TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM366)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M366UE-01J

本製品取り扱い上のお願い

- ソフトウエア使用権許諾契約書の同意無しに使用しないで下さい。

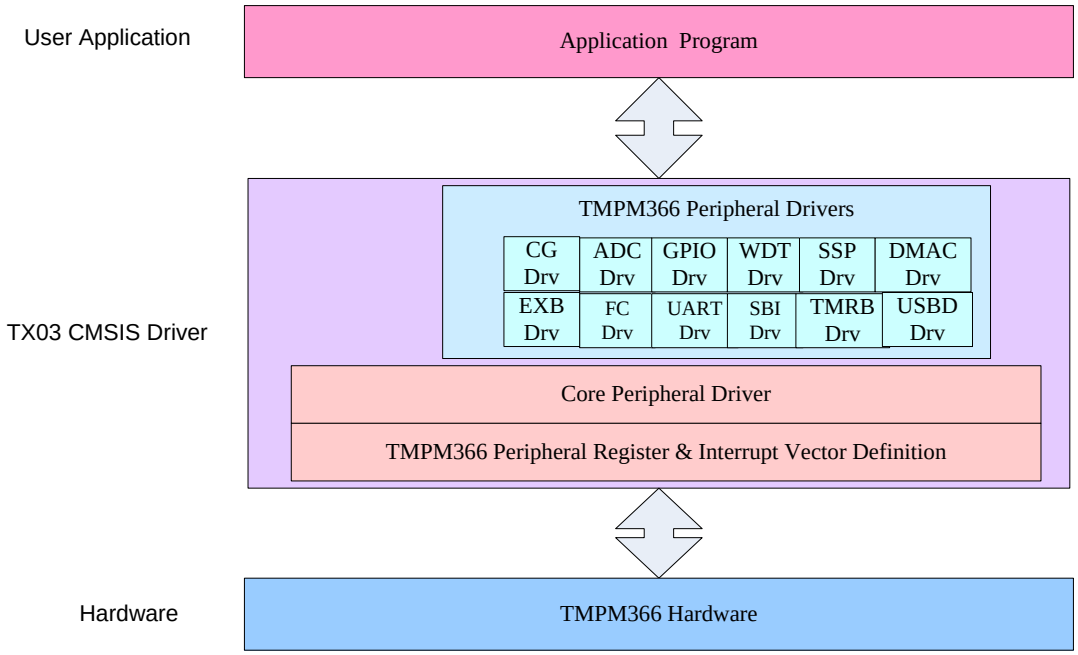
目次

1 はしがき

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TMPM366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/チャネル	使用／未使用
CG	クロックギア	使用(CG サンプル)
	PLL	8 逓倍
低消費電力モード	-	STOP1 モード
SysTick	-	未使用
ウォッチドッグタイマ (WDT)	-	使用(WDT サンプル)
外部割込み (INT)	INT0	使用(CG サンプル)
	上記以外のチャネル	未使用
シリアルチャネル (SIO/UART)	SIO0	使用(UART&DMAC サンプル: Tx)
	SIO1	使用(UART&DMAC サンプル: Rx)
同期式シリアルインタフェース(SSP)	SSP0	使用(SSP サンプル: Tx)
	SSP1	使用(SSP サンプル: Rx)
	SSP2	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)
DMAC	-	使用(DMAC サンプル)
16 ビットタイマ/イベントカウンタ(TMRB)	TMRB0	使用(TMRB: 汎用タイマサンプル)
	TMRB4	使用(TMRB: PPG 波形出力サンプル)
	上記以外のチャネル	未使用
10ビット A/Dコンバータ (ADC)	AIN11	使用(ADC サンプル)
	上記以外のチャネル	未使用
外部バスインタフェース (EXB)	CS0	未使用
	CS1	使用(SRAM リード/ライトサンプル)

4 端子用途

本サンプルプログラムは、IAR TMPM366-SK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。

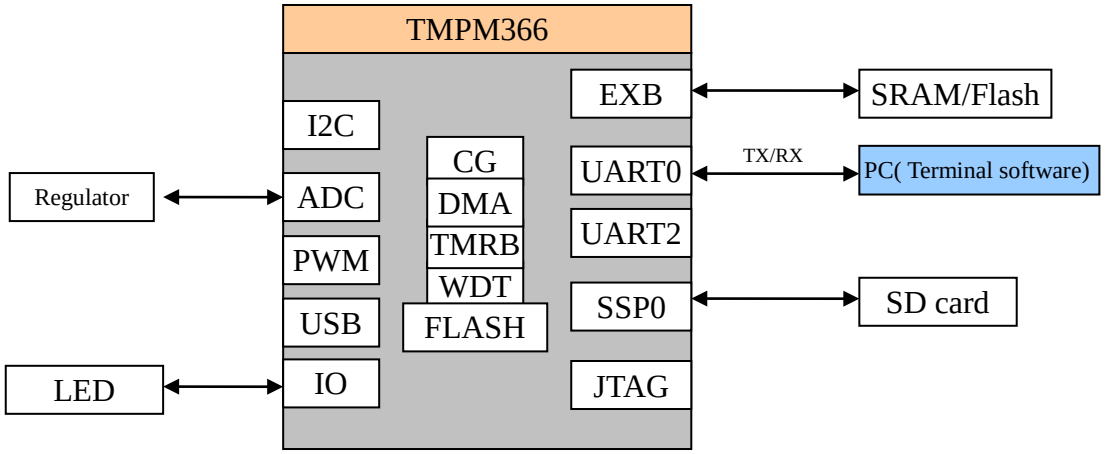
No	Name	Usage
1	AVDD3	+3.3V
2	AVSS	GND
3	AVREFH	+3.3V
4	AVREFL	GND
5	DVSSA	GND
6	DVDD3A	+3.3V
7	TRST, PI7	未使用
8	TDI, PI6	未使用
9	TDO, SWV, PI5	未使用
10	TMS, SWDIO, PI4	未使用
11	TCK, SWCLK, PI3	未使用
12	TRACECLK, PI2	未使用
13	TRACEDATA0, PI1	未使用
14	TRACEDATA1, PI0	未使用
15	TRACEDATA2, PH0	未使用
16	TRACEDATA3, PH1	未使用
17	DCD02, TB4OUT, A10, PH2	PPG 出力
18	DSR02, TB5OUT, A9, PH3	未使用
19	DTR02, INT8, A8, PH4	未使用
20	USBPON, INT1, PG5	未使用
21	RTS02, TB4IN1, A7, PG4	未使用
22	RIN02, TB4IN0, A6, INT0, PG3	INT0
23	CTS2, TB3IN1, A5, SCK0, PG2	未使用
24	IRIN, RX02, TB3IN0, A4, SCL0, SI0, PG1	SCL0
25	IROUT, TX02, A3, SDA0, SO0, PG0	SDA0
26	FTEST3	未使用
27	PC0, TXD1, A2, TB2IN0	TXD1
28	PC1, RXD1, A1, TB2IN1	RXD1
29	PC2, SCLK1, A0, TB0OUT, CTS1	未使用

No	Name	Usage
30	PF0, BOOT, TB6OUT	未使用
31	PF1, RD	RD
32	PF2, WR	WR
33	PF3, BELL	BELL
34	PF4, BELH, INT6, TB5IN0	BELH
35	PF5, CS1, INT7, TB5IN1	CS1
36	PF6, CS0	CS0
37	PF7, ALE	ALE
38	DVSSA	GND
39	DVDD3A	+3.3V
40	PA0, AD0, D0	LED1/ AD0
41	PA1, AD1, D1	LED2/ AD1
42	PA2, AD2, D2	LED3/ AD2
43	PA3, AD3, D3	LED4/ AD3
44	PA4, AD4, D4	LED5/ AD4
45	PA5, AD5, D5	LED6/ AD5
46	PA6, AD6, D6	LED7/ AD6
47	PA7, AD7, D7	LED8/ AD7
48	PB0, AD8, D8, SP1DO, A0	SP1DO/ AD8
49	PB1, AD9, D9, SP1DI, A1	SP1DI/ AD9
50	PB2, AD10, D10, SP1CLK, A2	SP1CLK/ AD10
51	PB3, AD11, D11, SP1FSS, A3	SP1FSS/ AD11
52	PB4, AD12, D12, SP2DO, A4	AD12
53	PB5, AD13, D13, SP2DI, A5	AD13
54	PB6, AD14, D14, SP2CLK, A6	AD14
55	PB7, AD15, D15, SP2FSS, A7	AD15
56	DVDD3A	+3.3V
57	DVSSA	GND
58	PD7, SP0FSS, SCOUT	SP0FSS
59	PD6, SP0CLK	SP0CLK
60	PD5, SP0DI	SP0DI
61	PD4, SP0DO	SP0DO
62	DVSS3C	GND
63	D-	未使用
64	D+	未使用
65	DVDD3C	+3.3V
66	PD3, A19, ADTRG	A19
67	PD2, A18, TB9OUT	A18
68	PD1, A17, TB8OUT	PSW1/ A17

No	Name	Usage
69	PD0, A16, TB7OUT	PSW2/ A16
70	PE0, TXD0, A20	TXD0/ A20
71	PE1, RXD0, A21	RXD0/ A21
72	PE2, SCLK0, TB2OUT, CTS0, A22	A22
73	PE3, INT5, A15, TB3OUT, A23	A23
74	MODE	GND
75	RESET	RESET
76	X2	12MHz 発振器
77	DVSS	GND
78	X1	12MHz 発振器
79	NMI	未使用
80	A14, SDA1, SO1, PE4	SDA1
81	A13, SCL1, SI1, PE5	SCL1
82	A12, SCK1, PE6	未使用
83	A11, INT4, PE7	未使用
84	DVDD3A	+3.3V
85	DVDD3A	+3.3V
86	DVSSA	GND
87	RVSS	GND
88	RVDD	+3.3V
89	AIN00, PJ0	未使用
90	AIN01, PJ1	未使用
91	AIN02, PJ2	未使用
92	AIN03, PJ3	未使用
93	AIN04, PJ4	未使用
94	AIN05, PJ5	未使用
95	TB0IN0, AIN06, PJ6	未使用
96	TB0IN1, INT9, AIN07, PJ7	未使用
97	TB1IN0, INT2, AIN08, PK0	未使用
98	TB1IN1, INT3, AIN09, PK1	未使用
99	TB6IN0, AIN10, PK2	未使用
100	TB6IN1, AIN11, PK3	AIN11

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:
   IAR TMPM366-SK ボード

2. 開発ツール:
   - IAR:
     1) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
     2) IDE: IAR Embedded workbench 6.30.1 version
   - KEIL:
     1) u-Link: Realview ULINK2
     2) IDE: KEIL uVision MDK version 4.14

6 機能説明

6-1 動作モード

本デバイスには 4 つの動作モードがあります: NORMAL, IDLE, STOP1, STOP2 モード
IDLE と STOP1、STOP2 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、STOP2 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:**

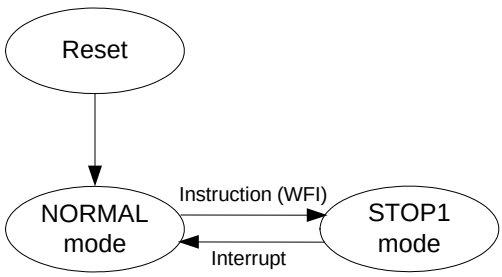
このモードは、CPU コアおよび周辺ハードウェアを高速クロックで動作させるモードです。リセット解除後は NORMAL モードになります。

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、450μs 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

6-2-1 ADC データリード

PK3/AIN11 に接続された歩テンションメータの値を変更します。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 にり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。

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、TMPM366 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。

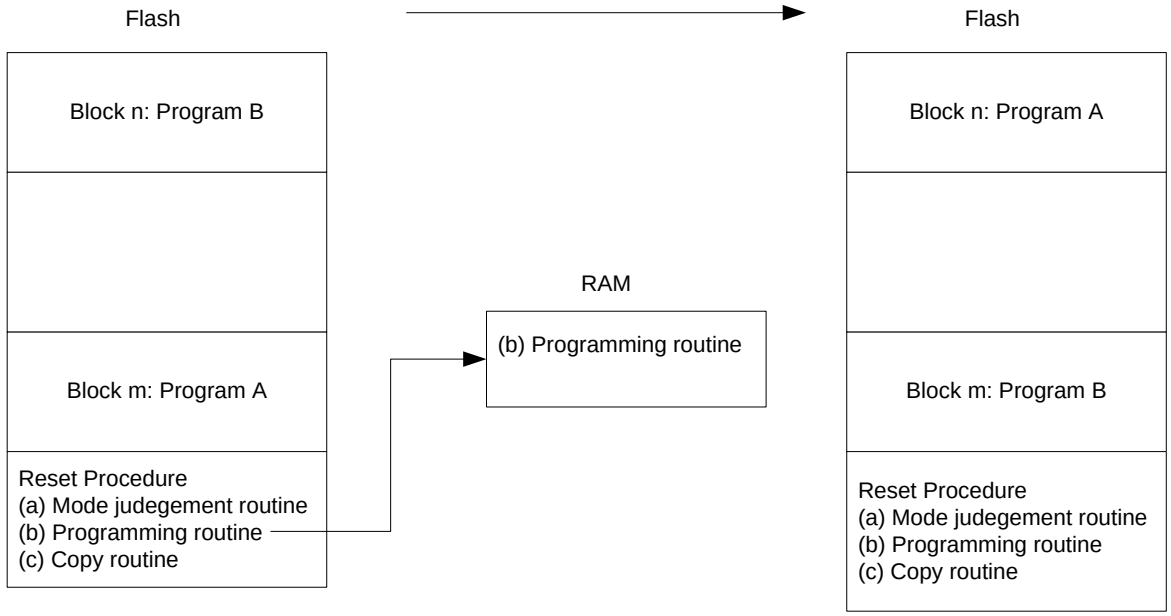
6-5 EXB

6-5-1 SRAM リード/ライト

この機能は評価ボードの外部 SRAM をリード/ライトでき、マルチプレクスバスモードでは 16 ビットバスでセットされます。SRAM の A.C スペック (cycles time) は、SRAM チップのデータシートを参照してください。ここでは外部 SRAM として 1M バイトの IS62WV51216BLL を使用します。

6-6 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: LED1 点灯, B: LED2 点灯)
初期状態は、プログラム A が最初に動作します。
　ーSW2 キーはリセット動作のモード判定に使います。
SW2 キーが押されていない場合 → Normal モードです。
SW2 キーが押された場合 → ユーザブートモードです。

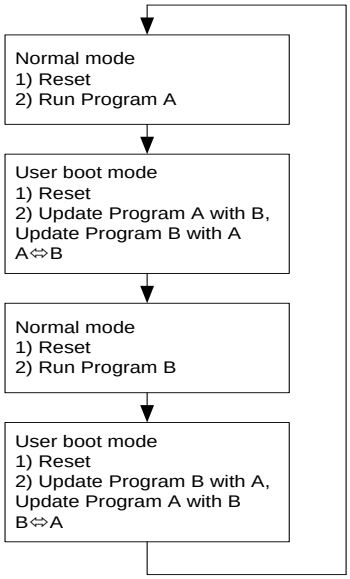


動作シーケンス

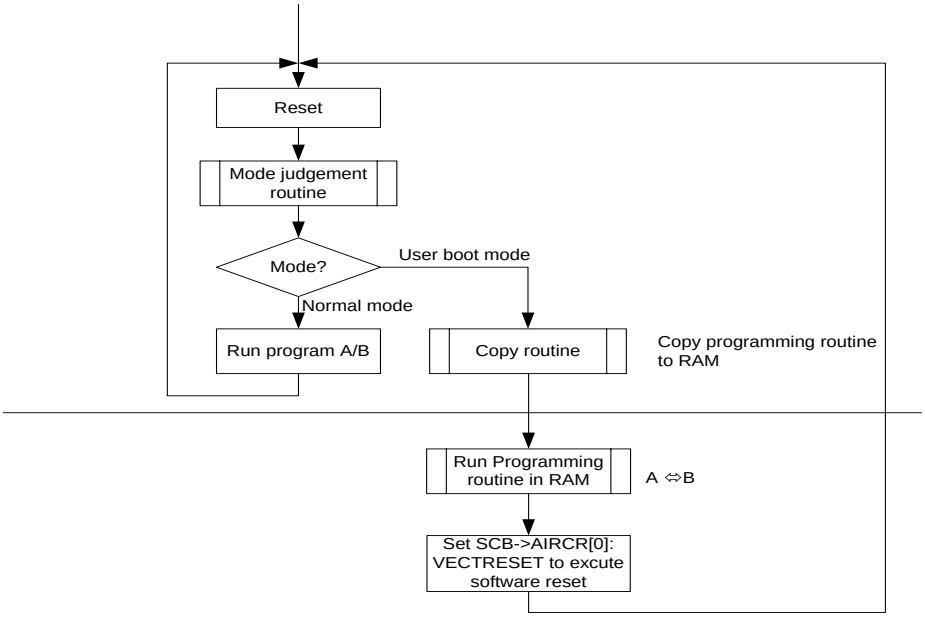
(1) 電源投入
SW2 を OFF した状態で電源投入またはリセット端子を ON してください。まずプログラム A が実行され、LED1 が点灯します。

(2) SW2 を ON している間にリセット端子を ON してください。下記の処理を行います。
スワップ処理: プログラムルーチンを RAM に転送し、その後プログラム A と B をスワップします。LED は LED4->LED3->LED2 の順で点灯します。
スワップ成功: LED3 と LED4 が点灯します。

(3) スワップ完了後、SW2 を OFF すると、システムリセットを行います。
LED2 を点滅し、プログラム B が動作します。



・サンプルプログラムのフローチャート



サンプルプログラムがユーザーブートモードに入ると、LED は以下の状態に変化します。

LED4 点灯
(ユーザブートモード)

RAM 転送もしくはフラッシュ書き込み中は、LED に現在の状態を出力します。

LED の状態例:

LED3 点灯	LED2 点灯	LED3/LED4 点灯
(RAM 転送中)	(A と B のスワップ中)	(スワップ完了、リセット待ち)

6-7 FUART

本プログラムにおいて、フル UART 送信と FIFO 受信が可能です。
本プログラムは 64 種の異なるデータをフル UART チャネル 0 の TX02 端子から送信し、RX02 端子からデータを受信します。スイッチ SW1 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 SW1 が数回オフ、オンとなった場合、本プログラムは RX02 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、LED1,2,3,4,5,6,7,8 を点滅します。
受信データが送信データと異なる場合、LED1,3,5,7 を点滅します。
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。
1 回目:
受信データが送信データと同一であるなら、RTS と CTS ハードウェアフロー制御をイネーブルにしてください。
2 回目:
受信データが送信データと異なるなら、RTS と CTS ハードウェアフロー制御をディセーブルにしてください。

補足: TMPM366 SK ボードの TX02 端子と RX02 端子を接続してください。
TMPM366 SK ボードの RTS 端子と CTS 端子を接続してください。

6-8 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。

6-9 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 gI2CrxData[]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 TMPM366 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。

6-10 SIO/UART

6-10-1 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。

6-10-2 UART FIFO

UART0 から"TMPM3661"というデータを FIFO を使用して UART1 へ送信し、同時に UART1 から"TMPM3662"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=”TMPM3662” と RxBuffer1="TMPM3661"となった場合にブレークポイントで停止します。

6-10-3 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)

6-11 SSP

6-11-1 使用した SSP0->SSP1 転送

SSP0はSPIマスタとして、SSP1はSPIスレーブとして設定します。その後端子接続を行います。送受信時のデータ加算にDMAを使用します。

補足: TMPM366-SK ボードを使用した端子接続状態

	SSP0 pins	補足	SSP1 端子
1	PD7/SP0FSS	PD7 - PB3	PB3/SP1FSS
2	PD6/SP0CLK	PD6 - PB2	PB2/SP1CLK
3	PD5/SP0DI	PD5 - PB0	PB1/SP1DI
4	PD4/SP0DO	PD4 - PB1	PB0/SP1DO

6-11-2 SSP0 セルフループバック

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かどうかの確認を行います。

補足: SDカードソケットは何も入っていない状態にしてください。

6-12 TMRB

6-12-1 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。

6-12-2 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%

6-13 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、TMPM366 MCU の主要機能を TMPM366-SK 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:

IAR EWARM:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm366_wdt_int.c
        ├─TX03_CMSIS
        │   │   system_TMPM366.c
        │   │   system_TMPM366.h
        │   │   TMPM366.h
        │   │
        │   └─startup
        │         startup_TMPM366.s
        │
        ├─TX03_Periph_Driver
        │   ├─inc
        │   │   tmpm366_wdt.h
        │   │   tmpm366_gpio.h
        │   │   tx03_common.h
        │   │
        │   └─src
        │         tmpm366_wdt.c
        │         tmpm366_gpio.c
        └─TMPM366-SK
                led.c
                led.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm366_wdt_int.c
        │
```

```
├─TX03_CMSIS
│      system_TMPM366.c
│      startup_TMPM366.s
│
├─TX03_Periph_Driver
│      tmpm366_wdt.c
│      tmpm366_gpio.c
│
└─TMPM366-SK
        led.c
```

7-1 ADC

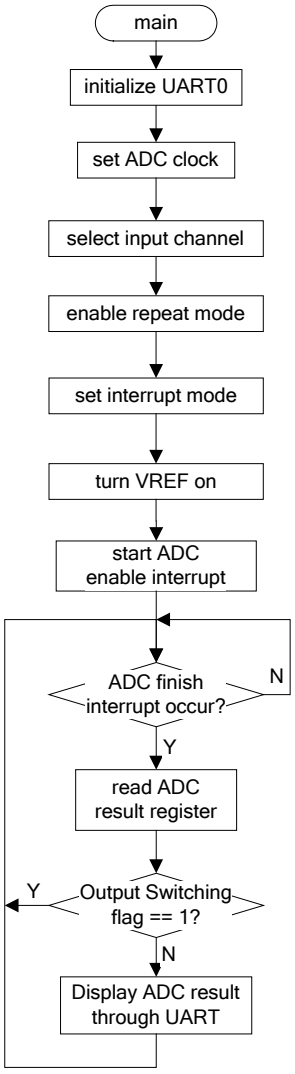
7-1-1 例: ADC データリード

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、チャネル固定シングル変換の設定と AD チャネルを選択します。その後、VREF を ON します。

```
/* Enable ADC clock supply */
ADC_SetClkSupply(ENABLE);

/* set ADC clock */
ADC_SetClk(ADC_CONVERSION_CLK_80, ADC_FC_DIVIDE_LEVEL_8);

/* select ADC input channel */
ADC_SetInputChannel(ADC_AN_11);

/* Enable ADC repeat mode */
ADC_SetRepeatMode(ENABLE);

/* Set interrupt mode */
ADC_SetINTMode(ADC_INT_CONVERSION_8);
```

```
/* Turn VREF on */
ADC_SetVref(ENABLE);

/* Wait at least 3us to ensure the voltage is stable */
Delay(10U);
```

AD 変換を開始します。

```
    ADC_Start();

/* enable AD interrupt */
NVIC_EnableIRQ(INTAD_IRQn);
```

AD 変換開始後、INTAD 割り込みの発生を待ちます。INTAD 割り込み発生後、ADC_Display()関数をコールします。

```
while (1) {
    if (fIntADC == 1U) {
        fIntADC = 0U;
        ADC_Display();
    }
}
```

ADC_Display()関数では、AD 変換結果を取得するため ADREG レジスタをリードし、AIN 兼用ポートの出力スイッチングフラグを確認します。

```
ADC_Result result = ADC_GetConvertResult(ADC_REG_00);

/* If OutputSwitching flag is set to 1, the accuracy of the result may be affected */
if (result.Bit.OutputSwitching == 1U) {
    return; /* discard this result */
}
```

7-2 CG

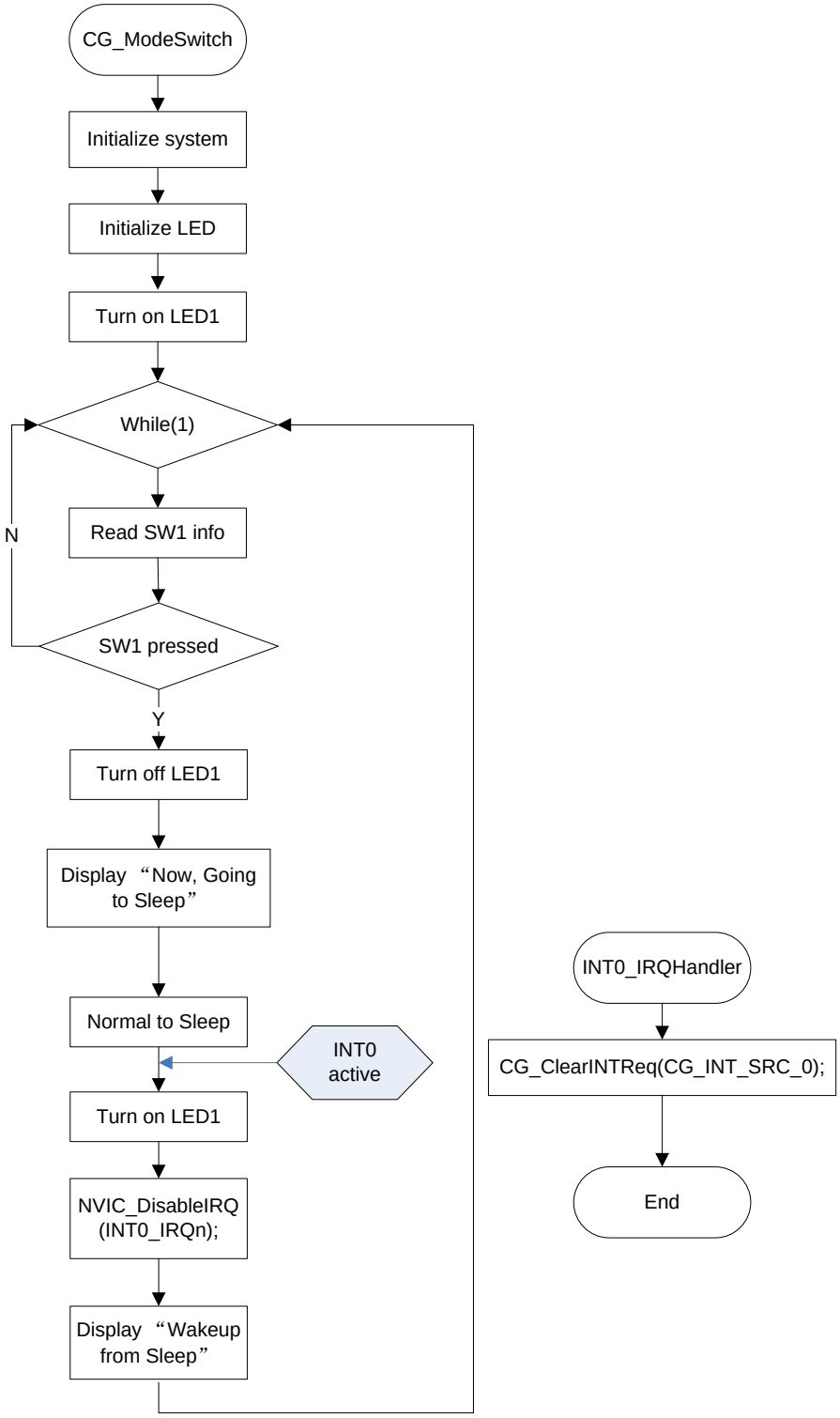
7-2-1 例: Power モード変更

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 */
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

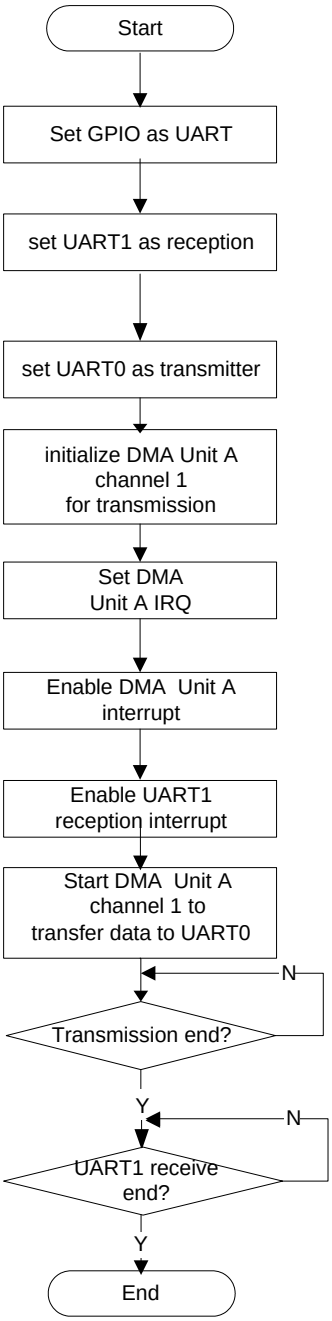
7-3-1 例: メモリから周辺回路

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_SetDMAChannel(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_T
  X_END);
    } else {
        /* Do nothing */
    }
} else {
    /* Do nothing */
}
```

7-4 EXB

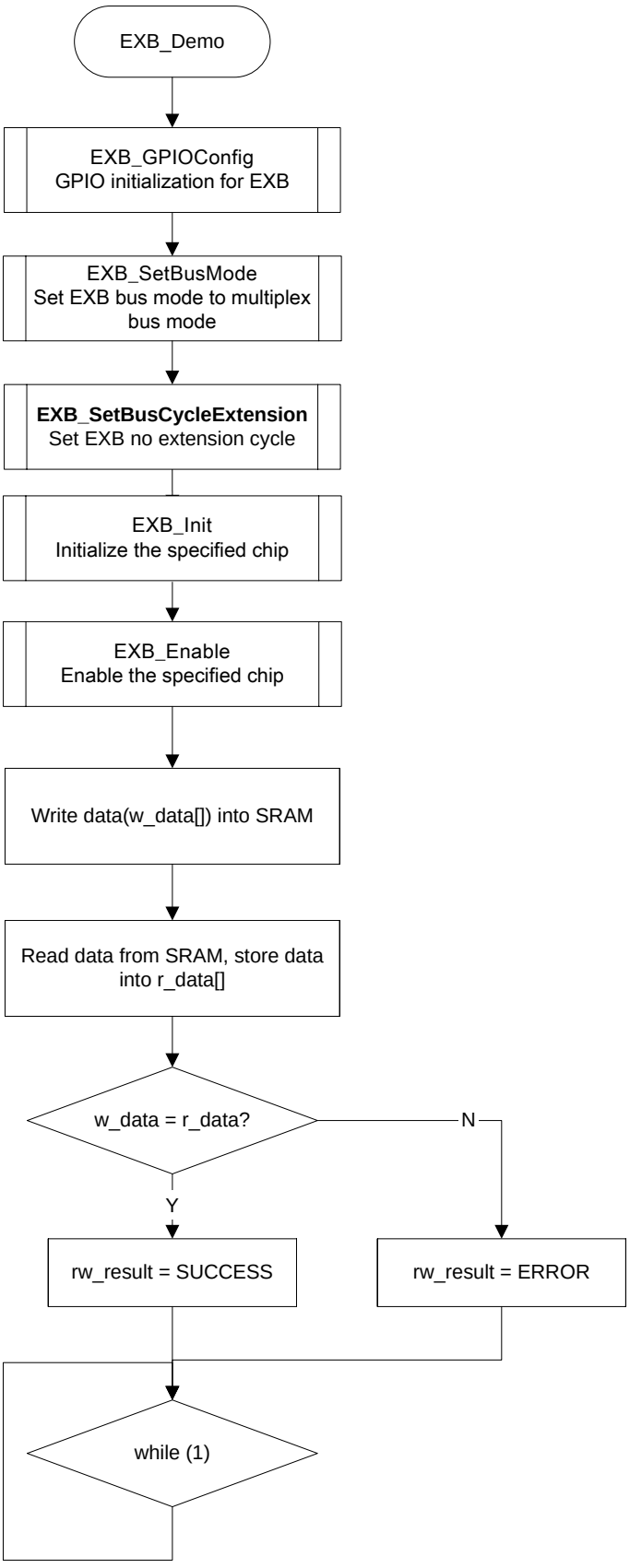
7-4-1 例: SRAM のリード/ライト

ペリフェラルドライバ(EXB, 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. EXB の初期設定
2. 外部 SRAM のリード/ライト

- **フローチャート:**


EXB_Demo
EXB_GPIOConfig
GPIO initialization for EXB
EXB_SetBusMode
Set EXB bus mode to multiplex bus mode
EXB_SetBusCycleExtension
Set EXB no extension cycle
EXB_Init
Initialize the specified chip
EXB_Enable
Enable the specified chip
Write data(w_data[]) into SRAM
Read data from SRAM, store data into r_data[]
w_data = r_data?
N
Y
rw_result = SUCCESS
rw_result = ERROR
while (1)

- **サンプルプログラムのコードと説明**

まず、EXB の初期化を行います。

```
	uint8_t chip = EXB_CS1;
	uint8_t BusMode = EXB_BUS_MULTIPLEX;
	uint8_t Cycle = EXB_CYCLE_NONE;

#ifdef SRAM_RW
	uint32_t w_data[TEST_DATA_LEN] = { 0U };
	uint32_t r_data[TEST_DATA_LEN] = { 0U };
	uint16_t rw_cnt = 0U;
	uint32_t *addr = NULL;
	uint16_t i = 0U;
#endif

	EXB_InitTypeDef InitStruct = { 0U };

	InitStruct.AddrSpaceSize = EXB_1M_BYTE;
	InitStruct.StartAddr = 0x00;
	InitStruct.BusWidth = EXB_BUS_WIDTH_BIT_16;
	/* Set cycles time according to AC timing of SRAM datasheet,base clock:
EXBCLK(fsys) */
	InitStruct.Cycles.InternalWait = EXB_INTERNAL_WAIT_2;
	InitStruct.Cycles.ReadSetupCycle = EXB_CYCLE_1;
	InitStruct.Cycles.WriteSetupCycle = EXB_CYCLE_1;
	InitStruct.Cycles.ALEWaitCycle = EXB_CYCLE_1;
	InitStruct.Cycles.ReadRecoveryCycle = EXB_CYCLE_1;
	InitStruct.Cycles.WriteRecoveryCycle = EXB_CYCLE_1;
	InitStruct.Cycles.ChipSelectRecoveryCycle = EXB_CYCLE_1;
```

EXB の設定と GPIO の設定を行い、その後 CS を有効にします。SRAM への書き込みデータを w_data[]に設定し、その後、SRAM からの読み出しデータを r_data[]へ書き込みます。SRAM ライト/リードがうまく行ったかどうかは rw_result を確認します。

```
#ifdef SRAM_RW
	EXB_GPIOConfig();
#endif

	EXB_SetBusMode(BusMode);
	EXB_SetBusCycleExtension(Cycle);
	EXB_Init(chip, &InitStruct);
	EXB_Enable(chip);

#ifdef SRAM_RW
	/* SRAM Read/Write demo */
	addr = (uint32_t *) (((uint32_t) InitStruct.StartAddr) | EXB_SRAM_START_ADDR);

	for (i = 0U; i < TEST_DATA_LEN; i++) {
		w_data[i] = i;
	}

	rw_cnt = TEST_DATA_LEN * sizeof(w_data[0]);
	memcpy(addr, w_data, (uint32_t) rw_cnt);
	__DSB();
```

```
    memcpy(r_data, addr, (uint32_t) rw_cnt);

    /* check rw_result to see if SRAM write/read is sucessful or not */
    if (memcmp(w_data, r_data, (uint32_t) rw_cnt) == 0U) {
        rw_result = SUCCESS;
    } else {
        rw_result = ERROR;
    }
#endif
    while (1) {
        /* Do nothing */
    }
```

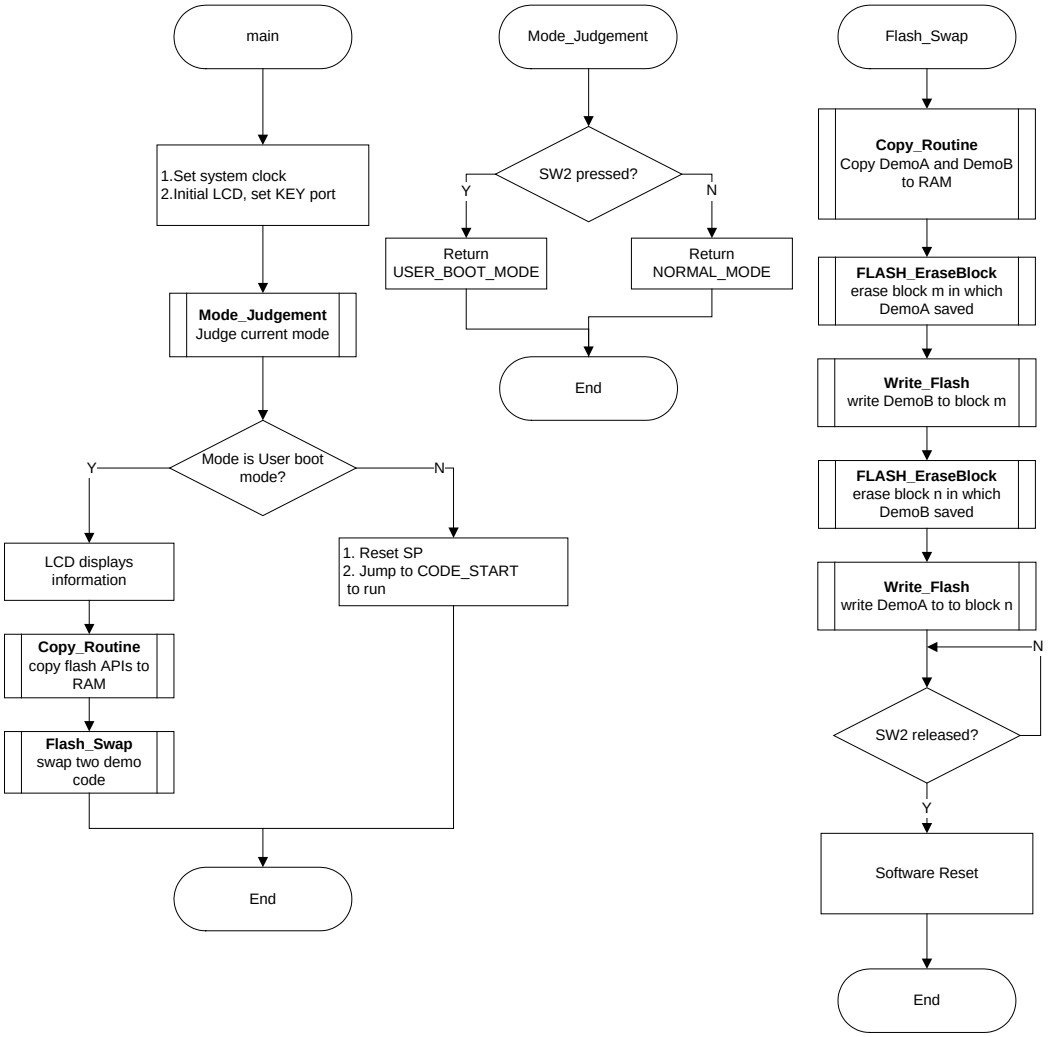
7-5 FLASH

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。

```
CG_SetFcSrc (CG_FC_SRC_FOSC);                  /* Select fosc */
CG_SetPLL (DISABLE);                           /* Disable PLL */
GPIO_SetInput (GPIO_PD, GPIO_BIT_0);           /* set port D to input */
LEDInit ();                                    /* LED initialization */
```

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 (GPIO_ReadDataBit (GPIO_PD, GPIO_BIT_0) ==
            GPIO_BIT_VALUE_0)? USER_BOOT_MODE: NORMAL_MODE;
}
```

リセット時に SW2 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(LED1 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
            startup = CODE_START;
            startup();                  /* jump to code start address to run */
```

リセット時に SW2 が押されている場合、ユーザブートモードになります(LED4 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。RAM 転送中は LED3 が点灯します。

```
    Status_Display (0x08U);       /* status 1: enter user boot mode */
    Copy_Routine (FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
                                                /* copy flash API to RAM */
     Status_Display (0x04U);       /* status 2: RAM transferring */
```

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。サンプル A とサンプル B のスワップ中は LED2 が点灯します。

```
    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 */

    Status_Display(0x02U);              /* status 2: swap the demo */

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

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

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

    /* write A to block n */
```

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

スワップ完了後、LED3 と LED4 が点灯します。SW が離されると、SCB->AIRCR 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(LED2 が点滅)。

```
Status_Display(0x0CU);                    /* status 3: complete and restart */

while (GPIO_ReadDataBit(GPIO_PD, GPIO_BIT_0) == GPIO_BIT_VALUE_0) {
            /* wait for SW2 release */
}
reg_value = SCB->AIRCR;        /* software reset */
reg_value = (uint32_t) 0x05FA0001;
SCB->AIRCR = reg_value;
```

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;
            *addr1 = FC_RESET_CMD;          /* Reset FLASH */
             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 4~7 */
        *addr3 = *source;
        source++;
    }
```

7-6 FUART

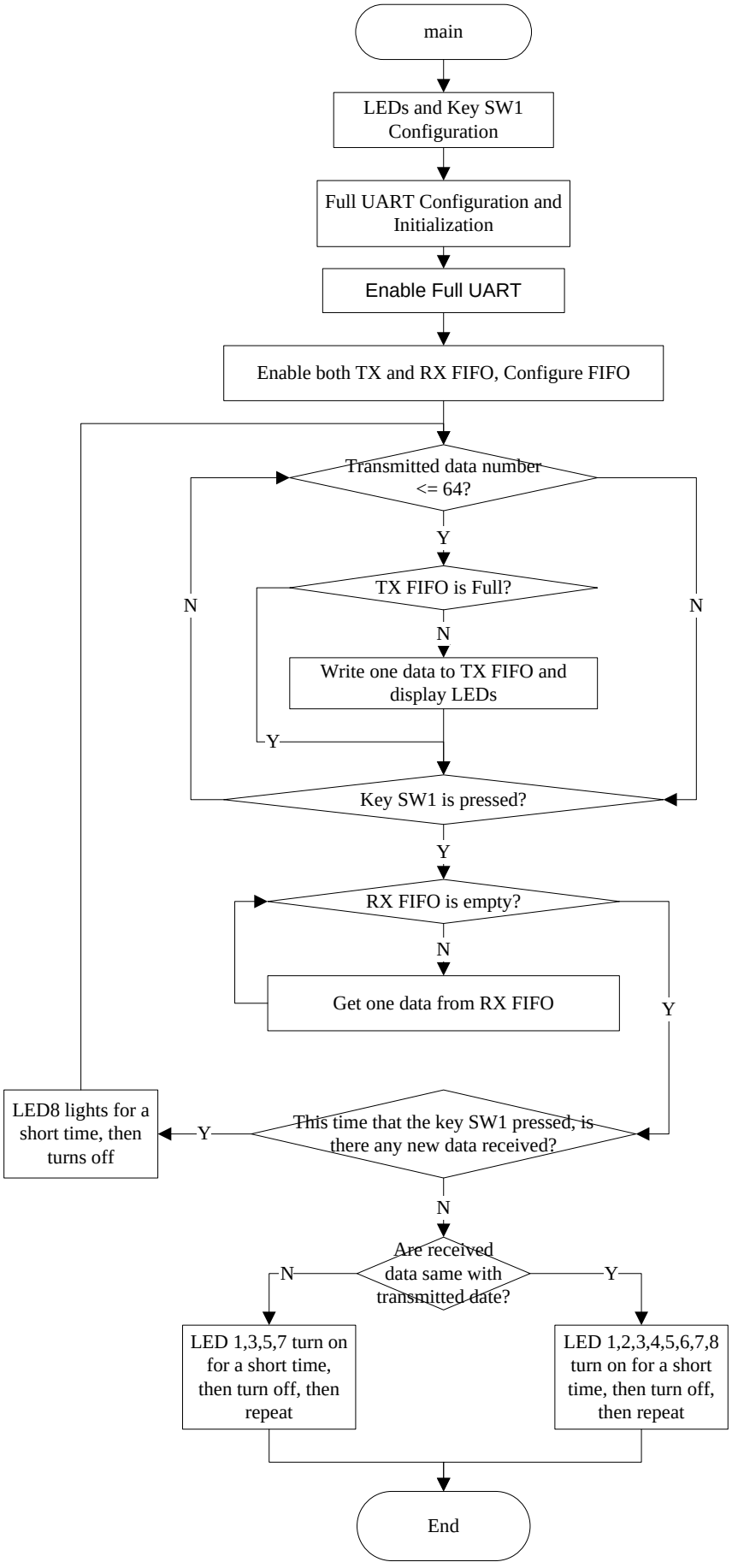
7-6-1 例:ループバック

TX03 ペリフェラルドライバ(FUART, GPIO)を使用し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:RTS と CTS ハードウェアフロー制御をイネーブルにします。
2 回目:RTS と CTS ハードウェアフロー制御ををディセーブルにします。

この例では以下を行います。
1. LED、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、SW1 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
LEDs and Key SW1 Configuration
Full UART Configuration and Initialization
Enable Full UART
Enable both TX and RX FIFO, Configure FIFO
Transmitted data number <= 64?
Y
TX FIFO is Full?
N
N
N
Write one data to TX FIFO and display LEDs
Y
Key SW1 is pressed?
Y
RX FIFO is empty?
N
Get one data from RX FIFO
Y
LED8 lights for a short time, then turns off
Y
This time that the key SW1 pressed, is there any new data received?
N
Are received data same with transmitted date?
N
Y
LED 1,3,5,7 turn on for a short time, then turn off, then repeat
LED 1,2,3,4,5,6,7,8 turn on for a short time, then turn off, then repeat
End

- **サンプルプログラムのコードと説明**

プログラム実行前に、RTS あるいは CTS フロー制御のどちらを使用するかを決定してください。RUN_NONE_FLOW_CONTROL が不定の場合、RTS あるいは CTS フロー制御がプログラム内でイネーブルになります。RUN_NONE_FLOW_CONTROL が定義された場合、プログラム内でイネーブルになるフロー制御はありません。

```
/* #define RUN_NONE_FLOW_CONTROL */
```

まず、プログラムは LED と SW1 を初期化します。
LED と SW1 用に GPIO を設定します。

```
    LEDInit();
    LedOff(LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8);
    SW1_Init();
```

FUART0 用 GPIO を設定します。

```
    /* Configure port PG0 to be TX02 */
    GPIO_SetOutput(GPIO_PG, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_0);

    /* Configure port PG1 to be RX02 */
    GPIO_SetInput(GPIO_PG, GPIO_BIT_1);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_1);

    /* Configure port PG2 to be CTS2 */
    GPIO_SetInput(GPIO_PG, GPIO_BIT_2);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_2);
    GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_2, DISABLE);

    /* Configure port PG4 to be RTS02 */
    GPIO_SetOutput(GPIO_PG, GPIO_BIT_4);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_4);
    GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_4, DISABLE);
```

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;

#ifdef RUN_NONE_FLOW_CONTROL
    myFUART.FlowCtrl = FUART_NONE_FLOW_CTRL;
#else
    myFUART.FlowCtrl = FUART_CTS_FLOW_CTRL | FUART_RTS_FLOW_CTRL;
#endif

    FUART_Init(FUART0, &myFUART);
```

FUART 周辺ドライバーを使用し、FUART0 をイネーブルにし FIFO を設定します。

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

その後 FUART0 は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++;
        }
    }
```

SW1 が On するごとに、プログラムは受信 FIFO からのデータを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。このとき LED8 は最終受信データを表示します。SW1 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、LED 1,2,3,4,5,6,7,8 を点滅します。受信データが送信データと異なる場合、LED 1,3,5,7 を点滅します。

```
if (SW1_DOWN == SW1_Get()) { /* press the key SW1 */
    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_RXDataDisplay(LED8_ON);
        delay(0x180000U);
        LED_RXDataDisplay(LED8_OFF);
    } else {                                        /* receiving data is finished */
        result = Buffercompare(Tx_Buf, Rx_Buf, MAX_BUFSIZE);
        if (result == SAME) {
    /* received data are same with trnsmitted data */
    /* RTS and CTS flow control has worked normally */
            while (1) {
                LedOn(LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8);
                delay(0xFFFFFU);
                LedOff(LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8);
                delay(0xFFFFFU);
            }
        } else {
    /* received data are different with trnsmitted data */
    /* RTS and CTS flow control doesn't work */
            while (1) {
                LedOn(LED1 | LED3 | LED5 | LED7);
                delay(0xFFFFFU);
                LedOff(LED1 | LED3 | LED5 | LED7);
                delay(0xFFFFFU);
            }
        }
    }
}
```

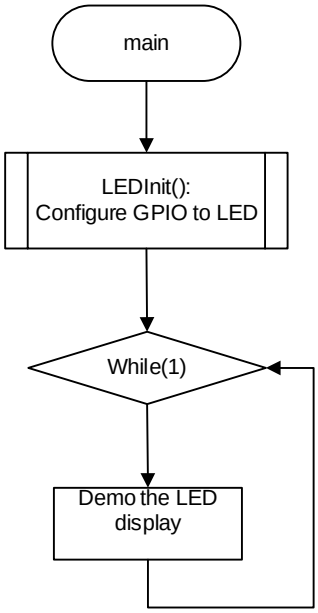
7-7 GPIO

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。

```
GPIO_SetOutput (LED_DATA_PORT, 0XFFU);
GPIO_WriteData (LED_DATA_PORT, 0XFFU);
```

サンプルでは、while ループにて LED を On/Off します。

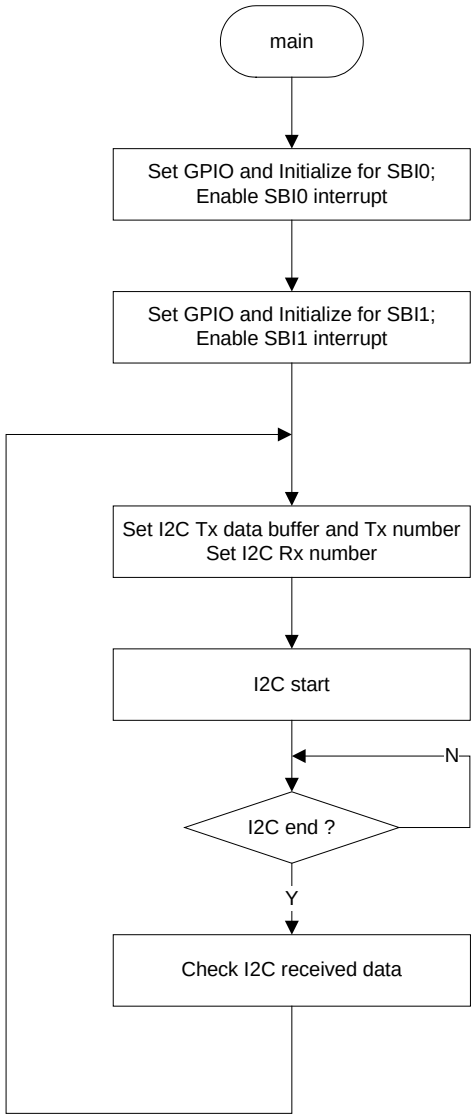
7-8 SBI

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:
        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;
        for (gICnt = 0U; gICnt < 8U; gICnt++) {
            gI2CRxData[gICnt] = 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 (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 */
    }
}
```

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 *SBIy;
    SBI_I2CState sbi1_sr;

    SBIy = TSB_SBI1;
    sbi1_sr = SBI_GetI2CState(SBIy);

    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(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 */
    }
}
```

7-9 SIO/UART

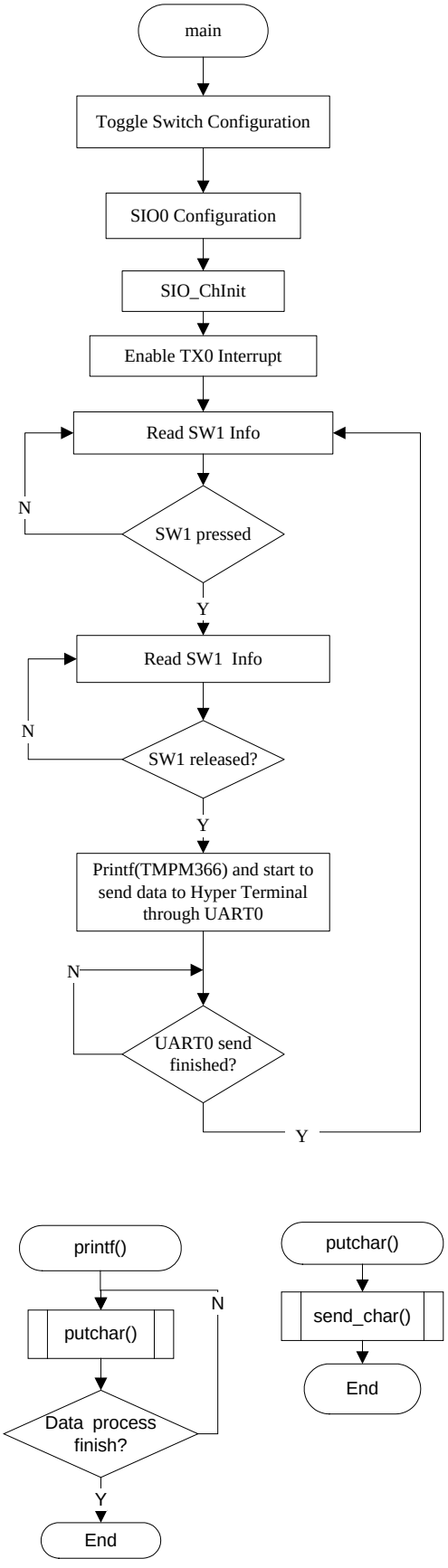
7-9-1 例: リターゲット

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 UART0 の TX 割り込みを使用
4. UART0 に printf()関数をリターゲット

- **フローチャート:**


main
Toggle Switch Configuration
SIO0 Configuration
SIO_ChInit
Enable TX0 Interrupt
Read SW1 Info
N
SW1 pressed
Y
Read SW1 Info
N
SW1 released?
Y
Printf(TMPM366) and start to send data to Hyper Terminal through UART0
N
UART0 send finished?
Y
printf()
N
putchar()
Data process finish?
Y
End
putchar()
send_char()
End

- **サンプルプログラムのコードと説明**

GPIO をスイッチに設定します。

```
GPIO_SetPullUp(TSW_PORT, GPIO_BIT_0, ENABLE);
GPIO_SetPullUp(TSW_PORT, GPIO_BIT_1, ENABLE);
GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_0, ENABLE);
GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_1, ENABLE);
```

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、データを設定します。以下は設定例です。

```
UART_InitTypeDef myUART;

myUART.BaudRate = 115200;     /* 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;
```

その後、UART を初期化します。

```
UART_Enable(UART0);
UART_Init(UART0, &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(INTTX0_IRQn);
```

スイッチ状態を取得します。

```
TSW_info = GPIO_ReadData(TSW_PORT) & TSWBITS;
```

SW1 が押されているかどうかを確認し、SW1 が押されている場合は SW1 が話されるまで待ちます。

```
if (TSW_info == TSW1) {
do {
    wait_TSW1 = 0U;
    TSW_info = GPIO_ReadData(TSW_PORT) & TSWBITS;
    } while (TSW_info != TSWRELEASE);                /* wait for TSW1 released */
}
```

UART 経由で printf()にてデータを送信します。

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

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 へ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(UART0, gSIOTxBuffer[gSIORdIndex++]);
            NVIC_EnableIRQ(INTTX0_IRQn);
      }

      return ch;
}
```

最後に UART0 送信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)
{
      if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
            UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);/* send data */
            fSIO_INT = SET;               /* SIO0 INT is enabled */
      } else {
            /* disable SIO0 INT */
            fSIO_INT = CLEAR;
            NVIC_DisableIRQ(INTTX0_IRQn);
            fSIOTxOK = YES;
      }
      if (gSIORdIndex >= gSIOWrIndex) {     /* reset buffer index */
            gSIOWrIndex = CLEAR;
            gSIORdIndex = CLEAR;
      } else {
            /* do nothing */
      }
}
```

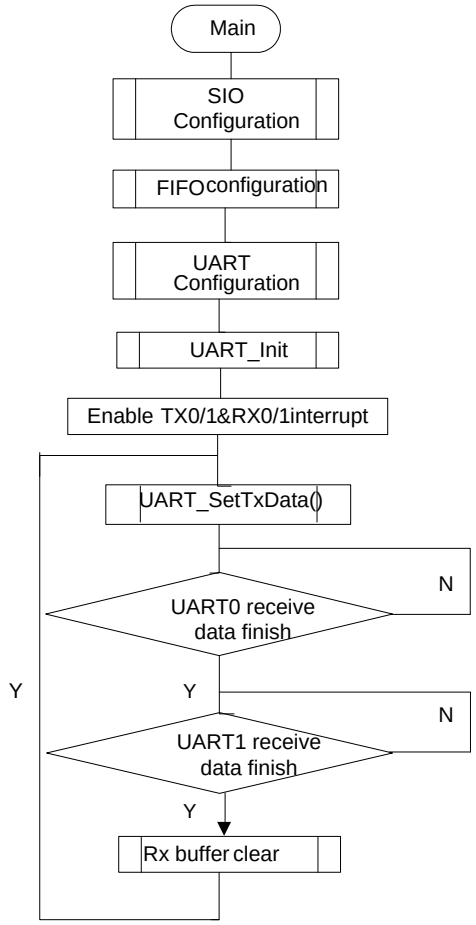
7-9-2 例: UART FIFO

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);
	}
}
```

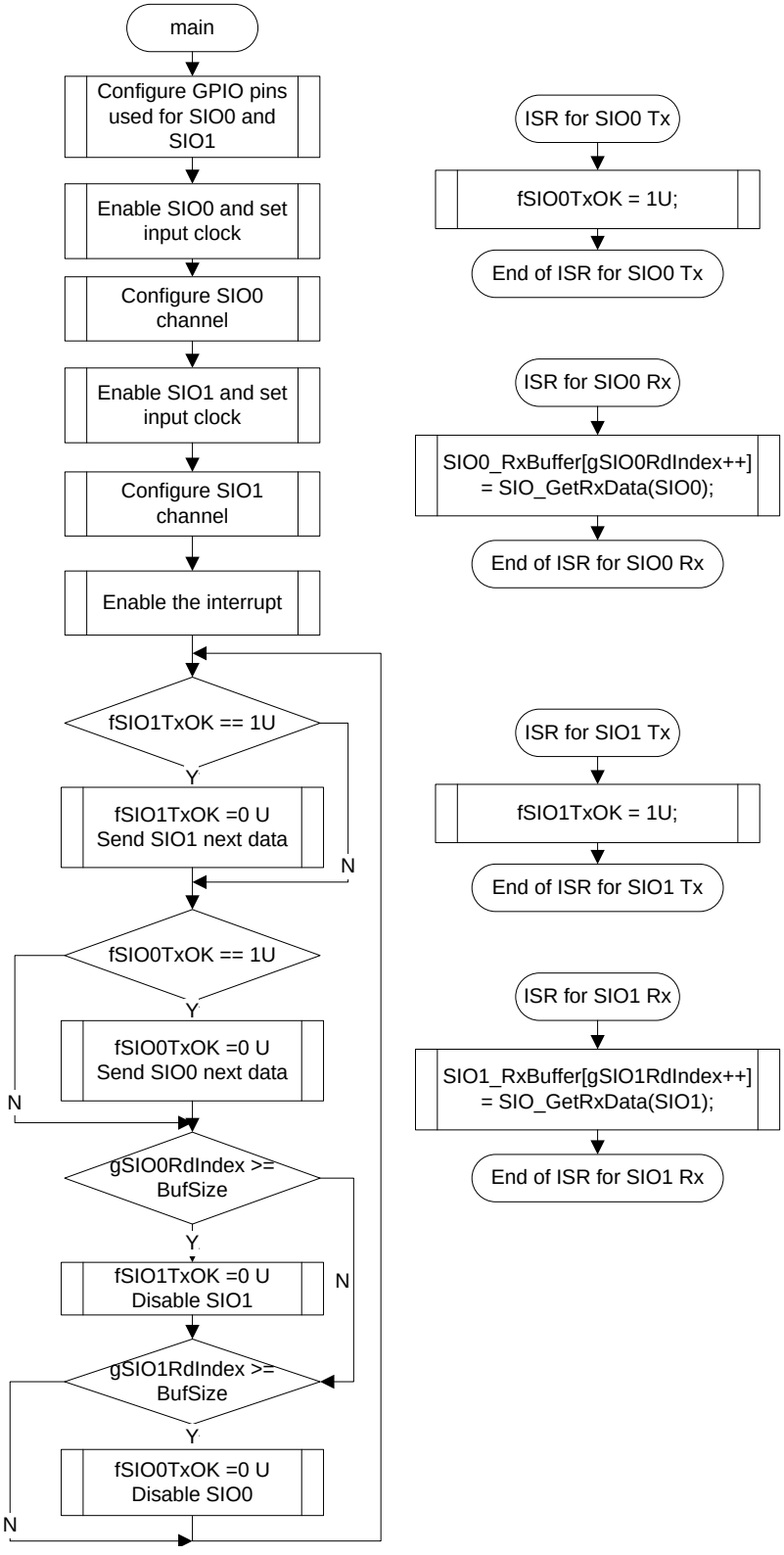
7-9-3 例: SIO

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、入力クロックの設定と 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_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 を有効にし、入力クロックの設定と 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_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);
}
```

7-10 SSP

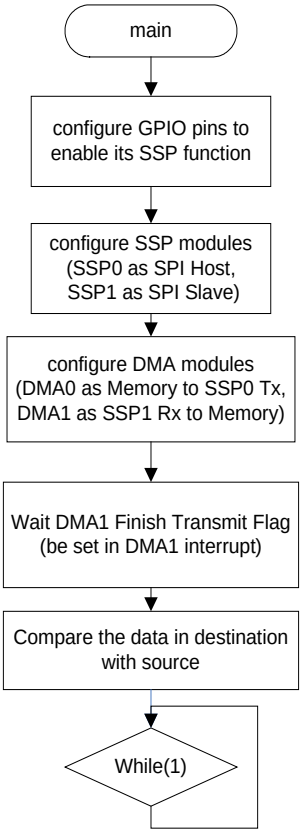
7-10-1 例: DMAC を使用した SSP0->SSP1 転送

ペリフェラルドライバ(SSP, DMAC, 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. SSP0 (マスタ)と SSP1(スレーブ)の初期化
2. DMA0 (メモリ->SSP0 Tx)と DMA1(SSP1 Rx->メモリ)の初期化
3. データ転送 (メモリ → SSP0 Tx → SSP1 Rx → メモリ)

- **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 SSP に設定します。

```
GPIO_SetSSP();
```

SSP を初期化します。

```
initSSP();
```

DMA を初期化し、DMA 転送を開始します。

```
InitDMA();
```

DMA 転送終了を待ちます。

```
/* Wait the end of transmission */
while (TxEndFlag != DONE) {
      /* Do nothing */
}
```

DMA 転送終了後、送受信データを比較します。

```
/* now DMA is finished, Set a Break Point here, */
/* after function Buffercompare() is called, result == SAME */
result = Buffercompare( SRC_Buffer, DST_Buffer, BUFFER_SIZE);
```

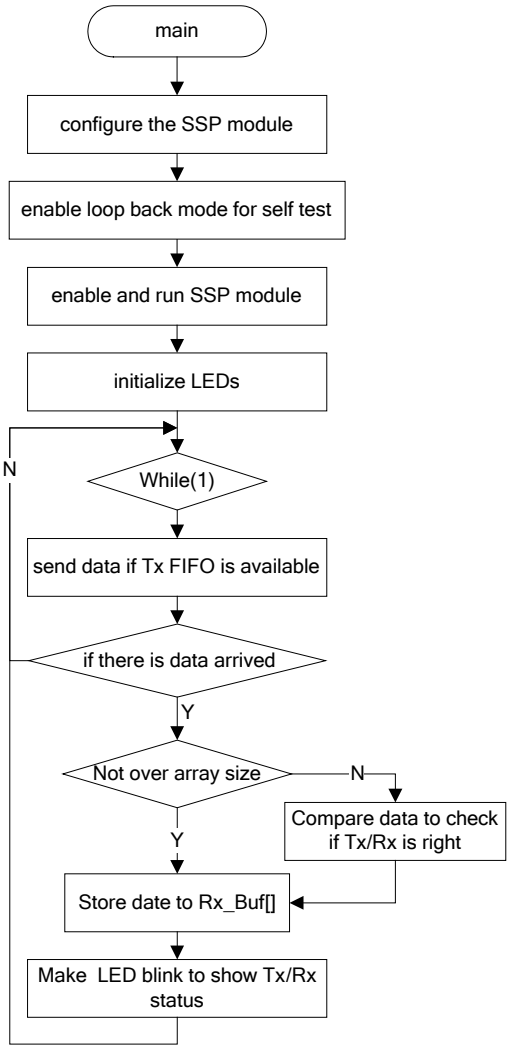
7-10-2 例: SSP セルフループバック

ペリフェラルドライバ (SSP, 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. SSP の初期化
2. ループバック設定

- **フローチャート:**



- **サンプルプログラムのコードと説明**

```
int main(void)
{
    SSP_InitTypeDef initSSP;
    SSP_FIFOState fifoState;

    uint16_t datTx = 0U;            /* must use 16bit type */
    uint32_t cntTx = 0U;
    uint32_t cntRx = 0U;

    uint16_t receive = 0U;
    uint16_t Rx_Buf[MAX_BUFSIZE] = { 0U };
    uint16_t Tx_Buf[MAX_BUFSIZE] = { 0U };

    /* configure the SSP module */
```

```
	initSSP.FrameFormat = SSP_FORMAT_SPI;

	/* 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);

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

	/* enable and run SSP module */
	SSP_Enable(TSB_SSP0);

	/* initialize LEDs on M366-SK board before display something */
	LEDInit();

	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 transmited from Tx_Buf is received in Rx_Buf. */
				/* When the line "#define BITRATE_MIN" is commented, the SSP is
run in maxium */
				/* 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);
                    }

                } else {
                    /* do nothing */
                }

                DisplayLED(receive);
            }
}
```

7-11 TMRB

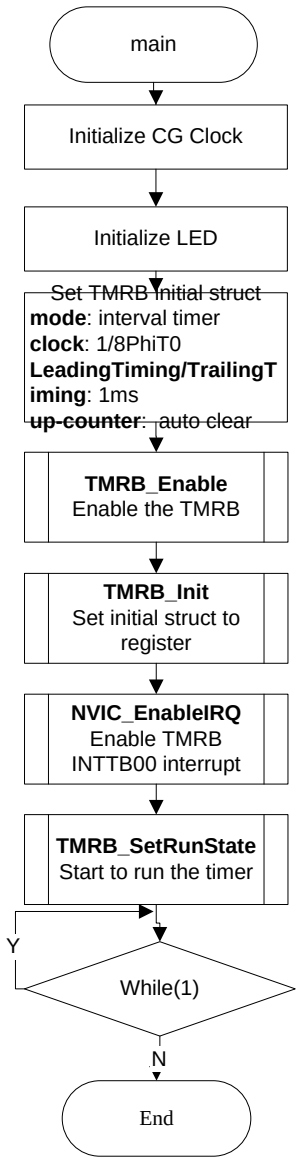
7-11-1 例: 汎用タイマ

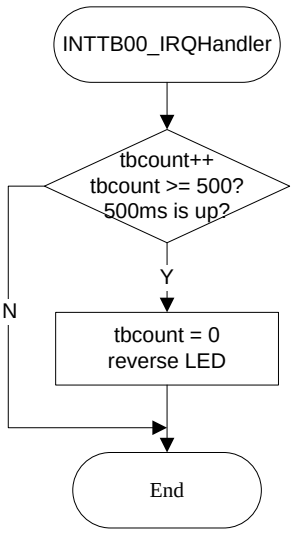
ペリフェラルドライバ(TMRB, GPIO)を使用したサンプルプログラムです。

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

- **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 On します。

```
CG_InitSystem();                    /* CG_SetSystem */
LEDInit();                          /* LED initialize */
LedDisable(LED1 | LED2 | LED3);
LedOn(LED4);                                    /* Turn on LED1 */
```

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_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) {
        LedOff(LED1);
    } else {
        LedOn(LED1);
    }
} else {
    /* do nothing */
}
```

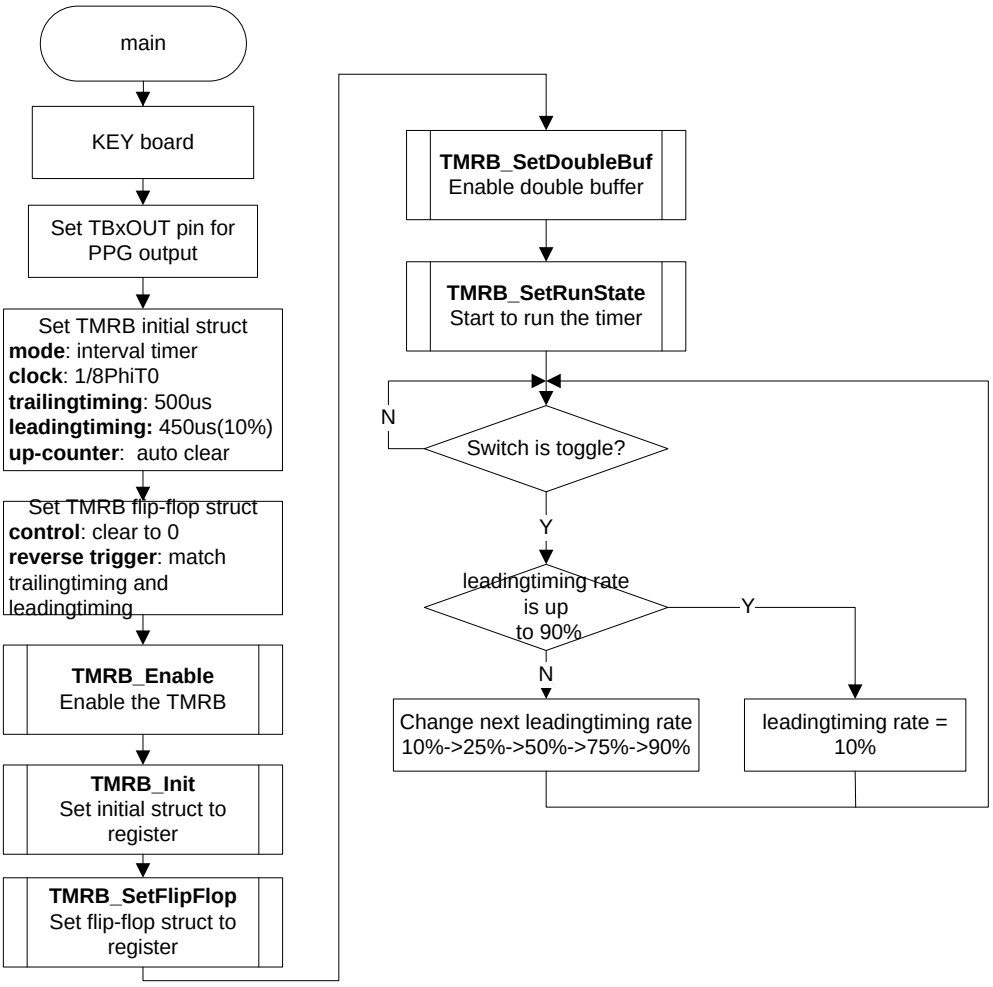
7-11-2 例: PPG 出力

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;                /* trailingtiming is 500us
*/
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
    m_tmrb.LeadingTiming = LeadingTiming[Rate];                              /*
leadingtiming, initial value 10% */
```

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

```
    PPGFFInital.FlipflopCtrl = TMRB_FLIPFLOP_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(0xFFFU);                /* 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(0xFFFU);                /* 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*25% = 150us, Low 幅は 500-125 = 375us, カウンタ値= 375us/Ttmrb = 0x8CAU

LeadingTiming = 50%: High 幅は 500*50% = 250us, Low 幅は 500-250 = 500us, カウンタ値= 250us/Ttmrb = 0x5DCU

Duty = 75%: High 幅は 500*75% = 375us, Low 幅は 500-375 = 125us, カウンタ値 = 125us/Ttmrb = 0x2EEU

Duty = 90%: High 幅は 500*90% = 450us, Low 幅は 500-450 = 50us, カウンタ値 = 50us/Ttmrb = 0x1FAU

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, 0x12CU };
/* leading timing: 10%, 25%, 50%, 75%, 90% */
```

7-12 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/fsys に設定されオーバーフロー時に NMI 割り込みを発生します。

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

WDT を初期化し、その後 WDT を有効にします。

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

DEMO1 では、NMI 割り込みの発生を待ちます。

```
while(1)
{
    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();
```