBI0_IO_Configuration();  
SBI1_IO_Configuration();
```

次に、SBI0 をイネーブルし、初期化します。その後 INTSBI0 をイネーブルにします。

```
myI2C.I2CSelfAddr = SELF_ADDR;  
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;  
myI2C.I2CACKState = ENABLE;  
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;  
SBI_Enable(TSB_SBI0);  
SBI_SWReset(TSB_SBI0);  
SBI_InitI2C(TSB_SBI0, &myI2C);  
NVIC_EnableIRQ(INTSBI0_IRQn);
```

そして SBI1 をイネーブルし、初期化します。その後 INTSBI1 をイネーブルにします。

```
myI2C.I2CSelfAddr = SLAVE_ADDR;  
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;  
myI2C.I2CACKState = ENABLE;  
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
```

```
SBI_Enable(TSB_SBI1);
SBI_SWReset(TSB_SBI1);
SBI_InitI2C(TSB_SBI1, &myI2C);
NVIC_EnableIRQ(INTSBI1_IRQn);
```

上記設定を行った後、I2C 受信を開始します。

I2C 受信バッファをクリアし、SBI TX バッファとバッファ長を設定します。その後 RX バッファをクリアします。

```
/* Initialize TRx buffer and Tx length */
case MODE_SBI_I2C_INITIAL:
    gl2CTxDatLen = 7U;
    gl2CTxDat[0] = gl2CTxDatLen;
    gl2CTxDat[1] = 'T';
    gl2CTxDat[2] = 'O';
    gl2CTxDat[3] = 'S';
    gl2CTxDat[4] = 'H';
    gl2CTxDat[5] = 'I';
    gl2CTxDat[6] = 'B';
    gl2CTxDat[7] = 'A';
    gl2CWCnt = 0U;
    for (glCnt = 0U; glCnt < 8U; glCnt++) {
        gl2CRxDat[glCnt] = 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\_SBI0) 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_SBI0);
    if (!i2c_state.Bit.BusState) {
        SBI_SetSendData(TSB_SBI0, SLAVE_ADDR | SBI_I2C_SEND);
        SBI_GenerateI2CStart(TSB_SBI0);
        gSBIMode = MODE_SBI_I2C_TRX;
    } else {
        /* Do nothing */
    }
    break;
```

INTSBI0 でデータ転送を行います。

INTSBI1 でデータ受信を行います。

INTSBI0 ハンドラ内で、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、I2C\_SetSendData() で次のデータを送信し、I2C プロセス終了時には、I2C\_GenerateStop() で I2C を停止します。

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

    SBIX = TSB_SBI0;
    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 (gl2CWCnt <= gl2CTxDatLen) {
                SBI_SetSendData(SBIx, gl2CTxDat[gl2CWCnt]);
                /* Send next data */
                gl2CWCnt++;
            } else { /* I2C data send finished. */
                SBI_GenerateI2CStop(SBIx); /* Stop I2C */
            }
        }
    } else { /* Rx Mode */
        /* Do nothing */
    }
} else { /* Slave mode */
    /* Do nothing */
}
}
```

INTSBI1 ハンドラで、I2C バス状態を取得し、その値で I2C スレーブ受信プロセスを決定します。SBI バッファの受信データ読み出しは I2C\_GetReceiveData() 関数を用いて行います。I2C 停止状態はマスタによって制御します。

```
void INTSBI1_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI_TypeDef *SBly;
    SBI_I2CState sbi1_sr;

    SBly = TSB_SBI1;
    sbi1_sr = SBI_GetI2CState(SBly);

    if (!sbi1_sr.Bit.MasterSlave) { /* Slave mode */
        if (!sbi1_sr.Bit.TRx) { /* Rx Mode */
            if (sbi1_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = SBI_GetReceiveData(SBly);
                gl2CRCnt = 0U;
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = SBI_GetReceiveData(SBly);
                gl2CRxDat[gl2CRCnt] = tmp;
                gl2CRCnt++;
            }
        } else { /* Tx Mode */
            /* Do nothing */
        }
    } else { /* Master mode */
        /* Do nothing */
    }
}
```

## 7-7 TMRB

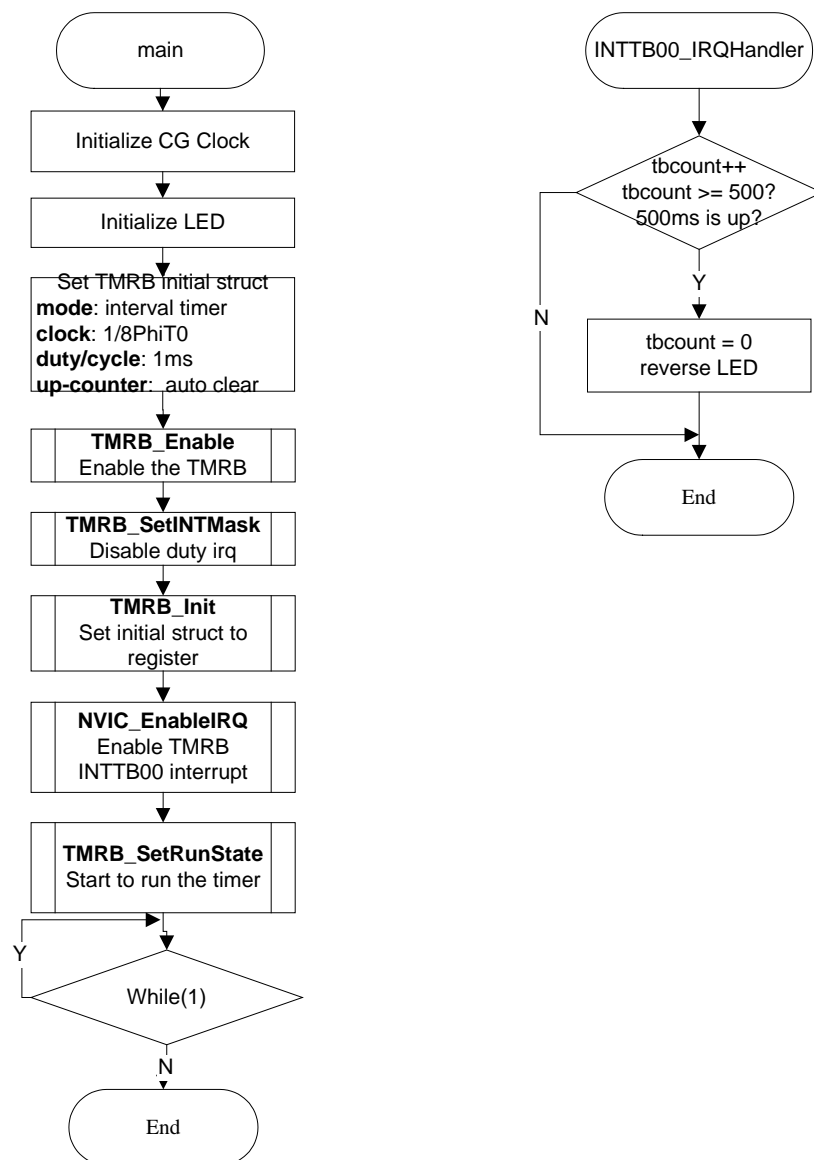
### 7-7-1 例: 汎用タイマ

ペリフェラルドライバ(TMRB, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. TMRB0 の初期化
2. 1ms の汎用タイマ

#### • Flow Chart



## ● サンプルプログラムのコードと説明

最初に LED を初期化し、LED を On します。

```
LED_Init(); /* LED initialize */
LedDisable(LED0 | LED1 | LED2 | LED3);
```

TMRB 設定用の構造体を用意し TMRB モード、クロック、アップカウンタクリア方法、周期とデューティを設定します。このデモでは、1ms の周期とデューティを設定します。TMRB\_1MS マクロは、0x1770 です。(φT0=fsys=12MHz \* PLL\* = 48MHz, ftmrB = 1/8φT0 = 6MHz, Ttmrb = 0.167us, 1ms/0.167us = 6000 = 0x1770 (クロック設定についての詳細は CG 章を参照してください))

```
TMRB_InitTypeDef m_tmrb;
m_tmrb.Mode = TMRB_INTERVAL_TIMER; /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8; /* 1/8PhiT0 */
m_tmrb.TrailingTiming = TMRB_1MS; /* periodic time is 1ms */
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR; /* up-counter auto clear */
m_tmrb.LeadingTiming = TMRB_1MS; /* periodic time is 1ms */
```

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_SetINTMask(TSB_TB0,TMRB_MASK_MATCH_LEADINGTIMING_INT);/*leading timing is not used in TMRB_INTERVAL_TIMER mode */
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(LED0);
    } else {
        Led_On(LED0);
    }
} else {
    /* Do nothing */
}
```

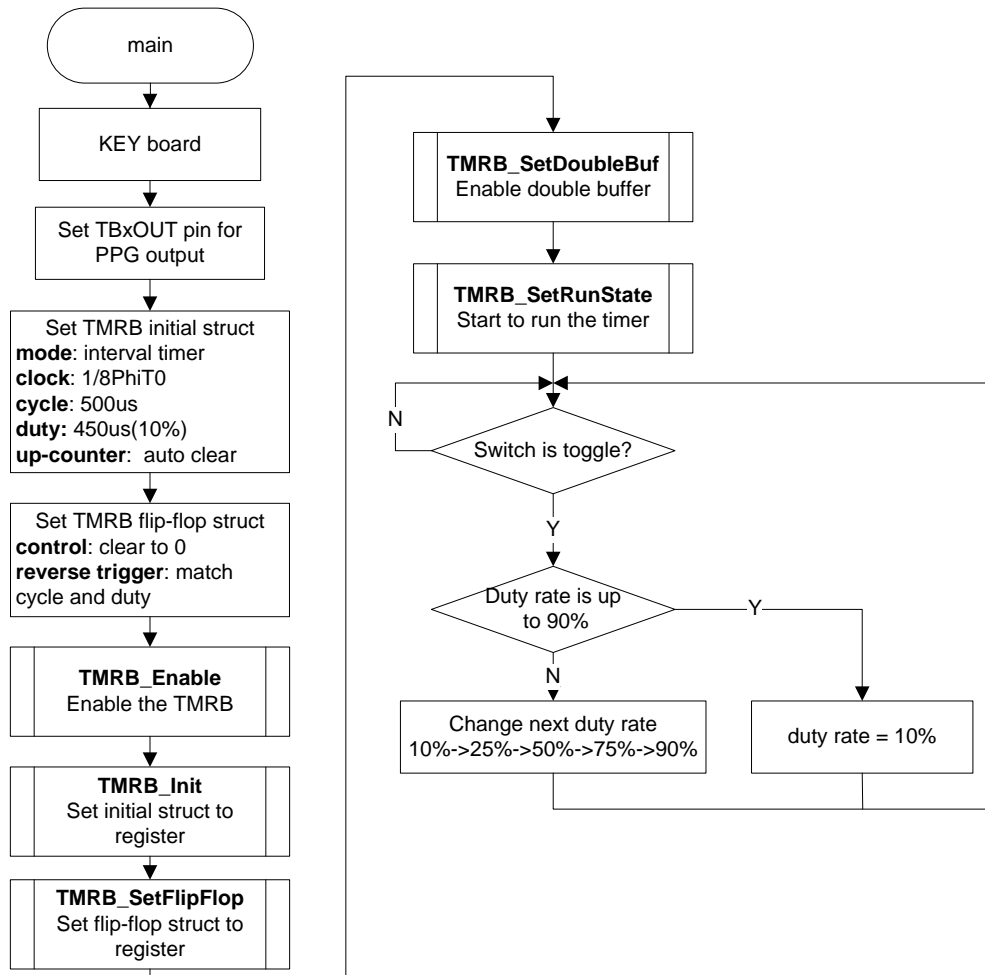
## 7-7-2 例: PPG 出力

ペリフェラルドライバ (TMRB, GPIO) を使用したサンプルプログラムです。

以下の例が含まれます。

1. TMRB4 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 GPIO を SW に設定し、PPG 出力用に PH2 を TB4OUT に設定します。

```

GPIO_SetInput(KEYPORT, GPIO_BIT_1); /* set KEY port to input */

/* Set PH2 as TB4OUT for PPG output */
GPIO_SetOutput(GPIO_PH, GPIO_BIT_2);
GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_3, GPIO_BIT_2);
    
```

TMRB の初期化用構造体を準備し、TMRB モード、クロック、アップカウンタクリア設定、サイクル、デューティを設定します。この例では 500us のサイクルを設定するため、TMRB4TIME マクロを定義してあります。このマクロは 0x0BB8 です。(φT0 = fsys = fc = 12MHz \* PLL \* 1/2 = 48MHz, ftmrB = 1/8 φT0 = 6MHz, TtmrB = 0.167us, 500us/0.167us = 3000 = 0x0BB8 (クロック設定の詳細は CG 章を参照してください))

```

TMRB_InitTypeDef m_tmrB;
    
```

```

m_tmrb.Mode = TMRB_INTERVAL_TIMER;          /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;              /* 1/8PhiT0 */
m_tmrb.TrailingTiming = TMRB4TIME;           /* trailing timing is 500us */
*/
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;          /* up-counter auto clear */
m_tmrb.LeadingTiming = LeadingTiming[Rate];  /* leading
timing, initial value 10% */

```

フリップフ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_CLEAR;
PPGFFInital.FlipflopReverseTrg=TMRB_FLIPFLOP_MATCH_TRAILINGTIMING |
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_TB4);
TMRB_Init(TSB_TB4, &m_tmrb);
TMRB_SetFlipFlop(TSB_TB4, &PPGFFInital);
TMRB_SetDoubleBuf(TSB_TB4, ENABLE); /* enable double buffer */
TMRB_SetRunState(TSB_TB4, TMRB_RUN);

```

スイッチ状態の変化を待ちます。

```

do {
    /* wait if switch is Low */
    keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_1);
    } while (GPIO_BIT_VALUE_0 == keyvalue);
delay(0xFFFFU); /* noise cancel */

```

スイッチ状態が変化すると、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。

```

do {
    keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_1);
    } while (GPIO_BIT_VALUE_1 == keyvalue);
delay(0xFFFFU); /* noise cancel */

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

TMRB_ChangeLeadingTiming(TSB_TB4, LeadingTiming[Rate]); /* switch is
High again */

```

デューティの算出方法:

Trailing Timing = 500us, ftmrb = 1/8 fphiT0 = 6MHz, Ttmrb = 0.167us (これらの時間に関するパラメータは、CG 設定により異なります)

Leading Timing = 10%: High 幅は 500\*10% = 50us, Low 幅は 500-50 = 450us, カウンタ

値 = 450us/Ttmrb = 0xA8C

LeadingTiming = 25%: High 幅は  $500 \times 25\% = 150\text{us}$ , Low 幅は  $500 - 125 = 375\text{us}$ , カウンタ値 =  $375\text{us}/\text{Ttmrb} = 0x8\text{CAU}$

LeadingTiming = 50%: High 幅は  $500 \times 50\% = 250\text{us}$ , Low 幅は  $500 - 250 = 250\text{us}$ , カウンタ値 =  $250\text{us}/\text{Ttmrb} = 0x5\text{DCU}$

Duty = 75%: High 幅は  $500 \times 75\% = 375\text{us}$ , Low 幅は  $500 - 375 = 125\text{us}$ , カウンタ値 =  $125\text{us}/\text{Ttmrb} = 0x2\text{EEU}$

Duty = 90%: High 幅は  $500 \times 90\% = 450\text{us}$ , Low 幅は  $500 - 450 = 50\text{us}$ , カウンタ値 =  $50\text{us}/\text{Ttmrb} = 0x1\text{FAU}$

これは上記計算式から求めたデューティ値の配列です。

```
uint32_t LeadingTiming[5] = { 0xA8CU, 0x8CAU, 0x5DCU, 0x2EEU, 0x1FAU };  
/* leading timing: 10%, 25%, 50%, 75%, 90% */
```

## 7-8 SIO/UART

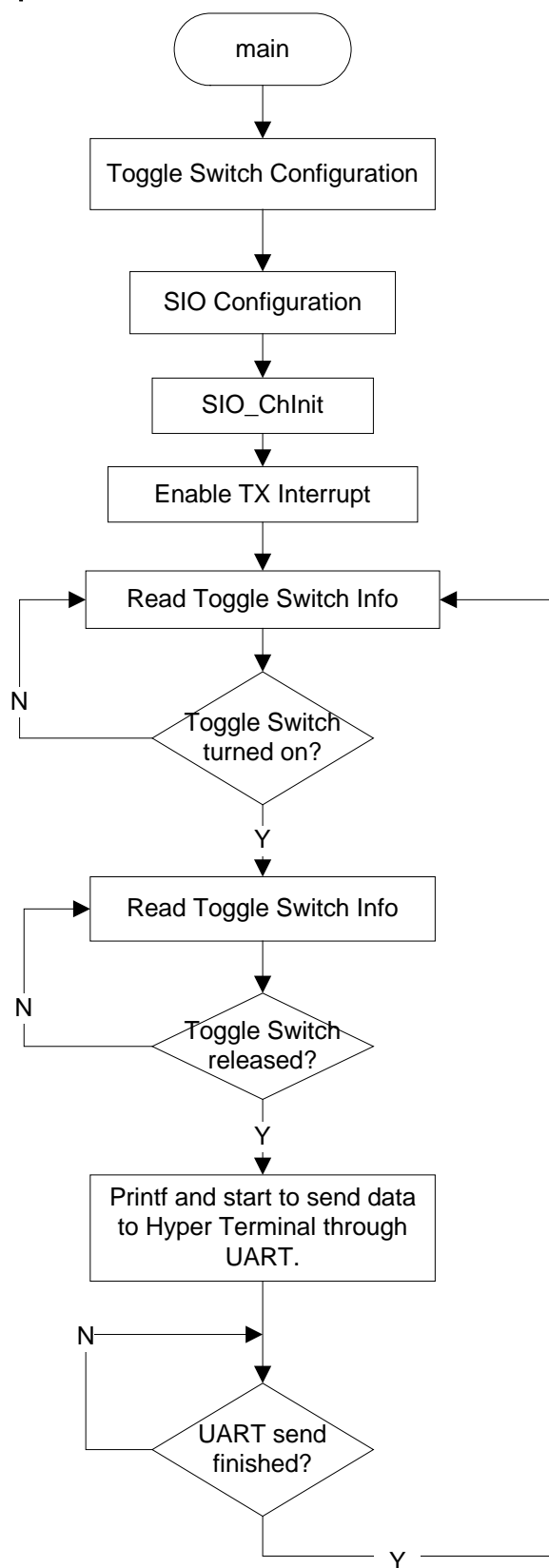
### 7-8-1 例: リターゲット

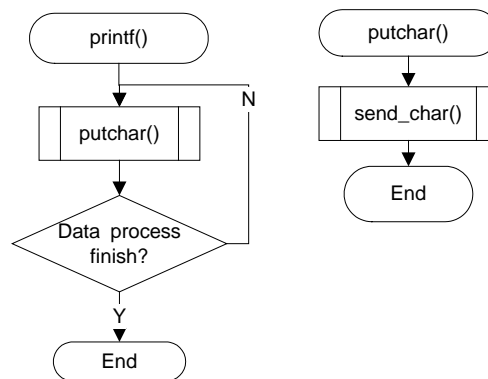
ペリフェラルドライバ (UART, GPIO)を用いたサンプルプログラムです。

この例では以下を行います。

1. UART 設定と初期化
2. UART 送信制御
3. データ送信に UART1 の TX 割り込みを使用
4. UART1 に printf()関数をリターゲット

- フローチャート:





## ● サンプルプログラムのコードと説明

GPIO を UART に設定します。

```

GPIO_SetOutputEnableReg(GPIO_PC, GPIO_BIT_0, ENABLE);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_1, GPIO_BIT_0);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_1, GPIO_BIT_1);
GPIO_SetInputEnableReg(GPIO_PC, GPIO_BIT_1, ENABLE);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_0, ENABLE);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_1, ENABLE);
  
```

UART\_InitTypeDef 構造体を準備し、データを設定後、UART を初期化します。

以下は設定例です。

```

UART_InitTypeDef myUART;

/* configure SIO1 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)
  
```

UART 経由で printf()にてデータを送信します。

```

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

最後に UART1 送信割り込み処理ルーチンを準備します。

以下は UART1 の送信割り込み処理ルーチンです。

```

void INTTX1_IRQHandler(void)
{
    if (gSIORdIndex < gSIOWrIndex) { /* buffer is not empty */
        UART_SetTxData(UART_RETARGET, gSIOTxBuffer[gSIORdIndex++]);
    /* send data */
        fSIO_INT = SET; /* SIO1 INT is enable */
    } else {
        /* disable SIO1 INT */
        fSIO_INT = CLEAR;
        NVIC_DisableIRQ(RETARGET_INT);
        fSIOTxOK = YES;
    }
}
  
```



```
    }
    if (gSIORdIndex >= gSIOWrIndex) { /* reset buffer index */
        gSIOWrIndex = CLEAR;
        gSIORdIndex = CLEAR;
    } else {
        /* Do nothing */
    }
}
```

IAR コンパイラでは printf() 関数が putchar() 関数をコールし、RealView コンパイラでは fputc() 関数をコールし、UART ヘデータを出力します。

```
#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;
}
```

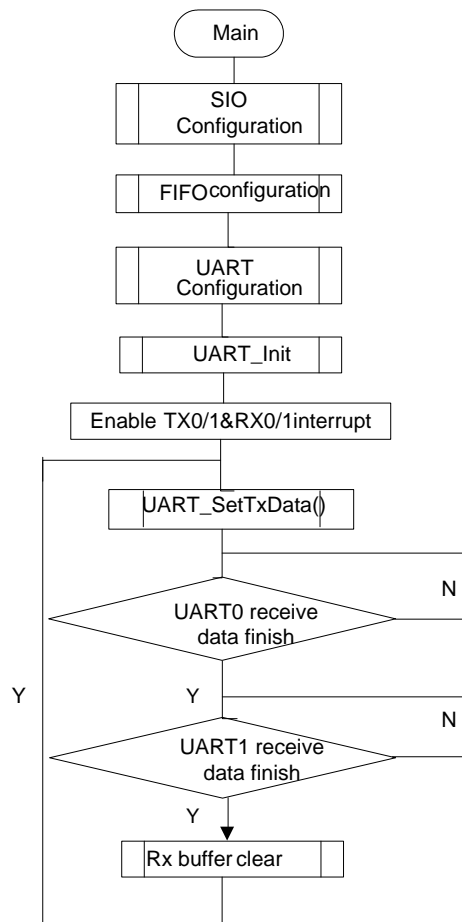
## 7-8-2 例: UART FIFO

ペリフェラルドライバ(UART, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

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_PC->CR |= GPIO_BIT_0;
        TSB_PC->FR1 |= GPIO_BIT_0;
        TSB_PC->FR1 |= GPIO_BIT_1;
        TSB_PC->IE |= GPIO_BIT_1;
    }
}
  
```

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);
    }
}
```

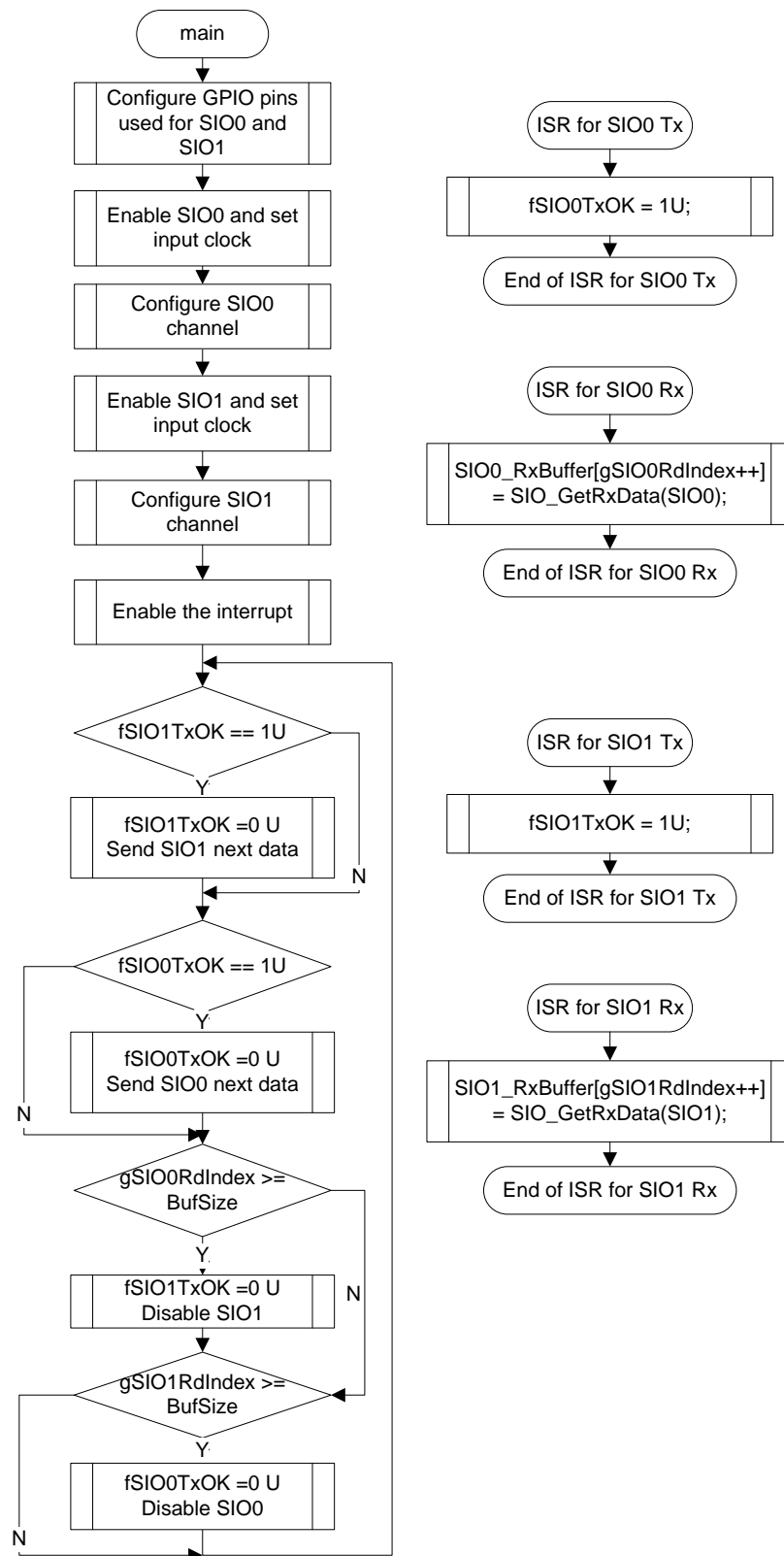
## 7-8-3 例: SIO

ペリフェラルドライバ(SIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. SIO 動作の基本設定
2. SIO0~SIO1 間のデータ転送
3. SIO の送受信割り込み

## • 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 端子を SIO に設定します。

その後、SIO0 を有効にし、入カクロックの設定と SIO0 の初期化を行います。

```
/*Enable the SIO0 channel */
SIO_Enable(SIO0);

/*initialize the SIO0 struct */
SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_8;
SIO0_Init.TransferMode = SIO_TRANSFER_FULDPX;
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_T4;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;
SIO_Init(SIO0, SIO_CLK_BAUDRATE, &SIO0_Init);
```

SIO1 を有効にし、入カクロックの設定と SIO1 の初期化を行います。

```
/*Enable the SIO1 channel */
SIO_Enable(SIO1);

/*initialize the SIO1 struct */
SIO1_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO1_Init.TransferMode = SIO_TRANSFER_FULDPX;
SIO1_Init.TransferDir = SIO_LSB_FRIST;
SIO1_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO1_Init.DoubleBuffer = SIO_WBUF_ENABLE;

SIO_Init(SIO1, SIO_CLK_SCLKINPUT, &SIO1_Init);
```

SIO の送信割り込みと受信割り込みを許可します。

```
/* Enable SIO0 Channel TX interrupt */
NVIC_EnableIRQ(INTTX0_IRQn);
/* Enable SIO1 Channel RX interrupt */
NVIC_EnableIRQ(INTRX1_IRQn);

/* Enable SIO1 Channel TX interrupt */
NVIC_EnableIRQ(INTTX1_IRQn);
/* Enable SIO0 Channel RX interrupt */
NVIC_EnableIRQ(INTRX0_IRQn);
```

すべての基本的な設定を行い、その後、データ転送処理に入ります。

```
while (1) {
    /* SIO1 send data from TXD1*/
    if (fSIO1TxOK == 1U) {
        fSIO1TxOK = 0U;
        SIO_SetTxData(SIO1, SIO1_TxBuffer[gSIO1WrIndex++]);
    } else {
        /*Do Nothing */
    }
    /* SIO0 send data from TXD0*/
    if (fSIO0TxOK == 1U) {
        fSIO0TxOK = 0U;
        SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
    } else {
        /*Do Nothing */
    }
}

/*SIO0 receive data end */
```

```
if (gSIO0RdIndex >= BufSize) {
    fSIO1TxOK = 0U;
    SIO_Disable(SIO1);
} else {
    /*Do Nothing */
}
/*SIO1 receive data end */
if (gSIO1RdIndex >= BufSize) {
    fSIO0TxOK = 0U;
    SIO_Disable(SIO0);
} else {
    /*Do Nothing */
} }
```

以下は SIO0 の送信割り込み処理ルーチンです。送信完了フラグをセットします。

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

以下は SIO0 の受信割り込み処理ルーチンです。受信バッファから受信データを取得します。

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

以下は SIO1 の送信割り込み処理ルーチンです。送信完了フラグをセットします。

```
void INTTX1_IRQHandler(void)
{
    fSIO1TxOK = 1U;
}
```

以下は SIO1 の受信割り込み処理ルーチンです。受信バッファから受信データを取得します。

```
void INTRX1_IRQHandler(void)
{
    SIO0_RxBuffer[gSIO1RdIndex++] = SIO_GetRxData(SIO1);
}
```

## 7-9 WDT

ペリフェラルドライバ(WDT, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

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 LED 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  $2^{25}/f_{sys}$  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)  
{  
    if (fIntNMI == 1U) {  
        fIntNMI = 0U;  
        /* Do something here */  
    }else {  
        /* Do nothing */  
    }  
}
```

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

```
WDT_Disable();
```

DEMO2 では、WDT クリアを行い、常に LED を点滅させます。

```
WDT_WriteClearCode();
```