TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM381/3)

第 0.102 版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M381UE-00xJ

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 2
5 開発環境 3
6 機能説明 4
6-1 動作モード選択 4
6-2 ADC 5
6-3 CG 5
6-3-1 動作モード変更 5
6-4 DNF 5
6-5 FLASH 6
6-6 FUART 6
6-7 GPIO 7
6-8 OFD 7
6-9 RMC 7
6-9-1 RMC 受信 7
6-10 RTC 8
6-11 SBI 8
6-11-1 SBI スレーブ 8
6-11-2 SBI マスタ 8
6-12 SSP 9
6-13 TMRB 9
6-13-1 汎用タイマ 9
6-13-2 PPG 波形出力 9
6-14 SIO/UART 9
6-14-1 SIO 9
6-14-2 リターゲット 10
6-14-3 UART FIFO 10
6-15 VLTD 10
6-15-1 VLTD Reset 10
6-16 WDT 10
7 ソフトウェア 11
7-1 ADC 12
7-1-1 例: ADC データリード 12
7-2 CG 14
7-2-1 例: 動作モードの変更 14
7-3 DNF 17
7-3-1 例: DNF 17
7-4 FLASH 19
7-4-1例: FLASH 書き込み 19

7-5 FUART 23
7-5-1 例: ループバック 23
7-6 GPIO 27
7-6-1 Example: GPIO Data Read 27
7-7 OFD 28
7-7-1 例: OFD 28
7-8 RMC 30
7-8-1 例: RMC 受信 30
7-9 RTC 32
7-10 SBI 35
7-10-1 例: SBI スレーブ 35
7-10-2 例: SBI マスタ 39
7-11 SSP 43
7-11-1 例: SSP0 セルフループバック 43
7-12 TMRB 46
7-12-1 例: 汎用タイマ 46
7-12-2 例: PPG 波形出力 48
7-13 SIO/UART 51
7-13-1 例: SIO 51
7-13-2 例: リターゲット 54
7-13-3 例: UART FIFO 57
7-14 VLTD 61
7-15 WDT 62

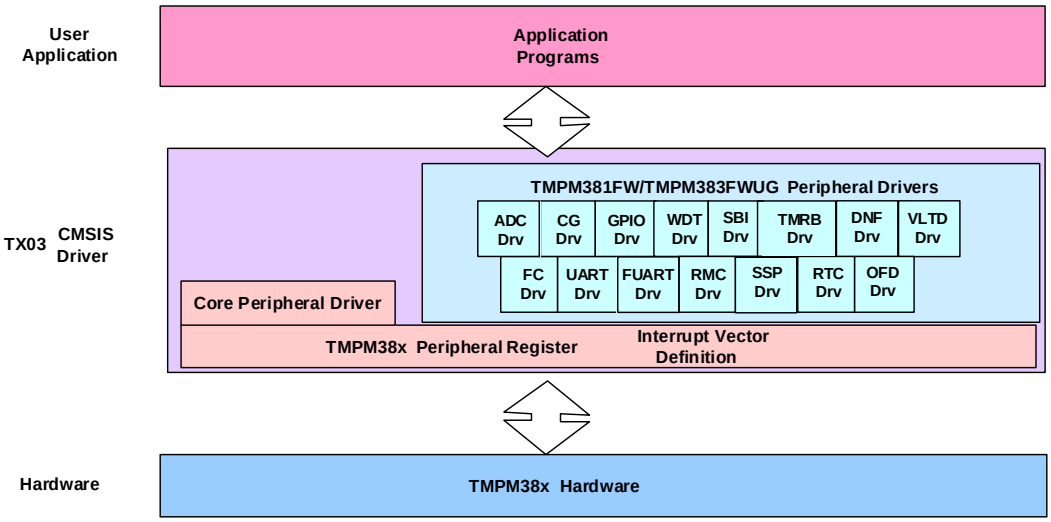
1 はしがき

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TMPM38x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。

注: 以降、TMPM38xはTMPM381/383を表します。

2 概要

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: CG サンプル
	PLL	PLL on (4 逓倍)
低消費電力モード	-	使用: CG サンプル(STOP モード)
ウォッチドッグタイマ (WDT)	-	使用: WDT サンプル
外部割り込み (INT)	INT0	使用: CG サンプル
	INT3	使用: DNF サンプル
	INTRTC	使用: RTC サンプル
	INTSBI0, INTSBI1	使用: SBI スレーブ(マスタ)サンプル / SIO サンプル

機能	チャネル	使用／未使用
	INTRX0	使用: UART SIO / FIFO サンプル
	INTRX1	使用: UART SIO / FIFO サンプル
	INTTX0	使用: UART SIO / Retarget / FIFO サンプル
	INTTX1	使用: UART SIO / FIFO サンプル
シリアルチャネル(SIO/UART)	SIO0	使用: UART retarget サンプル / SIO サンプル / UART FIFO サンプル
	SIO1	使用: SIO サンプル
16 ビットタイマ(TMRB)	TMRB0	使用: TMRB サンプル: 汎用タイマ
	TMRB6	使用: TMRB サンプル: PPG 出力
12 ビット A/D コンバータ	AIN0	使用: ADC データリード
RTC	-	使用: RTC サンプル
シリアルバスインタフェース(SBI)	SBI0	使用: スレーブ/マスタサンプル
同期式シリアルインタフェース(SSP)	SSP0	使用: SSP セルフループバックサンプル
非同期式シリアルインタフェース (UART)	-	使用: Full UART ループバックサンプル
リモコン判定機能 (RMC)	-	使用: RMC 信号受信
デジタルノイズフィルタ (DNF)	-	DNF サンプル
周波数検知回路 (OFD)	-	使用: OFD サンプル
電圧検出回路(VLTD)	-	使用: VLTD サンプル
Flash	-	使用: FC サンプル

4 端子用途

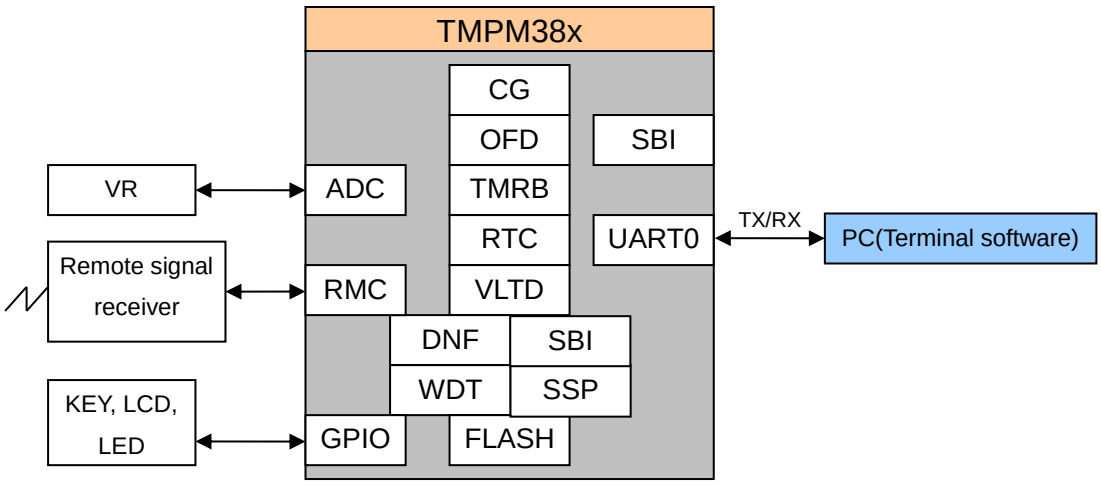
本サンプルプログラムは、TMPM38x評価ボードを用いてテストされています。以下に、端子用途を説明します。

Pin No.	端子名	用途
1	PH0	ADC AIN0 / CG INT0
2	PF1	DNF TB7OUT
3	PA0	DNF INT3/ LED0
4	PE0	UART TXD0 (M383)
5	PE1	UART RXD0 (M383)

Pin No.	端子名	用途
6	PC6	FUART UT0TXD50A
7	PC7	FUART UT0RXD50
8	PC5	FUART UT0TXD50B
9	PA5	TMRB TB6OUT UART TXD1 (M38x)
10	PA6	UART RXD1 (M38x)
11	PE2	UART SCLK0
12	PA4	UART SCLK0
13	PA1	LED1
14	PA2	LED2
15	PA3	LED3
16	PH4	SW0/SW4
17	PH5	SW1/SW5
20	PH6	SW2/SW6
21	PH7	SW3/SW7
22	X1	外部高速発振
23	X2	外部高速発振
24	XT1	外部低速発振
25	XT2	外部低速発振
26	MODE	MODE
27	RESET	リセット
28	BOOT	BOOT モード制御

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:

TMPM38x 評価ボード (非売品)
TMPM381-SK 評価ボード

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 7.30.4
   - KEIL 社:
     1) IDE: KEIL uVision MDK 5.14

6 機能説明

6-1 動作モード選択

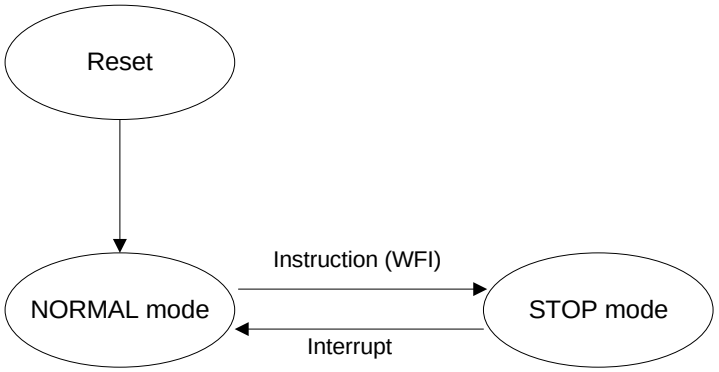
TMPM38x には 5 つの動作モードがあります: NORMAL, SLOW, IDLE, SLEEP, STOP モード

- **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、内蔵高速クロックによる NORMAL モードになります。
IDLE, SLEEP, STOP モードは低消費電力モードです。
低消費電力モードに移行するには、システム制御レジスタ CGSTBYCR0<STBY[2:0]>にて IDLE, SLEEP, STOP モードのいずれかを設定し、WFI (Wait For Interrupt)命令を実行します。
補足: サンプルプログラムは、STOP モードのみ使用します。

- **STOP モード:**

いくつかの周辺回路を除き、内蔵高速発振を含むすべての内蔵回路を停止させるモードです。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。
補足: CG サンプルプログラムにて STOP モードと NORMAL モードの遷移状態を確認することができます。



6-2 ADC

このサンプルプログラムは、AIN0 に接続された VR1 の値を変更するサンプルプログラムです。
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。

補足:
LED0~LED3をマイコンのPA0~PA3に接続します。

6-3 CG

6-3-1 動作モード変更

このサンプルプログラムは、NORMAL モードと STOP モードを切り替えることで 2 つの動作モードの動作状態を確認できます。動作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	SW4 を押す	NORMAL→ STOP	全 LED 消灯
STOP	SW4 を戻す	STOP →NORMAL	全 LED 点灯

補足:
LED0~LED3をマイコンのPA0~PA3に接続します。
SW0 をマイコンの PH0(INT0)と接続します。

6-4 DNF

DNF ドライバを使用したサンプルプログラムです。

DNF は任意の範囲での外部割り込み入力信号のノイズを除去することが可能です。TMRB を PPG 出力モードにし、任意の周期とデューティを持つ矩形波を出力します。
このサンプルプログラムでは、PPG 出力によるパルスを検知し、ノイズと判断すると割り込みを発生させます。

補足:
マイコンのPA0(INT3、DNF)とPF1(TB7OUT)を接続します。

6-5 FLASH

このサンプルプログラムは、Flash のドライバ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。
ーリセット動作: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、フラッシュ上のブロック 1 を書き換えます。

プログラムシーケンス

(1) 電源投入
Block0 のリセット処理プログラムが動作します。
SW0 を OFF した状態で電源投入またはリセットすると、LED1 が点滅します。

(2) SW0 を ON しながら電源投入またはリセットすると、書き換えルーチンが RAM へ転送されます。
まず block1 がイレースされ、その後 block1 の書き換えを行います。
Block1 のデータと書き込みデータを比較し、一致した場合は LED1 を点滅させ、LED2 を点灯します。

(3) 手順(1)に戻ります。

補足:
LED0~LED3をマイコンのPA0~PA3に接続します。
SW0 をマイコンの PH4 と接続します。

6-6 FUART

Full UART のドライバを使用して、Full UART 送信とFIFO 受信を行うサンプルプログラムです。
本プログラムは 64 種の異なるデータを Full UART チャネル 6 の TXD0 端子から送信し、RXD0 端子からデータを受信します。トグルスイッチ SW0 がオンの場合、本プログラムは受信 FIFO からのデータ読み出しを開始します。データの読み出しは受信 FIFO が空になるまで止まりません。
トグルスイッチ SW0 が数回オフ、オンとなった場合、本プログラムは RXD0 が受信した全データの読み出しを完了しています。その後プログラムは全受信データを送信データと比較します。
受信データが送信データと同一の場合、UART0 に"RX TX SAME"を表示します。
受信データが送信データと異なる場合、UART0 に"RX TX DIFF"を表示します。
本プログラムは、ハードウェアフロー制御機能の確認のため 2 回実行することができます。

補足:
マイコンのPC6(UT0TXD50A)とPC7(UT0RXD50)を接続します。

6-7 GPIO

このサンプルプログラムは、GPIO を LED に設定し、LED の点灯/消灯を行うサンプルプログラムです。

補足:
LED0~LED3をマイコンのPA0~PA3に接続します。
SW0~SW3 をマイコンの PH4~PH7 に接続します。

6-8 OFD

このサンプルプログラムは、クロックが OFD に設定された周波数範囲を超えると、OFD リセットを発行するサンプルプログラムです。

6-9 RMC

6-9-1 RMC 受信

このサンプルプログラムは、リモコン信号データを受信し、デコードするサンプルプログラムです。
またデコードしたデータをデバッガ上のターミナルウィンドウに表示します。
カスタムコードまたはアドレスコードとコマンドコードを Hex フォーマットで表示します。

表示フォーマット: TMPM38x RMC Demo
RMC_6: XX XX
RMC_6: YY YY

フォーマット	ターミナルウィンドウの表示
1	RMC_1:
2	RMC_2:
3	RMC_3:
4	RMC_4:
5	RMC_5:
6	RMC_6:

6-10 RTC

このサンプルプログラムは、RTC を使用して LCD に日時を表示するサンプルプログラムです。
初期設定では 2015/06/12, 時刻の設定が 13:54:00, 24 時間表示です。
更新間隔: 1 秒
LCD 表示:

```
2015/06/12
13:54:00
```

6-11 SBI

6-11-1 SBI スレーブ

このサンプルプログラムでは、I2C バスのスレーブモード動作を確認することができます。
2 つの評価ボードの I2C バスを接続します。
本サンプル側はスレーブとして動作します。

I2C バスのデータをリードするために SBI 割り込みを使用します。
ゼネラルコール、または SBII2CADR に設定されたスレーブアドレスと同一のアドレスを受信したとき、9 クロック目で SDA ラインを"Low"レベルに引き、アクノリッジ信号を出力します。

I2C スレーブ(SBI)は I2C マスタ(SBI)から文字列“TOSHIBA”を受信し、UART に出力します。
SBI サンプルプログラムを開始すると以下を表示します。

```
K1: SBI SLAVE RECV
```

SW4 を ON すると、SBI(スレーブ)の受信バッファから"TOSHIBA"をリードし、出力します。

補足:
SW4 と PH4 を接続します。

6-11-2 SBI マスタ

このサンプルプログラムでは、I2C バスのマスタモード動作を確認することができます。
2 つの評価ボードの I2C バスを接続します。
本サンプル側はマスタとして動作します。

I2C バスのデータをライトするために SBI 割り込みを使用します。
サンプルプログラムの実行を開始後、SW7 を ON すると、I2C マスタ(SBI)は I2C スレーブ(SBI)へ文字列“TOSHIBA”を送信します。

補足:

SW7 と PH を接続します。

6-12 SSP

データを送信し、その後、受信データを確認します。受信データが送信したものと同一の場合、LED2 と LED3 を点灯します。同一でない場合、LED0 と LED1を点灯します。受信データは UART に出力します。
UART 出力:

```
SSP RX DATA:
xxxxxx
```

補足: LED0~LED3 は PA0~PA3 と接続します。

6-13 TMRB

6-13-1 汎用タイマ

このサンプルプログラムは、MCU のタイマを使って、汎用タイマを実現するサンプルプログラムです。
タイマ設定は 1ms 周期です。
このタイマを使い 1s (500ms で ON、500ms で OFF) 間隔で LED 点滅を行います。

6-13-2 PPG 波形出力

このサンプルプログラムは、SW1 を使い、デューティ可変の PPG(プログラマブル矩形波)の波形を出力するサンプルプログラムです。PPG 波形は、PA5(TB6OUT)をオシロスコープに接続することで確認できます。
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 波形出力を開始します。
SW を押すことでデューティーを変更します。
10% → 25% → 50% → 75% → 90% → 10%

6-14 SIO/UART

6-14-1 SIO

このサンプルでは、SIO 機能を利用した同期式の送受信を確認できます。
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)

6-14-2　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。
このサンプルプログラムは、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うサンプルプログラムです。

6-14-3　UART FIFO

UART0 から" TMPM38x1"というデータを FIFO を使用して UART1 へ送信し、同時に UART1 から"TMPM38x2"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=” TMPM38x2”と RxBuffer1=" TMPM38x1"となった場合にブレークポイントで停止します。

6-15　VLTD

6-15-1　VLTD Reset

このサンプルプログラムでは、VLTDによる電圧状態の検出を確認することができます。
電圧が検出電圧より低い場合、自動的にリセットを行い、ソフトウェアによってPH0に"High"レベルを出力します。

6-16　WDT

このサンプルプログラムではウォッチドッグタイマの動作を確認することができます。ウォッチドッグタイマ (WDT) は、ノイズなどの原因により CPU が誤動作 (暴走) を始めた場合にこれを検出し、正常な状態に戻すことを目的としています。
WDT が暴走を検出すると、NMI 割り込みまたはリセットを発生させます。WDT は、高速クロックが停止する動作モード(STOP、SLEEP、SLOW)では使用できません。
リセット後ウォッチドッグタイマは有効となります。

サンプル例:

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。

7 ソフトウェア

このソフトウェアは、TMPM38x MCU の主要機能を TMPM38x 評価ボード上で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
        │   tmpm38x_wdt_int.c
        │   tmpm38x_wdt_int.h
        │
        ├─TX03_CMSIS
        │   │   system_TMPM38x.c
        │   │   system_TMPM38x.h
        │   │   TMPM38x.h
        │   └─startup
        │         startup_TMPM38x.s
        │
        ├─TX03_Periph_Driver
        │   ├─inc
        │   │   tmpm38x_gpio.h
        │   │   tmpm38x_wdt.h
        │   │   tmpm38x_cg.h
        │   │   tx03_common.h
        │   │
        │   └─src
        │         tmpm381_gpio.c
        │         tmpm381_wdt.c
        │         tmpm381_cg.c
        │
        └─TMPM38x-EVAL
                  led.c
                  led.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm38x_wdt_int.c
        │   tmpm38x_wdt_int.h
        │
        ├─TX03_CMSIS
        │    system_TMPM38x.c
        │    startup_TMPM381.s
        │
        ├─TX03_Periph_Driver
        │    tmpm381_gpio.c
        │    tmpm381_wdt.c
        │    tmpm381_cg.c
        │
        └─TMPM381-EVAL
             led.c
```

TMPM381-SK 評価ボード用のプロジェクトファイルは***-SK フォルダに格納されています。

7-1 ADC

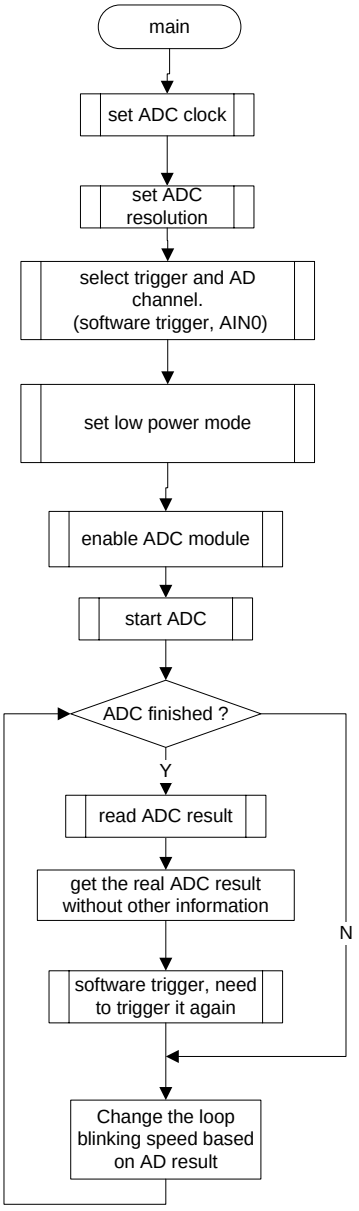
7-1-1 例: ADC データリード

ペリフェラルドライバ(ADC, GPIO)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. ADC 設定と初期化を行います。
2. AD 変換の開始と AD 変換結果の読み出しを行います。
3. ループ点滅スピードの調整に AD 変換結果を使用ます。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に AD クロック設定、チャネルやトリガタイプの選択を行い、ADC 動作を許可します。

```
    /* 1. set ADC clock */
    ADC_SetClk(TSB_AD, ADC_HOLD_FIX, ADC_FC_DIVIDE_LEVEL_NONE);

    /* 2. Set ADC resolution */
    ADC_SetResolution(TSB_AD, ADC_12BITS );

    /* 3. select trigger and AD channel, this time we use software trigger, */
    /*    the VR1 is connected to AIN0, remember to input with macro TRG_ENABLE() */
```

```
ADC_SetSWTrg(TSB_AD, ADC_REG0, TRG_ENABLE(ADC_AIN0));

/* 4. to let ADC module work, we must disable low power mode */
ADC_SetLowPowerMode(TSB_AD, DISABLE);

/* 5. enable ADC module */
ADC_Enable(TSB_AD);
```

次に AD 変換を開始します。

```
ADC_Start(TSB_AD, ADC_TRG_SW);
```

AD 変換を開始すると、AD 変換終了フラグがセットされるまで待ちます。AD 変換フラグがセットされると、AD 変換結果を取得し、次の AD 変換を開始します。

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

        if (adcState == DONE) {
            /* read ADC result when it is finished */
            result = ADC_GetConvertResult(TSB_AD, 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_AD, ADC_TRG_SW);
}
```

AD 変換結果を"myResult"変数に格納後、LED 点滅間隔を変更します。

7-2 CG

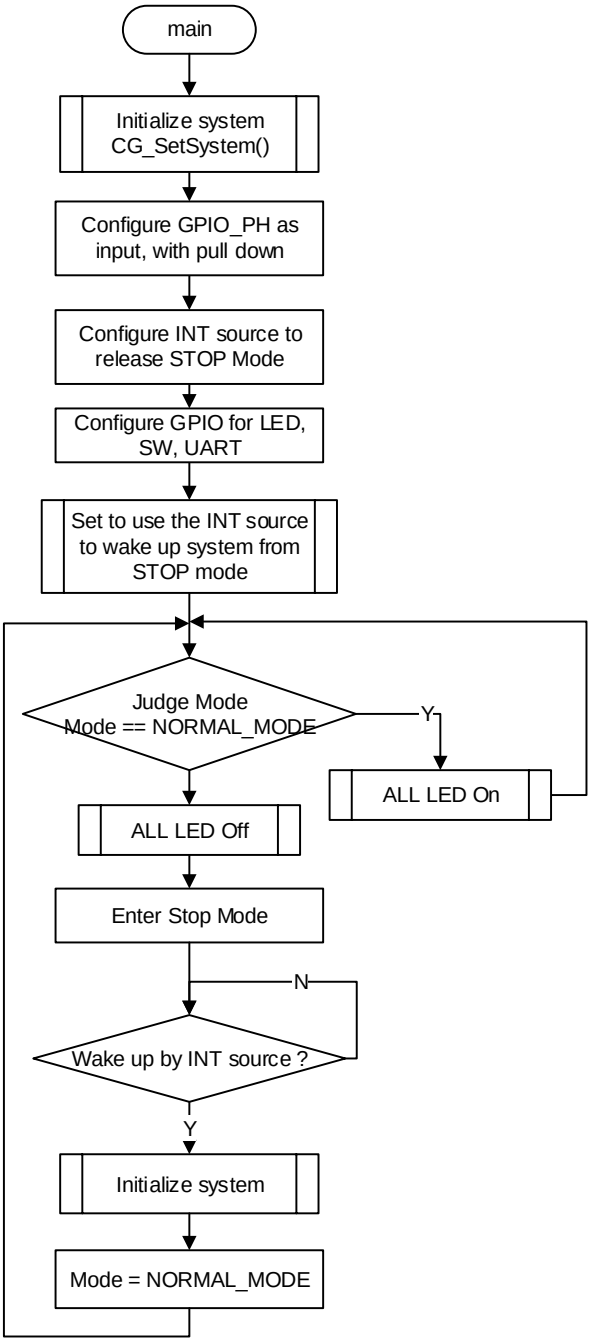
7-2-1 例: 動作モードの変更

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。

この例では以下を行います。

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サンプルプログラムです。

```
void CG_SetSystem(void)
{
    /* Set fgear = fc/2 */
    CG_SetFgearLevel(CG_DIVIDE_2);
    /* Set fperiph to fgear */
    CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
    CG_SetPhiT0Level(CG_DIVIDE_4);
    /* select external oscillator */
```

```
    CG_SetFoscSrc(CG_FOSC_EHOSC);
    CG_SetPortM(CG_PORTM_AS_HOSC);
    /* Enable high-speed oscillator */
    CG_SetFosc(CG_FOSC_EHOSC, ENABLE);
    /* 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();
}
```

スイッチ、LED、外部入力端子の設定

GPIO をスイッチ、LED、外部入力割り込み機能に設定します。

```
        /* Initialize system */
        CG_SetSystem();

        /* Configure GPIO for wakeup interrupt INT0 */
        GPIO_SetInput(GPIO_PH, GPIO_BIT_0);      /* Set PH0 to input */
        GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_1, GPIO_BIT_0);
/* Set PH0 for INT0 */
        GPIO_SetPullDown(GPIO_PH, GPIO_BIT_0, ENABLE);

        /* Configure GPIO for LED */
        LED_Init();

        /* Configure GPIO for SW */
        SW_Init();

        /* Configure GPIO for UART */
        hardware_init(UART_RETARGET);
```

スタンバイ解除のための外部割り込みの設定

スタンバイ解除のために INT0 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、INT0 を有効にします。

```
    /* Disable interrupts */
    __disable_irq();
    CG_ClearINTReq(CG_INT_SRC_0);
    /* Set external interrupt to wake up system */
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_0,
                                CG_INT_ACTIVE_STATE_RISING, ENABLE);
    NVIC_ClearPendingIRQ(INT0_IRQn);
    NVIC_EnableIRQ(INT0_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。

```
/* CG_NormalToStop */
void CG_NormalToStop(void)
{
    /* Set CG module: Normal ->Stop mode */
    CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
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。

```
Result CG_EnableClkMulCircuit(void)
{
    Result retval = ERROR;
    WorkState st = BUSY;
    retval = CG_SetPLL(ENABLE);
    if (retval == SUCCESS) {
        /* Set warm up time */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
CG_WUODR_PLL);
        CG_StartWarmUp();

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

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

    return retval;
}
```

7-3 DNF

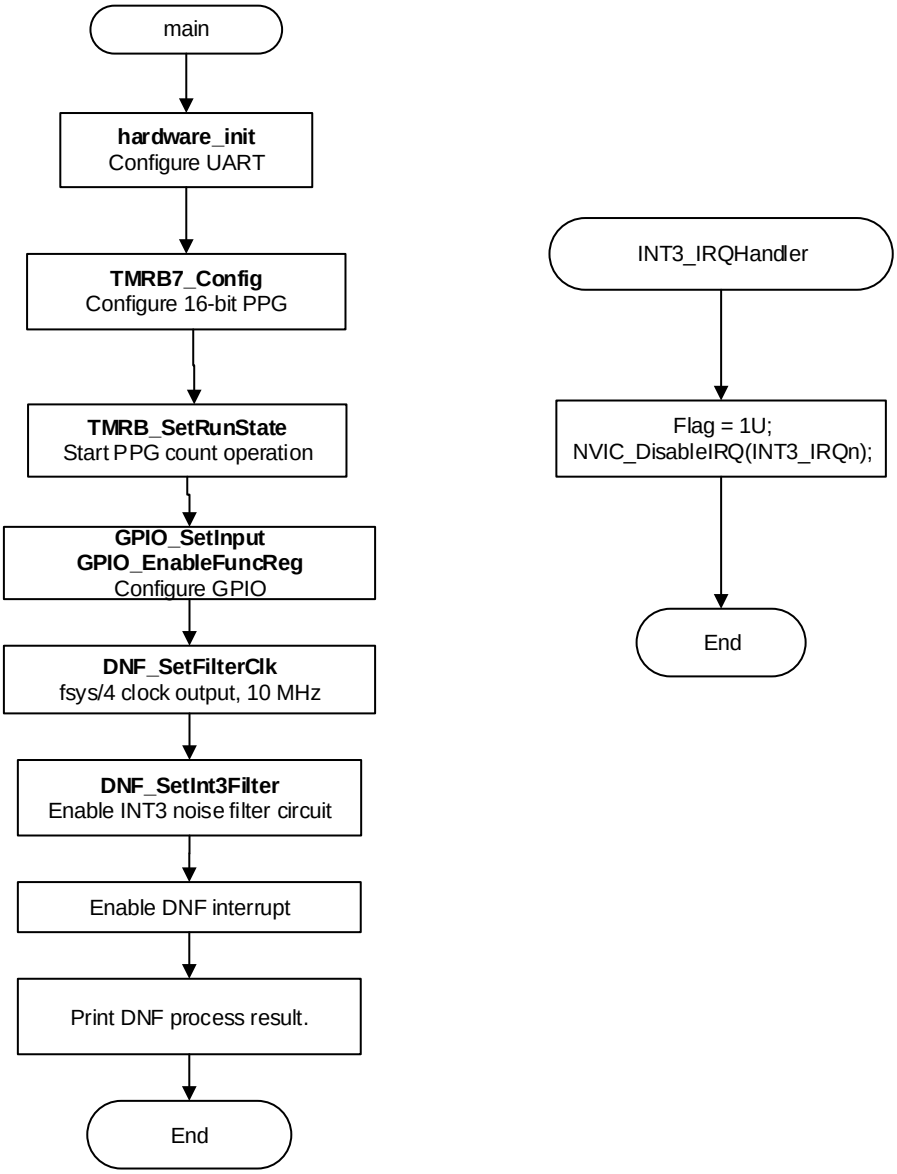
7-3-1 例: DNF

ペリフェラルドライバ(DNF, TMRB, GPIO, UART)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. DNF と TMRB の設定
2. 外部割り込み端子(INT7)に入力される信号のノイズ除去
3. DNF 結果の UART 表示

- **フローチャート**



- **サンプルプログラムのコードと説明**

文字列表示を行う UART の設定と TMRB の PPG 出力機能を利用して任意の周期とデューティの矩形波を生成するための設定を行います。その後、PPG 出力を開始します。

```
/* Configure UART */
hardware_init(UART_RETARGET);
/* Configure 16-bit PPG */
TMRB7_Config();
/* Start PPG count operation */
TMRB_SetRunState(TSB_TB7, TMRB_RUN);
/* Configure GPIO to INT3 (PA0) */
GPIO_SetInput(GPIO_PA, GPIO_BIT_0);
GPIO_EnableFuncReg(GPIO_PA, GPIO_FUNC_REG_2, GPIO_BIT_0);
```

最少ノイズ除去時間が 0.6μs となるようノイズ除去回路の設定を行います。その後、外部割り込み端子のノイズフィルタを許可します。

```
/* fsys/4 clock output, 10 MHz */
DNF_SetFilterClk(DNF_FILTER_CLK_FSYS_4);
/* Enable INT3 noise filter circuit */
DNF_SetInt3Filter(ENABLE);
```

外部割り込みを許可します。

```
__disable_irq();
NVIC_ClearPendingIRQ(INT3_IRQn);
NVIC_EnableIRQ(INT3_IRQn);
__enable_irq();
```

ノイズ除去結果の UART 出力を行います。

```
        while(1U) {
#if defined CORRECT_PULSE
            if (Flag == 1U) {
                common_uart_disp("DNF processed with error !\r\n");
            } else {
                common_uart_disp("Correct pulse is detected !\r\n");
            }
#elif defined NOISE_PULSE
            if (Flag == 1U) {
                common_uart_disp("Noise pulse is detected !\r\n");
            } else {
                common_uart_disp("DNF processed with error !\r\n");
            }
#endif
        }
```

7-4 FLASH

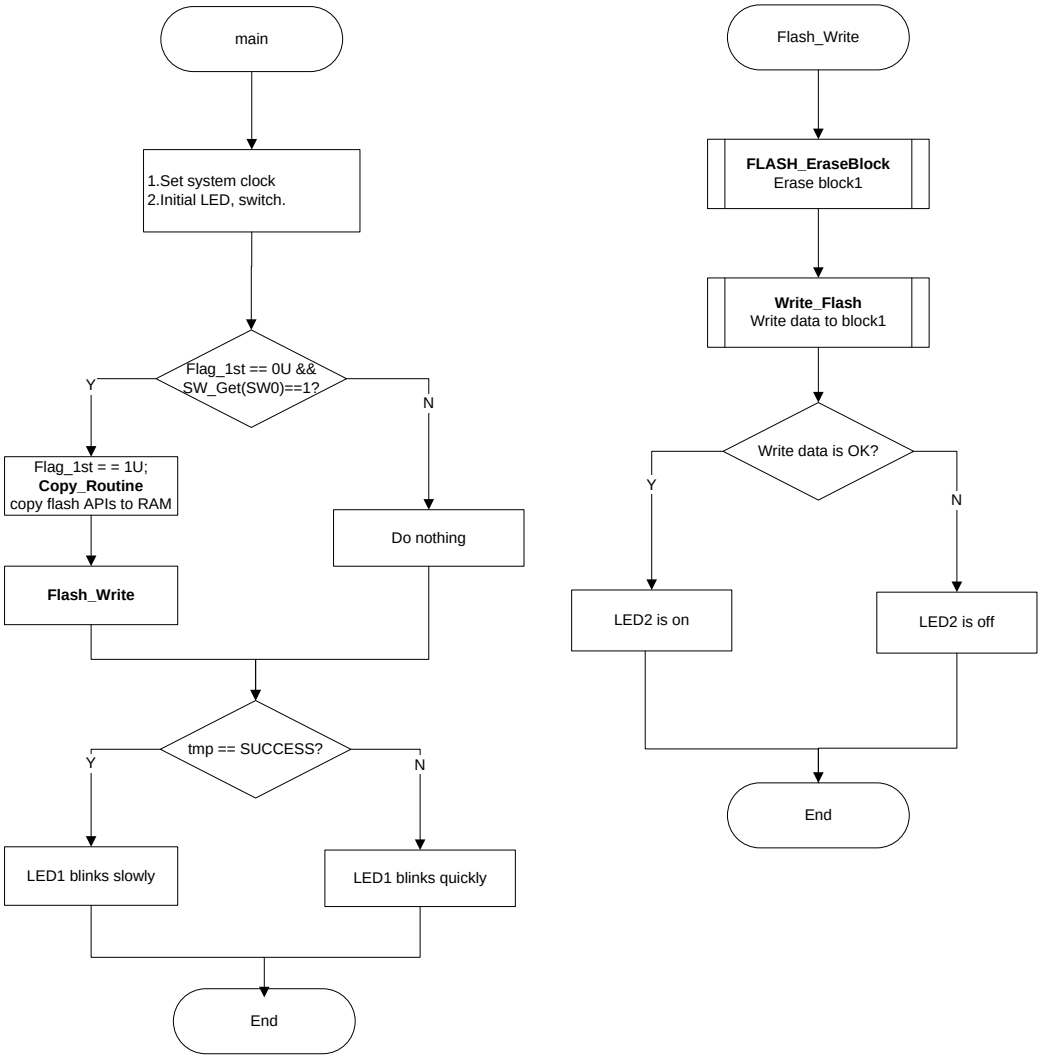
7-4-1 例: FLASH 書き込み

ペリフェラルドライバ(Flash, 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. 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、ユーザーブートモード)
3. ユーザー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, LCD, SW を初期化します。

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

リセット解除時に SW0 が OFF の場合、LED1 が早く点滅します。
SW0 が ON の場合、プログラムは Flash 操作用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。これは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です。

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

Flash 操作用 API を RAM にコピーした後、Flash_Write()関数を実行します。
正常に書き換えできた場合、LED1 がゆっくり点滅します。

```
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()関数は、FC_EraseBlock()関数と FC_WritePage()関数をコールし、
Block1 のブロック消去とページ書き換えを行います。
Block1 のリードデータと書き込みデータが同一の場合は LED2 は点灯し、不一致の場合は 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 API の FC_EraseBlock()関数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 API の FC_GetBlockProtectState()関数を使用し、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、ブロック消去が完了すると、Flash API の FC_GetBusyState()関数を使用して Flash ROM の状態を確認します。同時に、タイムアウトカウンタを使用し、動作が想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 */
                }
            }
```

Flash API の FC_WritePage ()関数は自動的に 1 ページ分の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 FUART

7-5-1 例: ループバック

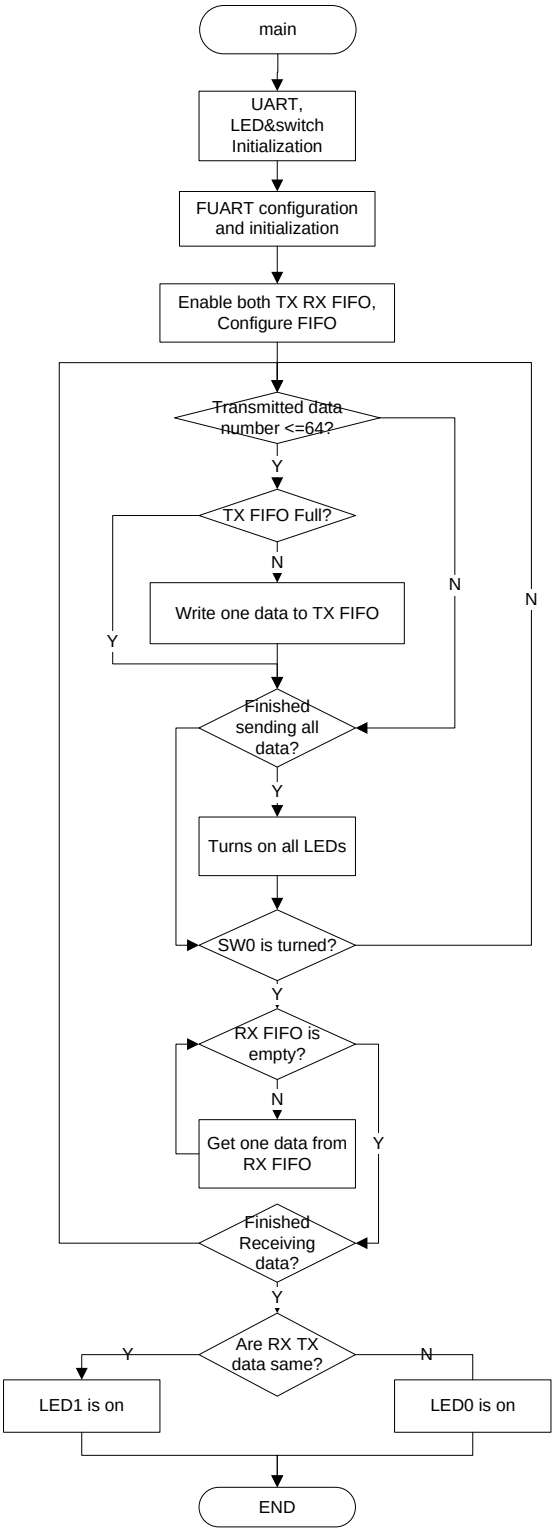
ペリフェラルドライバ(FUART)を用いたサンプルプログラムです。

本プログラムはハードウェアフロー制御機能を確認するために 2 回実行することができます。
1 回目:UT0RTS と UT0CTS ハードウェアフロー制御をイネーブルにします。
2 回目:UT0RTS と UT0CTS ハードウェアフロー制御ををディセーブルにします。

このサンプルプログラムは以下を行います。

1. LED、UART、スイッチの初期化、フル UART の設定と初期化
2. フル UART 送信データ処理
3. データ受信のため、SW0 を ON にする
4. データ受信終了後、受信データを送信データと比較

- **フローチャート:**


main
UART, LED&switch Initialization
FUART configuration and initialization
Enable both TX RX FIFO, Configure FIFO
Transmitted data number <=64?
Y
TX FIFO Full?
N
Write one data to TX FIFO
Y
N
N
Finished sending all data?
Y
Turns on all LEDs
SW0 is turned?
Y
RX FIFO is empty?
N
Get one data from RX FIFO
Y
Finished Receiving data?
Y
Are RX TX data same?
Y
N
LED1 is on
LED0 is on
END

- **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 GPIO を LED 用に設定します。

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

FUART0 用 GPIO を設定します。

```
/* Configure port PC5 to be UT0TXD50B */
GPIO_SetOutput(GPIO_PC, GPIO_BIT_5);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_5, ENABLE);
GPIO_SetPullDown(GPIO_PC, GPIO_BIT_5, DISABLE);
GPIO_SetOpenDrain(GPIO_PC, GPIO_BIT_5, DISABLE);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_5, GPIO_BIT_5);

/* Configure port PC6 to be UT0TXD50A */
GPIO_SetOutput(GPIO_PC, GPIO_BIT_6);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_6, ENABLE);
GPIO_SetPullDown(GPIO_PC, GPIO_BIT_6, DISABLE);
GPIO_SetOpenDrain(GPIO_PC, GPIO_BIT_6, DISABLE);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_5, GPIO_BIT_6);

/* Configure port PC7 to be UT0RXD50 */
GPIO_SetInput(GPIO_PC, GPIO_BIT_7);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_7, ENABLE);
GPIO_SetPullDown(GPIO_PC, GPIO_BIT_7, DISABLE);
GPIO_SetOpenDrain(GPIO_PC, GPIO_BIT_7, DISABLE);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_5, GPIO_BIT_7);
```

FUART_InitTypeDef 構成を生成し、全データフィールドを入力します。その後 FUART0 を初期化します。

```
FUART_InitTypeDef myFUART;

myFUART.BaudRate = 300U;
myFUART.DataBits = FUART_DATA_BITS_8;
myFUART.StopBits = FUART_STOP_BITS_1;
myFUART.Parity = FUART_1_PARITY;
myFUART.Mode = FUART_ENABLE_TX | FUART_ENABLE_RX;
myFUART.FlowCtrl = FUART_NONE_FLOW_CTRL;

FUART_Init(FUART0, &myFUART);
```

50%デューティモードを許可します。

```
FUART_SetPeriodDetection(FUART0, FUART_PERIOD_DETEC_7 );
FUART_SetTxTerminalMode(FUART0, FUART_1_TERMINAL);
FUART_SetStartBitTerminal(FUART0, FUART_TXD50A);
FUART_EnableDuty(FUART0);
```

FUART 周辺ドライバを使用して FUART0 の許可、と FIFO の設定を行います。

```
FUART_Enable(FUART0);
FUART_EnableFIFO(FUART0);
FUART_SetINTFIFOLevel(FUART0, FUART_RX_FIFO_LEVEL_16,
FUART_TX_FIFO_LEVEL_4);
```

その後、データ送信を開始します。フル UART は 64 種のデータ値のみ送信したのち、送信 FIFO が正常あるいは空の場合、データを送信します。各データが送信されると、 LED はデ

ータを表示します。全データが送信されると、全 LED が点灯します。

```
    if (cntTx < MAX_BUFSIZE) {
        FIFOStatus = FUART_GetStorageStatus(FUART0, FUART_TX);
        if ((FIFOStatus == FUART_STORAGE_EMPTY) || (FIFOStatus ==
FUART_STORAGE_NORMAL)) {
            FUART_SetTxData(FUART0, Tx_Buf[cntTx]);
            LED_TXDataDisplay(Tx_Buf[cntTx]);
            cntTx++;
            if (64U == cntTx) {
                LED_On(LED_ALL);      /* sending data is finished */
            }
        }
    }
```

SW0 が On するごとに、プログラムは受信 FIFO からのデータ読み出しを開始します。 データが存在している場合、プログラムは FIFO が空になるまでデータの読み出しを継続します。SW0 が On であり、データが存在しない場合には、受信データと送信データの比較を開始します。受信データが送信データと同一の場合、LED1 を点灯し、受信データが送信データと異なる場合、LED0 を点灯します。

```
    SW0_last = SW0_this;
    SW0_this = SW_Get(SW0);
    SW0_this = !SW0_last;
    if (SW0_last != SW0_this) {        /* turn the switch SW0 */
        while (FUART_STORAGE_EMPTY != FUART_GetStorageStatus(FUART0,
FUART_RX)) {
            receive = FUART_GetRxData(FUART0);
            Rx_Buf[cntRx] = receive;
            cntRx++;
        }

        rxlast = rxthis;
        rxthis = cntRx;

        if (rxlast != rxthis) {        /* there are some data that has been received   */
            LED_Off(LED_ALL);
            LED_On(LED2);
        } else {                    /* receiving data is finished */
            result = Buffercompare(Tx_Buf, Rx_Buf, MAX_BUFSIZE);
            if (result == SAME) {
                /* received data are same with transmitted daa */
                LED_Off(LED_ALL);
                LED_On(LED1);
                while (1) {
                }
            } else {
                /* received data are different with transmitted data */
                LED_Off(LED_ALL);
                LED_On(LED0);
                while (1) {
                }
            }
        }
    }
```

7-6 GPIO

7-6-1 Example: GPIO Data Read

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 からのデータ読み出し
3. GPIO へのデータ書き込み

- **サンプルプログラムのコードと説明:**

まず GPIO_SetOutput()を使用して GPIO を LED に設定します。そして GPIO_SetInput()を使用して GPIO をキーに設定します。

```
    GPIO_SetOutput(GPIO_PA, GPIO_BIT_0 | GPIO_BIT_1 | GPIO_BIT_2 |
GPIO_BIT_3);

    GPIO_SetInput(GPIO_PH,   GPIO_BIT_4 | GPIO_BIT_5 | GPIO_BIT_6 |
GPIO_BIT_7);
```

for(; ;)ループの中で、LED サンプルを実行します: 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_WriteDataBit()を使用して LED を点灯します。

```
    if (GPIO_ReadDataBit(GPIO_PH, GPIO_BIT_4) == 1U) {
                    tmp = 1U;
    } else {
                  /*Do nothing */
    }
```

GPIO_WriteData()を使用して LED を点灯します。

```
    uint8_t tmp;
    tmp = GPIO_ReadData(GPIO_PA);
    tmp |= led;
    GPIO_WriteData(GPIO_PA, tmp);
```

GPIO_WriteData()を使用して LED を消灯します。

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

7-7 OFD

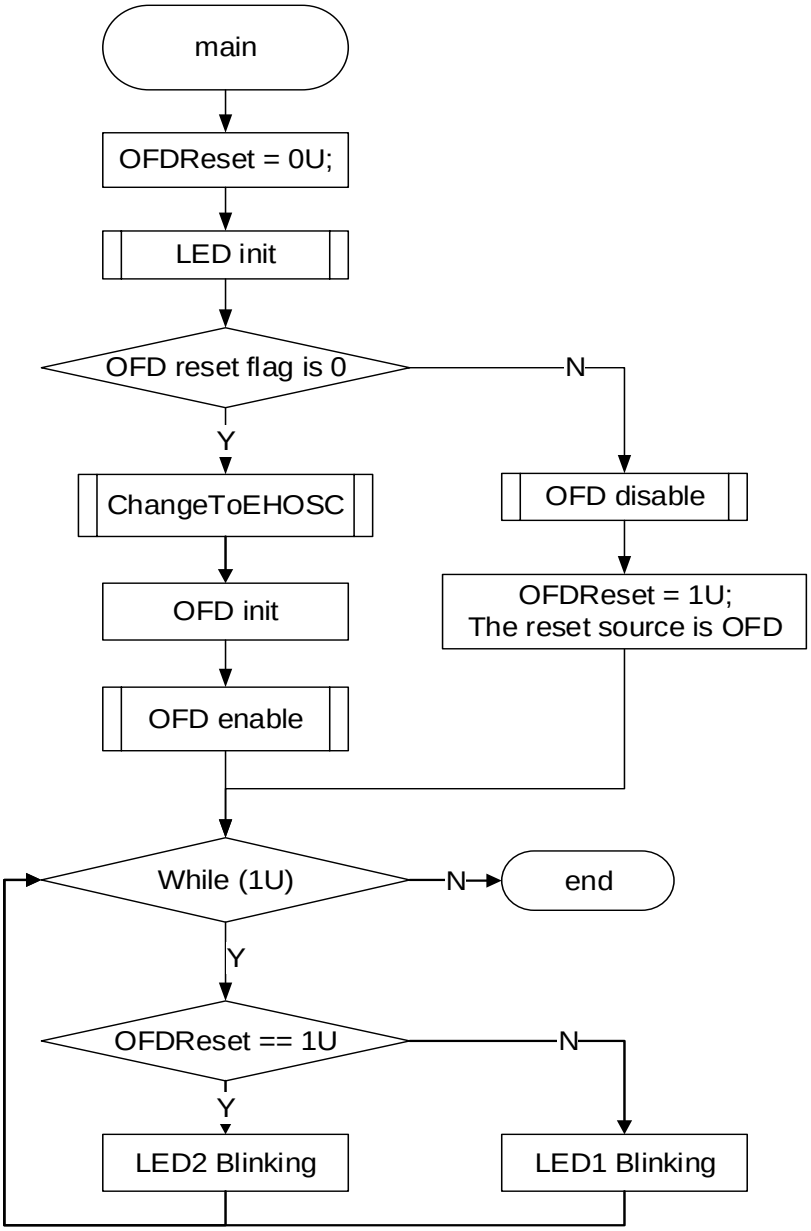
7-7-1 例: OFD

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。

この例では以下を行います。

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 のリセットフラグを確認します。

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

このフラグが設定されていない場合、外部発振器の許可と検知周波数を設定して OFD を初期化します。以下は外部発振器を許可する処理です。

```
void ChangeToEHOSC(void)
{
    /* Use the external oscillation */
    TSB_CG->OSCCR &= 0x000FFFFFUL;
    TSB_CG->OSCCR |= 0xFFF00000UL;    /* Set the warm up time WUODR[11:0]
*/
    TSB_CG_OSCCR_HOSCON = 1U;     /* Set gpio 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 */
    while (TSB_CG_OSCCR_WUEF) {
        /* Wait for warm-up finish */
    }
    TSB_CG_OSCCR_OSCSEL = 1U;     /* Use the external oscillation */
    TSB_CG_OSCCR_XEN2 = 1U;        /* Enable internal oscillator for ofd */
}
```

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);
```

検出周波数を設定します。

```
    OFD_SetDetectionFrequency(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_HIGHER_COUNT_ABNORMAL,
                                                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示します。

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。

外部発振器にて発振している場合は LED1 を点滅し、内部発振器にて発振している場合は、LED2 を点滅します。

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

7-8 RMC

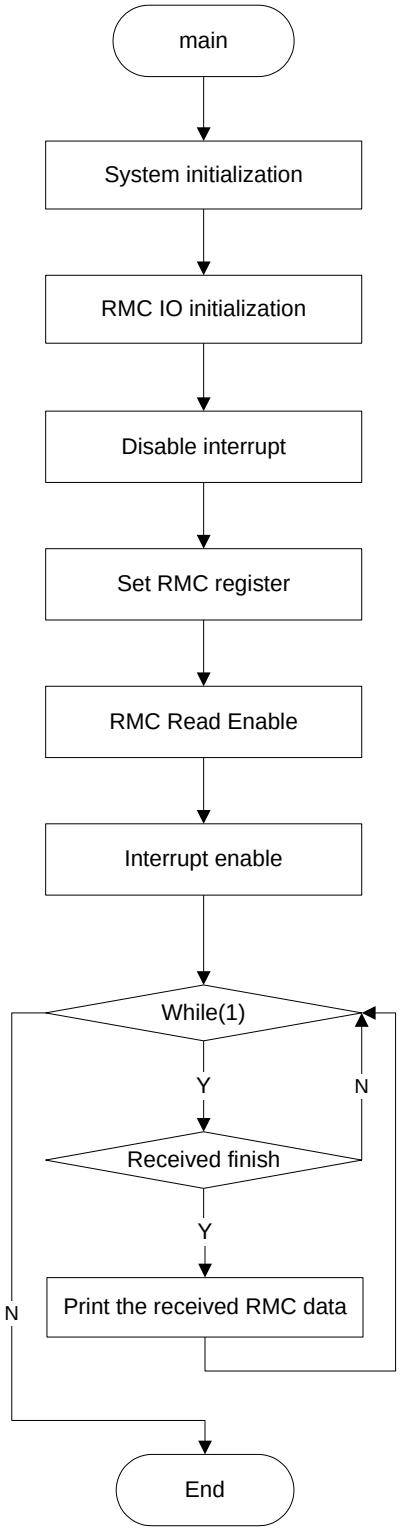
7-8-1 例: RMC 受信

ペリフェラルドライバ(RMC, UART, GPIO)を使用した簡単なサンプルソフトです。

この例では以下を行います。

1. RMC 用 IO の初期化
2. RMC データの受信

- **フローチャート**


main
System initialization
RMC IO initialization
Disable interrupt
Set RMC register
RMC Read Enable
Interrupt enable
While(1)
Y
N
Received finish
Y
N
Print the received RMC data
End

- **サンプルプログラムのコードと説明**

まず main()にて myRMC structure 構造体を作成し、データフィールドにデータを入力します。

```
RMC_InitTypeDef myRMC;

myRMC.LeaderPara.MaxCycle = RMC_MAX_CYCLE;
myRMC.LeaderPara.MinCycle = RMC_MIN_CYCLE;
myRMC.LeaderPara.MaxLowWidth = RMC_MAX_LOW_WIDTH;
myRMC.LeaderPara.MinLowWidth = RMC_MIN_LOW_WIDTH;
myRMC.LeaderPara.LeaderDetectionState = ENABLE;
myRMC.LeaderPara.LeaderINTState = DISABLE;
myRMC.FallingEdgeINTState = DISABLE;
myRMC.SignalRxMethod = RMC_RX_IN_CYCLE_METHOD;
myRMC.LowWidth = RMC_TRG_LOW_WIDTH;
myRMC.MaxDataBitCycle = RMC_TRG_MAX_DATA_BIT_CYCLE;
myRMC.LargerThreshold = RMC_LARGER_THRESHOLD;
myRMC.SmallerThreshold = RMC_SMALLER_THRESHOLD;
myRMC.InputSignalReversedState = DISABLE;
myRMC.NoiseCancellationTime = RMC_NOISE_CANCELLATION_TIME;
```

次に、RMC チャネル 0 を初期化し、有効にします。

```
RMC_Enable(RMCx);
```

構造体には RMC の基本設定が含まれます。

```
RMC_Init(RMCx, &myRMC);
```

RMC チャネル 0 の受信を有効にする。

```
RMC_SetRxCtrl(RMCx, ENABLE);
```

割り込み INTRMCRX_IRQHandler()にて RMC チャネル 0 の割り込み要因を取得します。

```
RMC_INTFactor myRMC_INTFactor;
myRMC_INTFactor = RMC_GetINTFactor(TSB_RMC);
```

RMC チャネル 0 のリーダー検出結果を取得します。

```
RMC_LeaderDetection myRMC_LeaderDetection;
myRMC_LeaderDetection = RMC_GetLeader(TSB_RMC);
```

RMC チャネル 0 の受信データを取得します。次に、RMC チャネル 0 を初期化し、有効にします。

```
RMC_RxDataTypeDef myRMC_RxDataDef;
myRMC_RxDataDef = RMC_GetRxData(TSB_RMC);
```

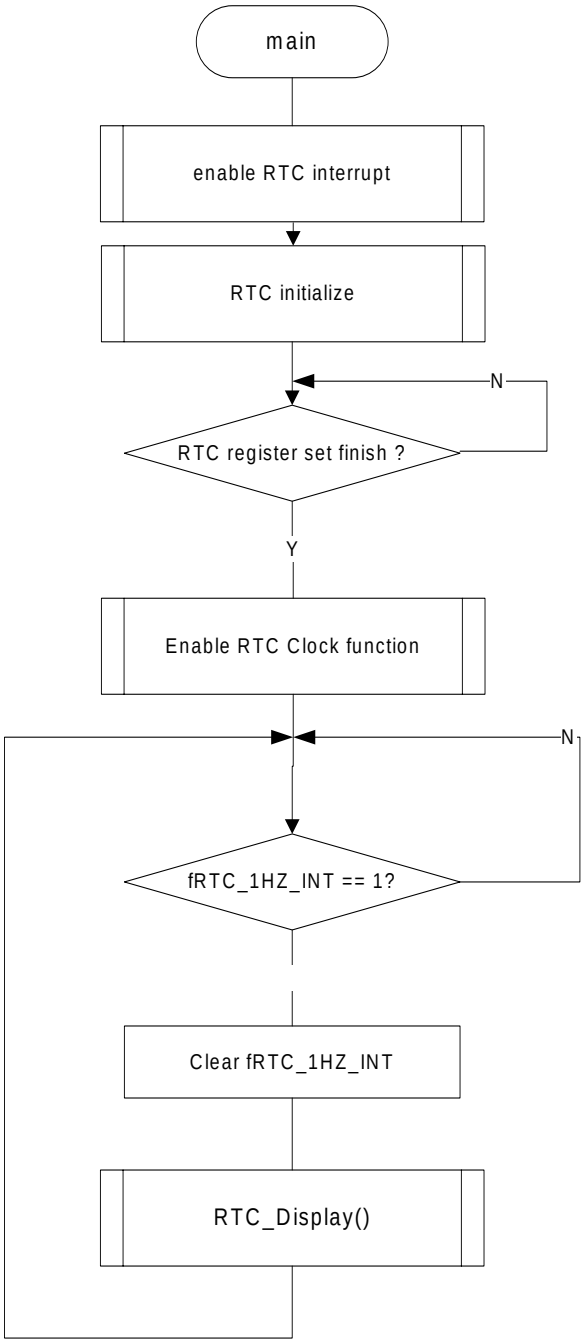
7-9 RTC

ペリフェラルドライバ(RTC, CG, GPIO)を使用した簡単なサンプルソフトです。

この例では以下を行います。

1. RTC の初期化
2. RTC 秒の取得

- **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、RTC 割り込みによる SLEEP モードの解除設定を行います。

```
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_RTC,
                                    CG_INT_ACTIVE_STATE_FALLING, ENABLE);
```

RTC の初期化で、RTC_DateTypeDef と RTC_TimeTypeDef 構造を作成し、全データ項目を設定します。ここでは 2015/06/12 13:54:00, 24 時間表示フォーマットを初期設定としています。

```
    Date_Struct.LeapYear = RTC_LEAP_YEAR_3;
```

```
Date_Struct.Year = (uint8_t) 15U;
Date_Struct.Month = (uint8_t) 6U;
Date_Struct.Date = (uint8_t) 12U;
Date_Struct.Day = RTC_FRI;

Time_Struct.HourMode = RTC_24_HOUR_MODE;
Time_Struct.Hour = (uint8_t) 13U;
Time_Struct.Min = (uint8_t) 54U;
Time_Struct.Sec = (uint8_t) 00U;
```

クロックとアラーム機能を禁止します。

```
RTC_DisableClock();
RTC_DisableAlarm();
```

RTC 秒カウンタのリセット、1Hz 割り込みの許可、RTCINT の許可を行います。

```
RTC_ResetClockSec();
RTC_EnableClock();
RTC_EnableAlarm();
```

RTC の時間と日付を設定し、1Hz 割り込みと RTC 割り込みを設定します。

```
RTC_SetTimeValue(&Time_Struct);
RTC_SetDateValue(&Date_Struct);
RTC_SetAlarmOutput(RTC_PULSE_1_HZ);
RTC_SetRTCINT(ENABLE);
```

上記項目を設定後、RTC レジスタ設定の終了を待ち、RTC 時計機能を許可します。その後、RTC クロック機能を許可します。

```
NVIC_EnableIRQ(INTRTC_IRQn);
__enable_irq();

/* waiting for RTC register set finish */
while (fRTC_1HZ_INT != 1U) {
      /* Do nothing */
}
fRTC_1HZ_INT = 0U;

/* Enable RTC Clock function */
RTC_EnableClock();
```

RTC 割り込みが 1 秒ごとに発生するよう設定を行い、その後、RTC 割り込み要求のクリアを行います。

```
fRTC_1HZ_INT = 1U;
/*clear RTC interrupt request */
CG_ClearINTReq(CG_INT_SRC_RTC);
```

RTC 割り込み発生後、第 2 番目の値がバイナリで LED ディスプレイに送信されます。
下記に、第 2 番目の値の取得方法と表示方法を表します。

```
/* Send RTC Date value to LCD */
/* Get RTC Date value */
Year = RTC_GetYear();
Month = RTC_GetMonth();
Date = RTC_GetDate(RTC_CLOCK_MODE);
```

```
/* Set LCD display */
/* Dispaly year */
RTC_Disp_YMD[0] = '2';
RTC_Disp_YMD[1] = '0';
RTC_Disp_YMD[2] = (Year / 10U) + 0x30U;
RTC_Disp_YMD[3] = (Year % 10U) + 0x30U;
RTC_Disp_YMD[4] = '/';
/* Display month */
RTC_Disp_YMD[5] = (Month / 10U) + 0x30U;
RTC_Disp_YMD[6] = (Month % 10U) + 0x30U;
RTC_Disp_YMD[7] = '/';
/* Display date */
RTC_Disp_YMD[8] = (Date / 10U) + 0x30U;
RTC_Disp_YMD[9] = (Date % 10U) + 0x30U;

RTC_Disp_YMD[10] = 0U;
Send_To_LCD(FIRST_LINE, LCD_COMMAND);
Send_LCD_Text(RTC_Disp_YMD);

/* Send RTC Time value to LCD */
/* Get RTC Time value */
Hour = RTC_GetHour(RTC_CLOCK_MODE);
Min = RTC_GetMin(RTC_CLOCK_MODE);
Sec = RTC_GetSec();

/* Display hour */
RTC_Disp_HMS[0] = (Hour / 10U) + 0x30U;
RTC_Disp_HMS[1] = (Hour % 10U) + 0x30U;
RTC_Disp_HMS[2] = ':';
/* Display min */
RTC_Disp_HMS[3] = (Min / 10U) + 0x30U;
RTC_Disp_HMS[4] = (Min % 10U) + 0x30U;
RTC_Disp_HMS[5] = ':';
/* Display sec */
RTC_Disp_HMS[6] = (Sec / 10U) + 0x30U;
RTC_Disp_HMS[7] = (Sec % 10U) + 0x30U;
RTC_Disp_HMS[8] = 0U;

/* LCD Display */
Send_To_LCD(SECOND_LINE, LCD_COMMAND);
Send_LCD_Text(RTC_Disp_HMS);
```

7-10 SBI

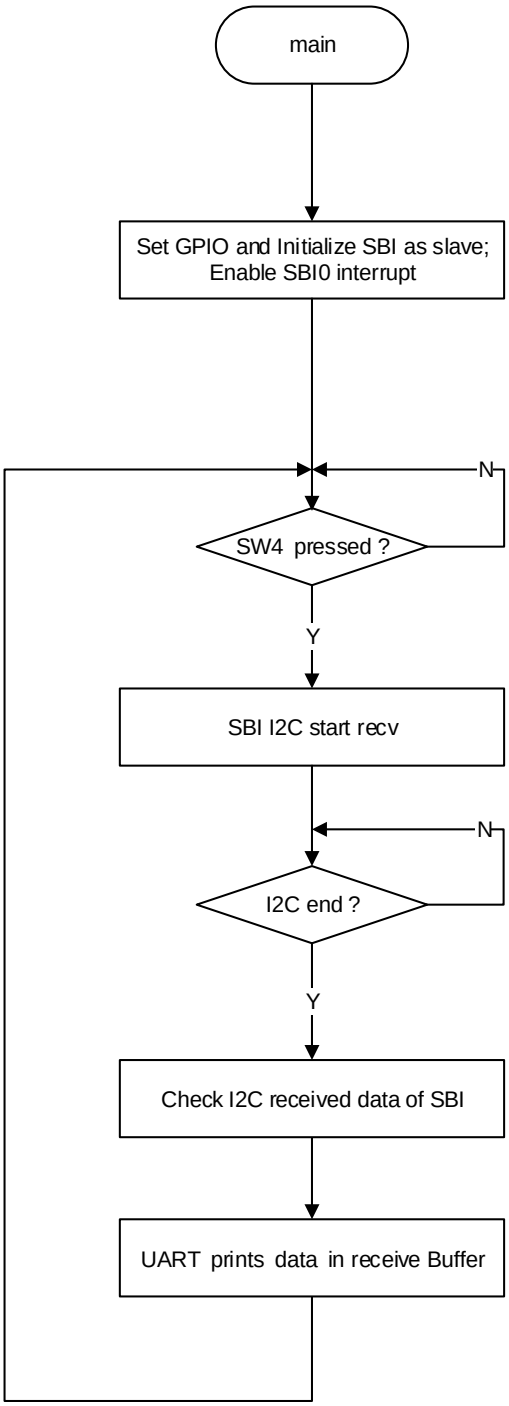
7-10-1 例: SBI スレーブ

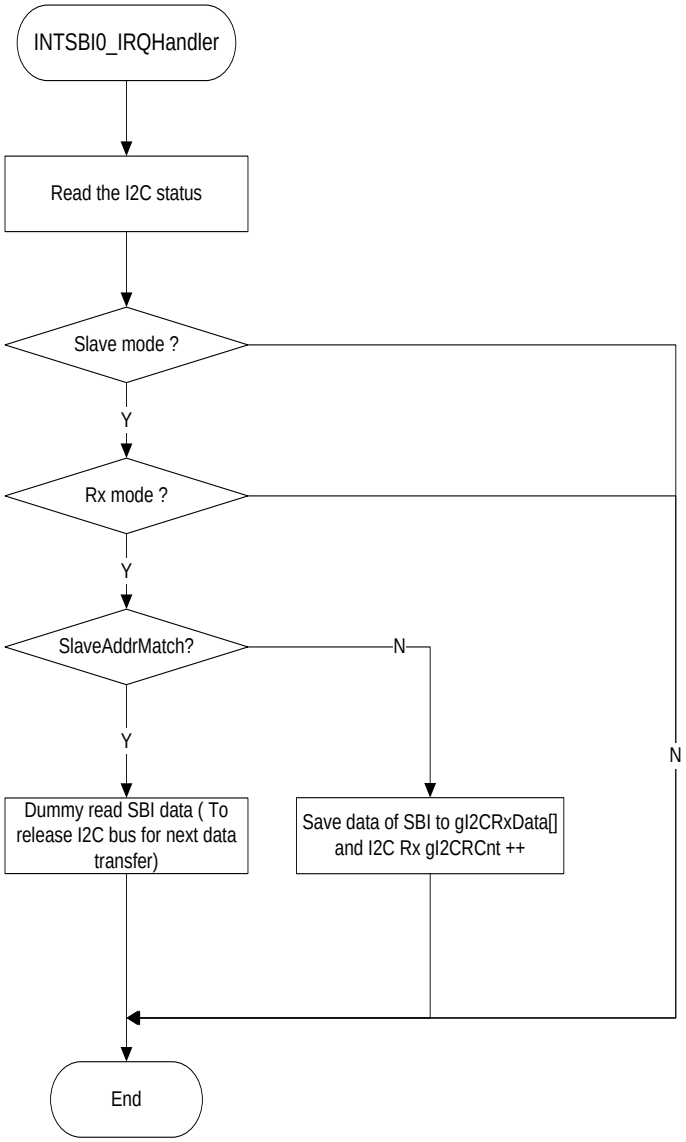
ペリフェラルドライバ(SBI, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. SBI 設定、I2C 初期化
2. I2C スレーブによるデータ受信

- **フローチャート**


main
Set GPIO and Initialize SBI as slave;
Enable SBI0 interrupt
N
SW4 pressed ?
Y
SBI I2C start recv
N
I2C end ?
Y
Check I2C received data of SBI
UART prints data in receive Buffer


INTSBI0_IRQHandler
Read the I2C status
Slave mode ?
Y
Rx mode ?
Y
SlaveAddrMatch?
N
Y
N
Dummy read SBI data (To release I2C bus for next data transfer)
Save data of SBI to gI2CRxData[] and I2C Rx gI2CRCnt ++
End

- **サンプルプログラムのコードと説明**

まず、GPIO を SBI I2C 用に設定します。

```
    GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_3, GPIO_BIT_0 |
GPIO_BIT_1);
    GPIO_SetOutputEnableReg(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1,
ENABLE);
    GPIO_SetInputEnableReg(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
    GPIO_SetOpenDrain(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
/* Pull up for SDA&SCL */
    GPIO_SetPullUp(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
```

次に SBI チャネルの初期化と INTSBI 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_SBI0);
    SBI_SWReset(TSB_SBI0);
    SBI_InitI2C(TSB_SBI0, &myI2C);
    NVIC_EnableIRQ(INTSBI0_IRQn);
```

上記設定を行った後、I2C 受信待ちを行います。
I2C 受信バッファをクリアし、RX バッファーをクリアします。

```
        /* 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 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 SBI_GetReceiveData() 関数を用いて行います。
I2C ストップコンディションはマスタによって制御します。

```
void INTSBI0_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI0_TypeDef *SBIy;
    SBI_I2CState sbi0_sr;

    SBIy = TSB_SBI0;
    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 */
    }
}
```

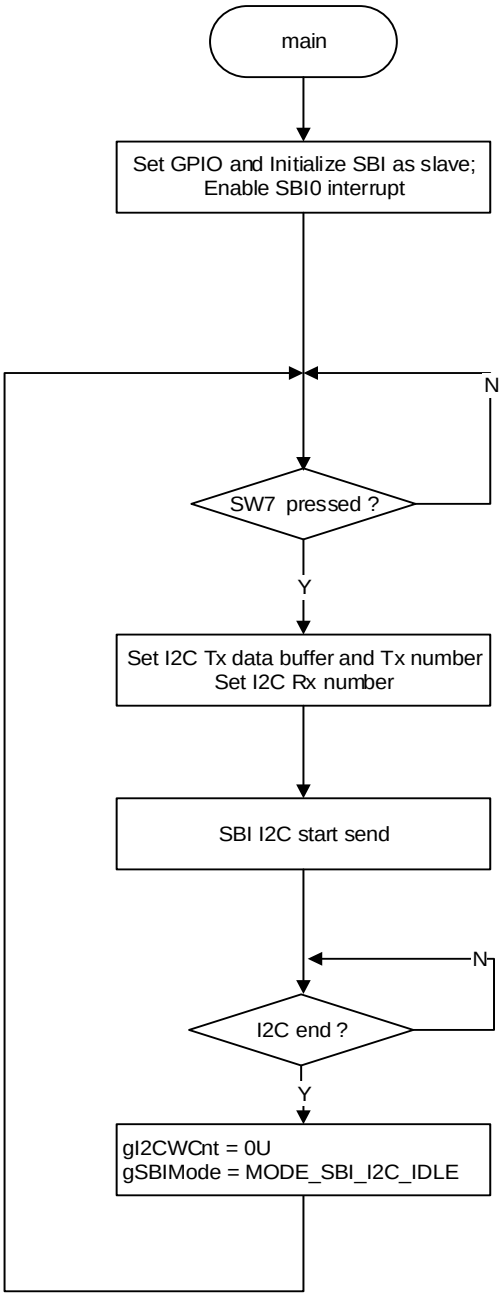
7-10-2　例: SBI マスタ

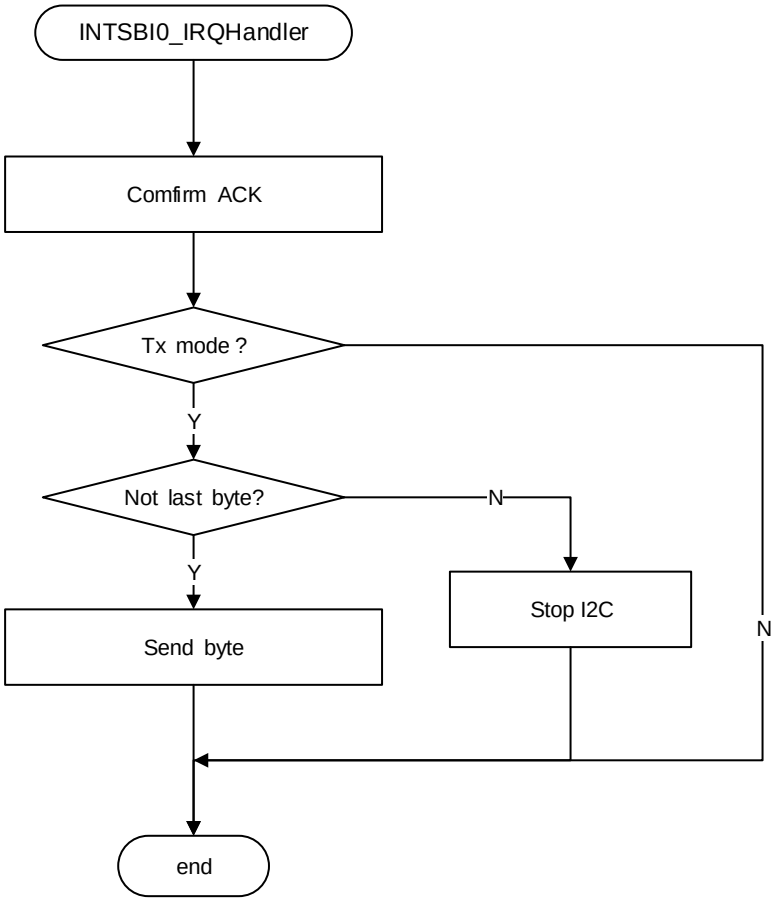
ペリフェラルドライバ(SBI, GPIO)を使用した簡単なサンプルソフトです。

この例では以下を行います。

1. SBI 設定、I2C 初期化
2. I2C マスターによるデータ送信

- **フローチャート**


main
Set GPIO and Initialize SBI as slave;
Enable SBI0 interrupt
N
SW7 pressed ?
Y
Set I2C Tx data buffer and Tx number
Set I2C Rx number
SBI I2C start send
N
I2C end ?
Y
gI2CWCnt = 0U
gSBIMode = MODE_SBI_I2C_IDLE



- **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_PC, GPIO_FUNC_REG_3, GPIO_BIT_0 |
GPIO_BIT_1);
GPIO_SetOutputEnableReg(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
GPIO_SetInputEnableReg(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
GPIO_SetOpenDrain(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
/* Pull up for SDA&SCL */
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_0 | GPIO_BIT_1, ENABLE);
```

次に SBI0 チャネルの初期化と INTI2C 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);
```

上記設定を行った後、I2C 送信開始します。
I2C 受信バッファをクリアし、SBI TX バッファーとバッファ長を設定します。その後 RX バッファーをクリアします。

```
        /* 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_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;
```

データ転送は INTSBI0 にて処理します。

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、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_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 (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 */
    }
}
```

7-11 SSP

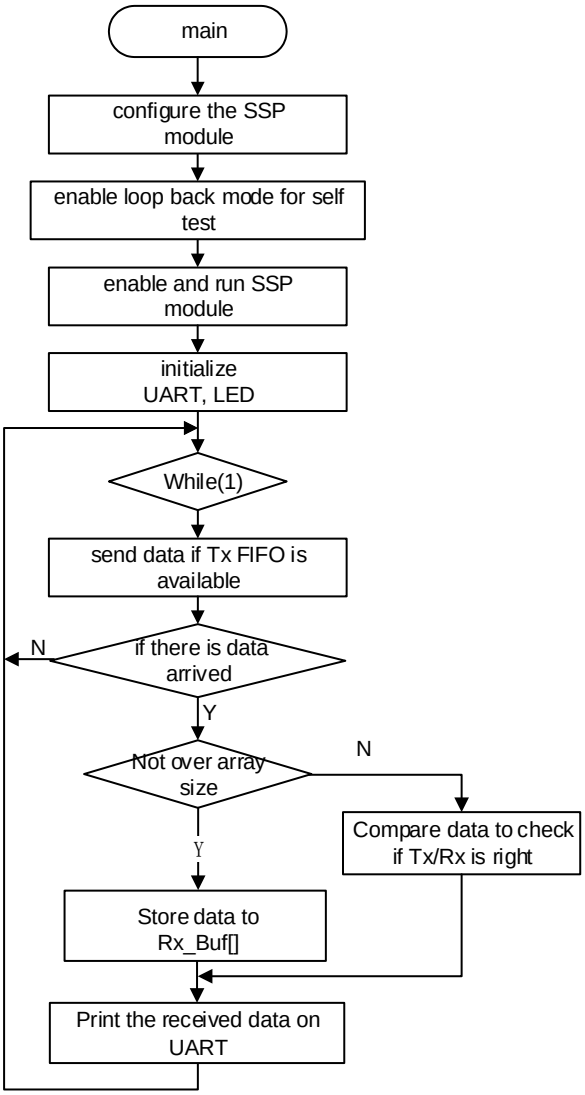
7-11-1 例: SSP0 セルフループバック

ペリフェラルドライバ(SSP, GPIO、UART)を使用したサンプルソフトです。

この例では以下を行います。

1. SSP0 の設定と初期化
2. 自身で送受信を行うループバックモードの許可

- **フローチャート**



- **サンプルプログラムのコードと説明**

SPI フレームフォーマットを選択します。

```
    /* Configure the SSP module */
    initSSP.FrameFormat = SSP_FORMAT_SPI;
```

最大または最小ビットレートを選択します。

```
    /* Default is to run at maximum bit rate */
    initSSP.PreScale = 2U;
    initSSP.ClkRate = 1U;

    /* Define BITRATE_MIN to run at minimum bit rate */
    /*     BitRate = fSYS / (PreScale x (1 + ClkRate)) */
#ifdef BITRATE_MIN
    initSSP.PreScale = 254U;
    initSSP.ClkRate = 255U;
#endif
    initSSP.ClkPolarity = SSP_POLARITY_LOW;
    initSSP.ClkPhase = SSP_PHASE_FIRST_EDGE;
    initSSP.DataSize = 16U;
```

```
    initSSP.Mode = SSP_MASTER;
    SSP_Init(TSB_SSP0, &initSSP);
```

SSP0 をループバックモードに設定します。

```
    /* Enable loop back mode for self test */
    SSP_SetLoopBackMode(TSB_SSP0, ENABLE);

    /* Enable and run SSP module */
    SSP_Enable(TSB_SSP0);
```

LED の設定を行います。

```
    /* Initialize LEDs on M38x board before display something */
    LED_Init();
    hardware_init(UART_RETARGET);
```

whileループの中で、データ送信を行い、受信データが同じ場合はLED2とLED3を点灯し、異なる場合はLED0とLED1を点灯します。受信データはUART経由で確認することができます。

```
    while (1) {

        datTx++;
        /* Send data if Tx FIFO is available */
        fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_TX);
        if ((fifoState == SSP_FIFO_EMPTY) || (fifoState == SSP_FIFO_NORMAL))
        {
            SSP_SetTxData(TSB_SSP0, datTx);
            if (cntTx < MAX_BUFSIZE) {
                Tx_Buf[cntTx] = datTx;
                cntTx++;
            } else {
                /* Do nothing */
            }
        } else {
            /* Do nothing */
        }

        /* Check if there is data arrived */
        fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_RX);
        if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
            receive = SSP_GetRxData(TSB_SSP0);
            if (cntRx < MAX_BUFSIZE) {
                Rx_Buf[cntRx] = receive;
                cntRx++;
            } else {
                /* Place a break point here to check if receive data is right. */
                /* Success Criteria:       */
                /* Every data transmitted from Tx_Buf is received in Rx_Buf. */
                /* When the line "#define BITRATE_MIN" is commented, the SSP
                   is run in maximum */
                /* bit rate, so we can find there is enough time to transmit date
                   from 1 to    */
                /* MAX_BUFSIZE one by one. But if we uncomment that line,

                   SSP is run in */
                /* minimum bit rate, we will find that receive data can't catch
```

```
                    "datTx++", */
                /* in this so slow bit rate, when the Tx FIFO is available, the
                    cntTx has       */
                /* been increased so much. */
                __NOP();
                result = Buffercompare(Tx_Buf, Rx_Buf, MAX_BUFSIZE);
                if (result    == NOT_SAME)
                {
                    LED_On(LED0);
                    LED_On(LED1);
                } else
                {
                    LED_On(LED2);
                    LED_On(LED3);
                }
            }

        } else {
            /* Do nothing */
        }

        sprintf((char *) SSP_RX_Data, “SSP RX DATA : %d", receive);
        common_uart_disp(SSP_RX _Data);
        common_uart_disp(“\n”);
    }
```

7-12　TMRB

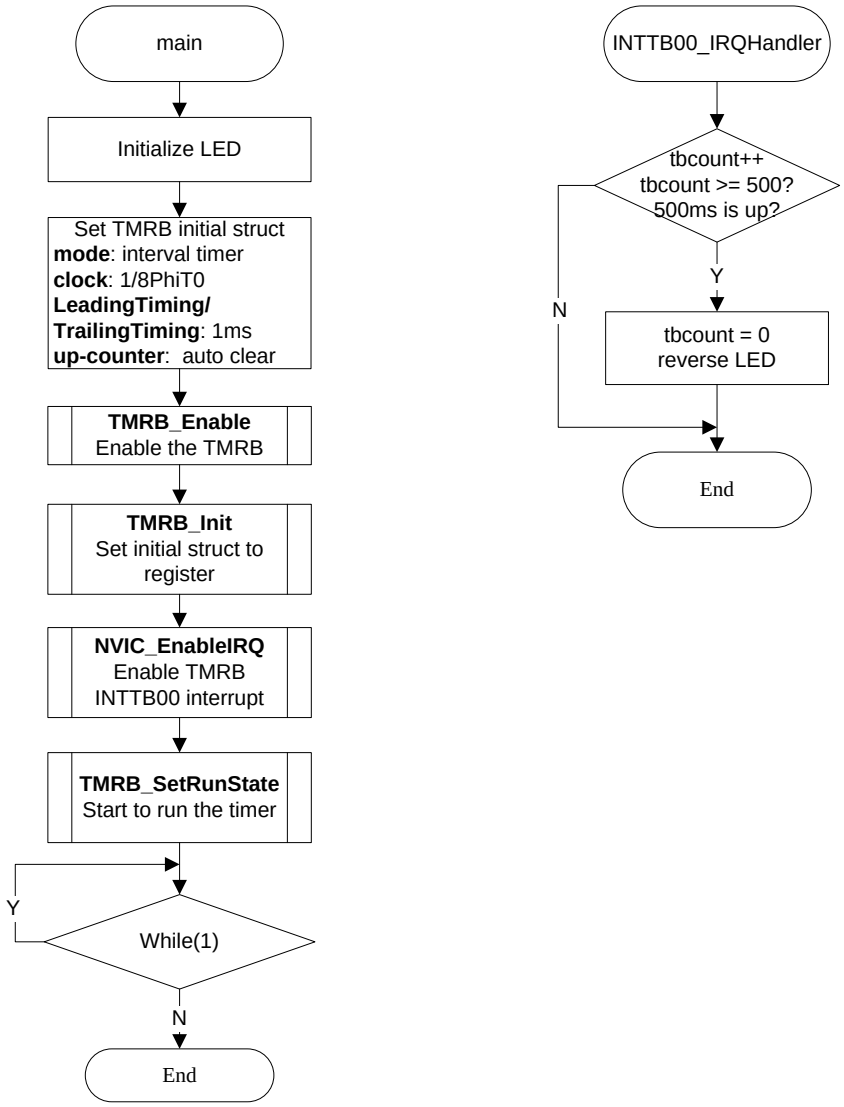
7-12-1　例: 汎用タイマ

ペリフェラルドライバ(TMRB, GPIO)を使用した簡単な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

- **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 を初期化し、LED を点灯します。

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

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。このサンプルでは、1ms の周期とデューティを設定します。TMRB_1MS マクロは、0x1388 です。(φT0=fsys=10MHz * PLL* = 40MHz, ftmrb = 1/8φT0 = 5MHz, Ttmrb = 0.2us, 1ms/0.2us = 5000 = 0x1388 (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_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。

```
static uint16_t tbcount = 0U;
static uint8_t ledon = 1U;
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 */
}
```

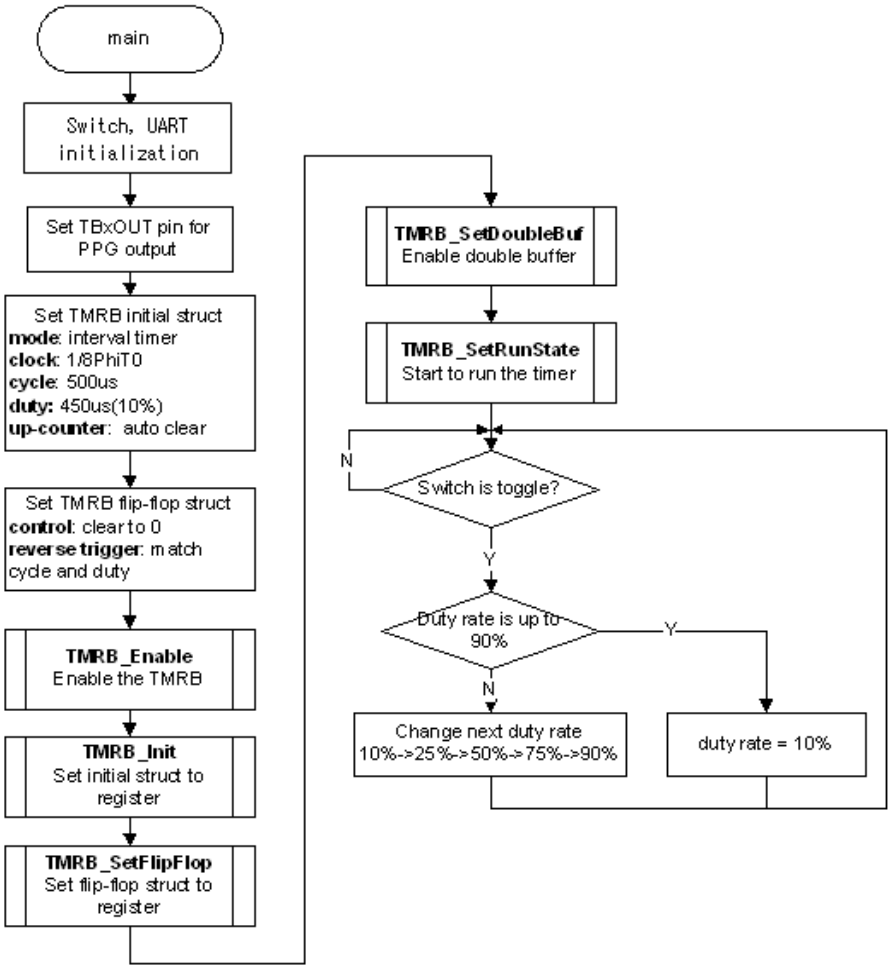
7-12-2 例: PPG 波形出力

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

この例では以下を行います。

1. TMRB6 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**

まず、スイッチとUARTの初期化を行い、PPG出力用にPA5をTB6OUTに設定します。

```
/* main.c – main function */
/* UART & switch initialization */
SW_Init();
hardware_init(UART_RETARGET);

/* Set PA5 as TB6OUT for PPG output */
GPIO_SetOutput(GPIO_PA, GPIO_BIT_5);
GPIO_EnableFuncReg(GPIO_PA, GPIO_FUNC_REG_2, GPIO_BIT_5);
```

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、TMRB6TIME マクロを定義してあります。このマクロは 0x09C4 です。(φT0 = fsys = fc = 10MHz * PLL * = 40MHz, ftmrb = 1/8 φT0 = 5MHz, Ttmrb = 0.2us, 500us/0.2us = 2500 = 0x09C4 (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.UpCntCtrl = TMRB_AUTO_CLEAR;
m_tmrb.TrailingTiming = TMRB6TIME;          /* TrailingTiming is 500us */
```

```
    m_tmrb.LeadingTiming = LeadingTiming[Rate]; /* LeadingTiming is 450us */
```

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、デューティー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_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_TB6);
    TMRB_Init(TSB_TB6, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB6, &PPGFFInital);
    /* enable double buffer */
    TMRB_SetDoubleBuf(TSB_TB6,ENABLE, TMRB_WRITE_REG_SEPARATE);
    TMRB_SetRunState(TSB_TB6, TMRB_RUN);
```

スイッチ状態の変化を待ち、現在のデューティ状態を LED に表示します。

```
    do {                            /* wait if switch is Low */
            keyvalue = SW_Get(SW1);
            LeadingTiming_display(); /* display current leading timing */
       } while (GPIO_BIT_VALUE_1 == keyvalue);
```

スイッチ状態が変化すると、下記のようにデューティーを設定します。
10%->25%->50%->75% ->90%その後 再び 90%から 10%になります。

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

    TMRB_ChangeLeadingTiming(TSB_TB6, LeadingTiming[Rate]);
```

デューティの算出方法:

- TrailingTiming = 500us, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us (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)
- LeadingTiming = 10%: High 幅は 500*10% = 50us, Low 幅は 500-50 = 450us, カウンタ値 = 450us/Ttmrb = 0x8CAU
- LeadingTiming = 25%: High 幅は 500*25% = 125us, Low 幅は 500-125 = 375us, カウンタ値= 375us/Ttmrb = 0x753U
- LeadingTiming = 50%: High 幅は 500*50% = 250us, Low 幅は 500-250 = 250us, カウンタ値= 250us/Ttmrb = 0x4E2
- LeadingTiming = 75%: High 幅は 500*75% = 375us, Low 幅は 500-375 = 125us, カウンタ値 = 125us/Ttmrb = 0x271
- LeadingTiming = 90%: High 幅は 500*90% = 450us, Low 幅は 500-450 = 50us, カウンタ値 = 50us/Ttmrb = 0xFA

これは上記計算式から求めたデューティ値の配列です。

```
/* leading timing: 10%, 25%, 50%, 75%, 90% */
uint32_t LeadingTiming[5] = { 0x8CAU, 0x753U, 0x4E2U, 0x271U, 0xFAU };
```

7-13 SIO/UART

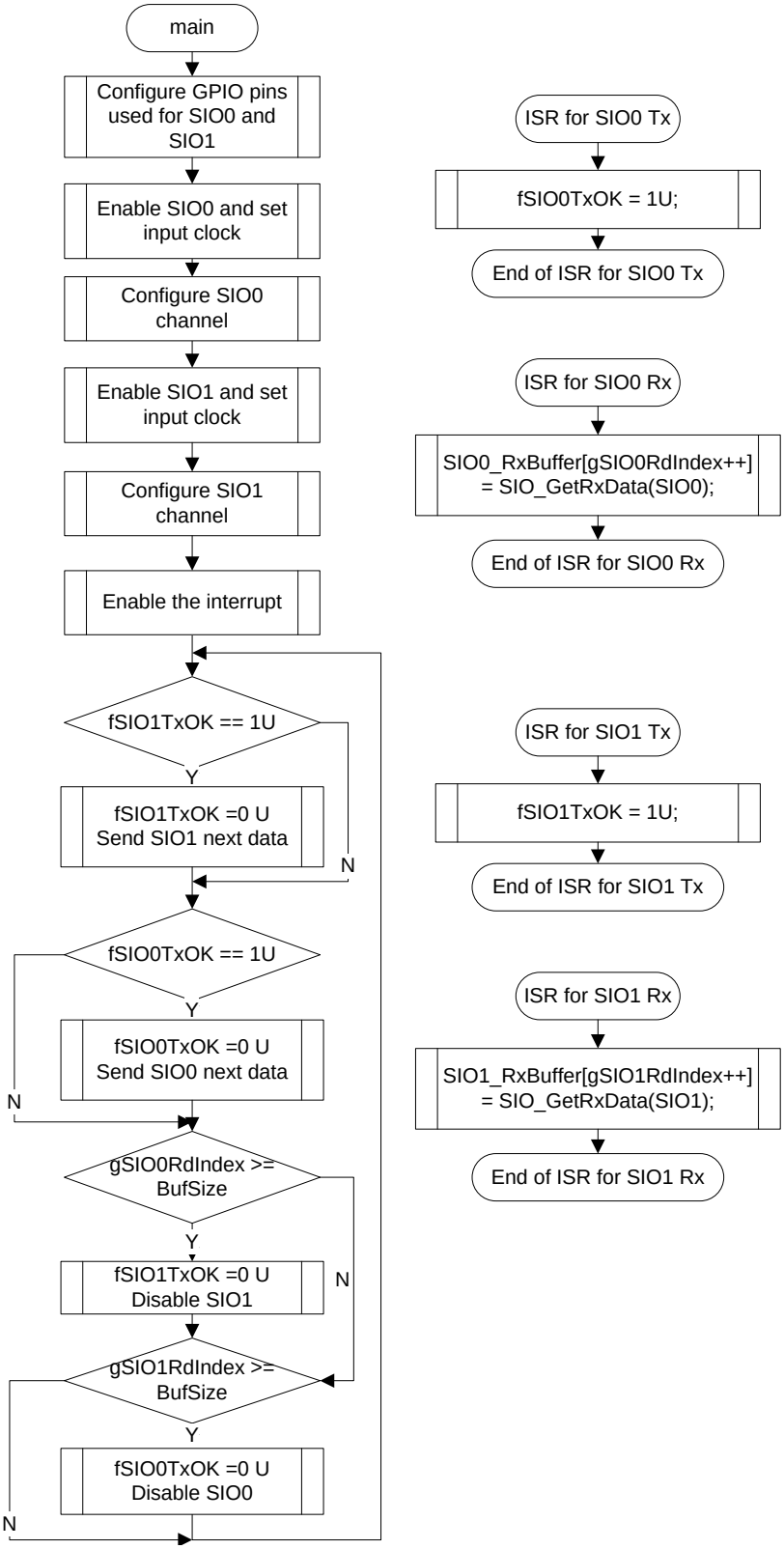
7-13-1 例: SIO

ペリフェラルドライバ (SIO)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. SIO の基本設定
2. SIO0⇔SIO1 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 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);

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

SIO1 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 SIO1 channel */
    SIO_Enable(SIO1);

    /*initialize the SIO1 struct */
    SIO1_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
    SIO1_Init.TransferMode = SIO_TRANSFER_FULLDPX;
    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 の送受信割り込みを有効にします。

```
    /* 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);
```

すべての基本設定を行い、その後、データ送信を行います。

```
 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 */
        }
    }
```

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);
}
```

SIO1 送信の ISR にて、転送設定を行います。

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

SIO1 受信の ISR にて、受信バッファからデータを受信します。

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

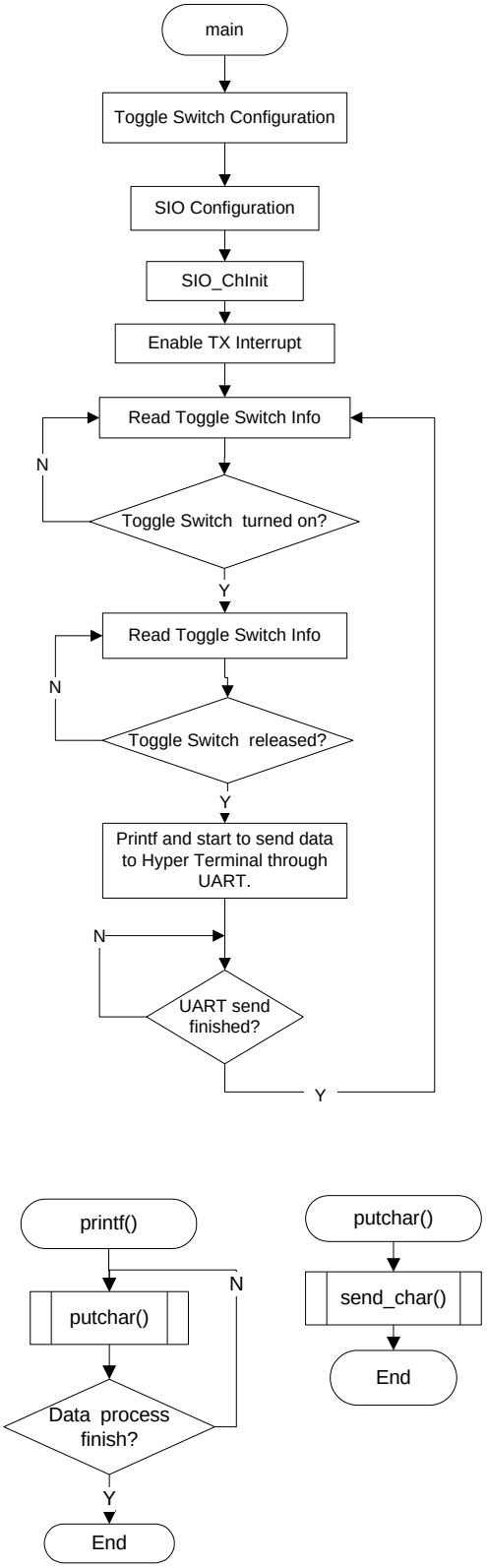
7-13-2　例: リターゲット

ペリフェラルドライバ(UART、GPIO)を使用した簡単な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 UART の TX 割り込みを使用
4. UART 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**

  まず、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、UART0 を初期化します。
  GPIO ペリフェラルドライバを使用し、GPIO をトグルスイッチ用に設定します。

```
    GPIO_SetInput(GPIO_PH, GPIO_BIT_4 | GPIO_BIT_5 | GPIO_BIT_6 |
GPIO_BIT_7);
    GPIO_SetPullDown(GPIO_PH, GPIO_BIT_4 | GPIO_BIT_5 | GPIO_BIT_6 |
GPIO_BIT_7, ENABLE);
```

次に GPIO ペリフェラルドライバを使用し、GPIO を UART0 用に設定します。

```
    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。

```
    UART_InitTypeDef myUART;

    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 ペリフェラルドライバを使用し、UART0 の許可と初期化を行います。

```
    SIO_Configuration(UART_RETARGET);
    SIO_ChInit();
```

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)
```

その後、トグルスイッチをリードします。

```
    SW_info = SW_Get(SW_PORT);
```

トグルスイッチが ON の場合、トグルスイッチが OFF になるまでウェイトします。

```
    if (SW_info == 1U) {
        /* wait for switch SW0 released */
        do {
            wait_SW0 = 0U;
            SW_info = SW_Get(SW_PORT);
            delay();
            } while (SW_info != SWRELEASE);
     }
```

その後データ送信を開始します。ここでの TxBuffer は文字列です。

```
    printf("%s\r\n", TxBuffer);        /* SIO0 send data */
```

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;
}
```

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です。

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;		/* SIO1 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 */
	}
}
```

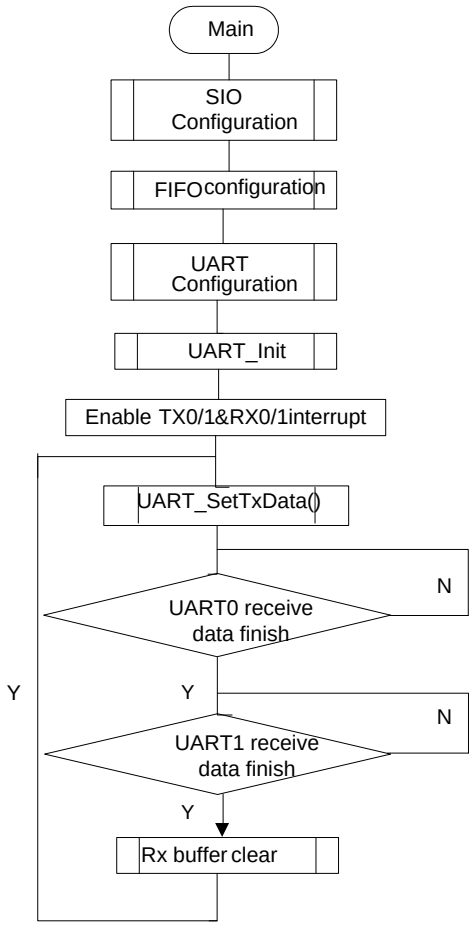
7-13-3　例: UART FIFO

ペリフェラルドライバ UART と 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. 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) {
		/*SET PE0 AS TXD0 */
		TSB_PE->FR1 |= 1U ;
		TSB_PE->CR |=  1U ;

		/*SET PE1 AS RXD0 */
		TSB_PE->FR1 |= (1U << 1U);
		TSB_PE->IE |= (1U << 1U);
	} else if (SCx == TSB_SC1) {
		/*SET PA5 AS TXD1 */
		TSB_PA->FR1 |= (1U << 5U);
		TSB_PA->CR |= (1U << 5U);

		/*SET PA6 AS RXD1 */
		TSB_PA->FR1 |= (1U << 6U);
		TSB_PA->IE |= (1U << 6U);
	}
```

```
}
```

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;

myUART.BaudRate = 115200U;
myUART.DataBits = UART_DATA_BITS_8;
myUART.StopBits = UART_STOP_BITS_1;
myUART.Parity = UART_NO_PARITY;
myUART.Mode = UART_ENABLE_TX|UART_ENABLE_RX;
myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
```

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。
UART0 の送信割り込み処理ルーチン:

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

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

UART1 の送信割り込み処理ルーチン:

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

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

UART0 の受信割り込み処理ルーチン:

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

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

UART1 の受信割り込み処理ルーチン:

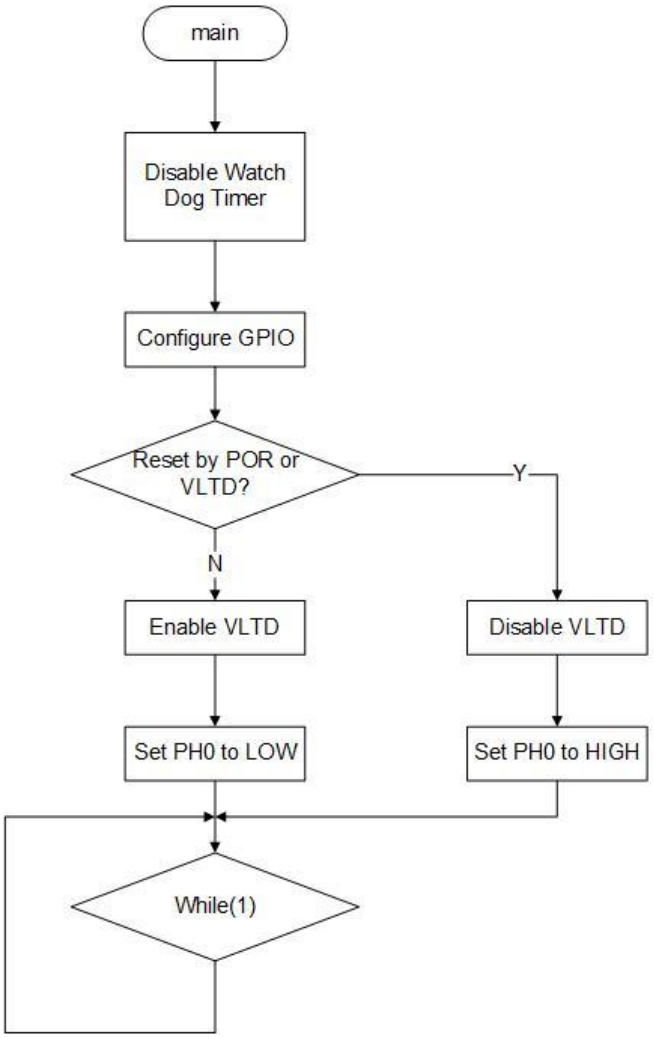
```
void INTRX1_IRQHandler(void)
{
    volatile UART_Err err;

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

7-14 VLTD

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。

この例では以下を行います。
1.VLTD の設定

- **フローチャート**



- **サンプルプログラムのコードと説明**

最初にシステムを初期化します。不用意なWDT 割割り込みを避けるためWDTを無効にします。その後 PH0 を出力ポートに設定します。

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

リセットフラグをリードし、VLTD によるリセットか、またはPOR によるリセットの場合は、検出電圧の設定と VLTD 動作を許可し、PH0 から Low 出力を行います。

```
if (CG_GetResetFlag().Bit._PowerOn == 0U) {
        VLTD_Enable();
```

```
            GPIO_WriteData(GPIO_PH, 0x00U);
```

リセットフラグの要因が上記以外の場合、VLTD は無効にし、PH0 から High 出力を行います。

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

7-15 WDT

ペリフェラルドライバ (WDT, 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. 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();
```