TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM370)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M370UE-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。

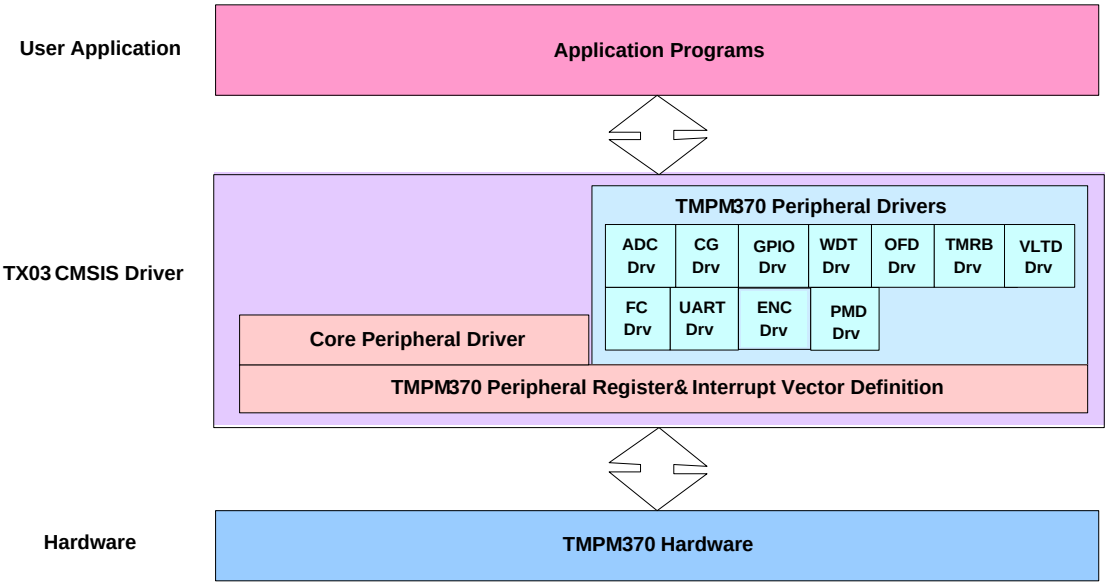
目次

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TMPM370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、必要な機能を動作させることができます。

2 概要

TX03 ペリフェラルドライバを下記のように使用します。



3 使用する機能

機能	チャネル	使用／未使用
クロック／モード制御（CG）	クロックギア	使用
	PLL	PLL8 逓倍
スタンバイモード		使用(STOP モードのみ)
SysTick	-	未使用
ウォッチドッグタイマ（WDT）		使用
外部割り込み (INT)	INT0	未使用
	INT1	未使用
	INT2	未使用

機能	チャネル	使用／未使用
	INT3	使用(CG サンプルにおける STOP モードからの復帰)
	INT4	未使用
	INT5	未使用
	INT6	未使用
	INT7	未使用
	INT8	未使用
	INT9	未使用
	INT10	未使用
SIO シリアルチャネル（SIO）	SIO0	使用(UART 通信制御)
	SIO1	使用(UART FIFO, SIO 通信制御)
	SIO2	未使用
	SIO3	未使用
16 ビットタイマ(TMRB)	TMRB0	使用(汎用タイマ)
	TMRB1	未使用
	TMRB2	未使用
	TMRB3	未使用
	TMRB4	未使用
	TMRB5	未使用
	TMRB6	未使用
	TMRB7	使用(PPG 出力)
周波数検知回路(OFD)	-	使用
パワーオンリセット回路(POR)	-	未使用
電圧検出回路(VLTD)	-	使用
エンコーダ入力回路(ENC)	ENC0	使用: ホイールサンプル
	ENC1	未使用
モータ制御回路(PMD)	PMD0	未使用
	PMD1	使用: PMD 位相出力
12ビットA/Dコンバータ(12-bitA/D converter)	AINA0	使用(ADC データリード)
	AINA1	未使用
	AINA2	未使用
	AINA3	未使用
	AINA4	未使用
	AINA5	未使用
	AINA6	未使用
	AINA7	未使用
	AINA8	未使用
	AINA9	未使用
	AINA10	未使用
	AINA11	未使用

機能	チャネル	使用／未使用
	AINA12	未使用
	AINA13	未使用
	AINA14	未使用
	AINB0	未使用
	AINB1	未使用
	AINB2	未使用
	AINB3	未使用
	AINB4	未使用
	AINB5	未使用
	AINB6	未使用
	AINB7	未使用
	AINB8	未使用
	AINB9	未使用
	AINB10	未使用
	AINB11	未使用
	AINB12	未使用
	AINB13	未使用
	AINB14	未使用
	AINB15	未使用
	AINB16	未使用

4 端子用途

本サンプルプログラムは、開発環境にIAR社TMPM370-SK評価ボードを用いてテストされています(東芝製TMPM370評価ボードを用いてテストされたVLTDを除きます)。以下に、端子用途を説明します。

No	Name	Usage
1	CVREFD	VR4を接続
2	CVREFABC	VR3と接続
3	DVSS	GND
4	INT3/TB0IN/PA0	INT3/SW1
5	TB0OUT/PA1	SW2
6	INT4/TB1IN/PA2	未使用
7	TB1OUT/PA3	未使用
8	CTS1/SCLK1/PA4	未使用
9	TB6OUT/TXD1/PA5	未使用
10	TB6IN/RXD1/PA6	未使用
11	INT8/TB4IN/PA7	未使用
12	TXD0/PE0	TXD0 (UART0データ送信)
13	RXD0/PE1	RXD0 (UART0データ受信)
14	CTS0/SCLK0/PE2	未使用
15	TB4OUT/PE3	未使用
16	DVDD5	未使用
17	INT5/TB2IN/PE4	未使用
18	TB2OUT/PE5	未使用
19	INT6/TB3IN/PE6	未使用
20	INT7/TB3OUT/PE7	未使用
21	DVSS	GND
22	INTB/PL0	未使用
23	INTA/PL1	未使用
24	UO0/PC0	LED1
25	XO0/PC1	未使用
26	VO0/PC2	LED2
27	YO0/PC3	未使用
28	WO0/PC4	LED3
29	ZO0/PC5	未使用
30	EMG0/PC6	未使用
31	OVV0/PC7	未使用
32	TB5IN/ENCA0/PD0	ENCA0
33	TB5OUT/ENCB0/PD1	ENCB0

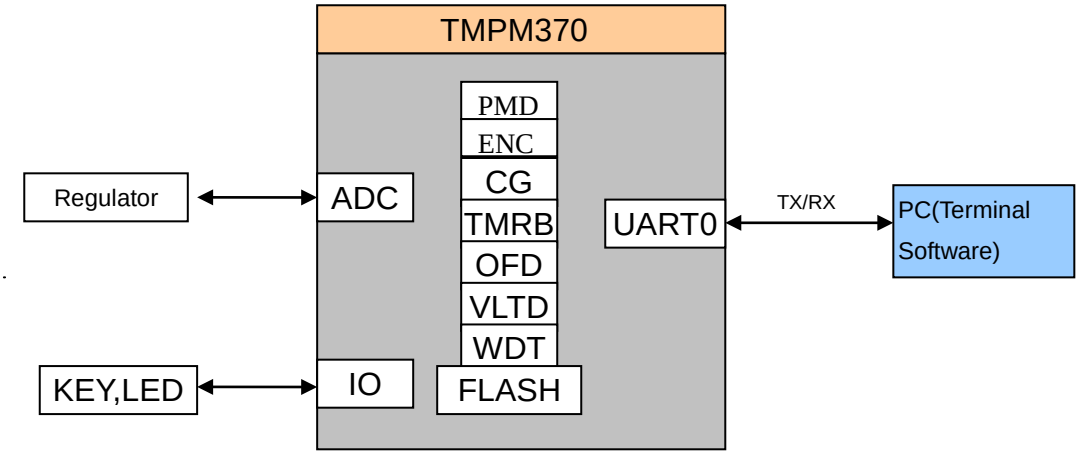
No	Name	Usage
34	ENCZ0/PD2	未使用
35	INT9/PD3	未使用
36	CTS2/SCLK2/PD4	未使用
37	TXD2/PD5	未使用
38	RXD2/PD6	未使用
39	PG0/UO1	PMD UO1 出力
40	PG1/XO1	PMD XO1 出力
41	PG2/VO1	PMD VO1 出力
42	DVDD5	+5V
43	DVSS	GND
44	PG3/YO1	PMD YO1 出力
45	PG4/WO1	PMD WO1 出力
46	PG5/ZO1	PMD ZO1 出力
47	PG6/EMG1	PMD EMG1 入力
48	OVV1/PG7	未使用
49	X1	10MHz 発振子接続
50	DVSS	GND
51	PM1/X2	10MHz 発振子接続
52	PF0/TB7IN/BOOT	未使用
53	PF1/TB7OUT	PPG出力
54	PF2/ENCA1/SCLK3	未使用
55	PF3/ENCB1/TXD3	未使用
56	PF4/ENCZ1/RXD3	未使用
57	VOUT15	GND
58	DVSS	GND
59	MODE	GND
60	RVDD5	+5V, VLTD検知端子 (*)
61	RESET	RESET
62	VOUT3	GND
63	DVDD5E	+5V
64	DVSS	GND
65	PB0/TRACECLK	未使用
66	PB1/TRACEDATA 0	未使用
67	PB2/TRACEDATA 1	未使用
68	PB3/TMS/SWDIO	未使用
69	PB4/TCK/SWCLK	未使用
70	PB5/TDO/SWV	未使用
71	PB6/TDI	未使用
72	PB7/TRST	未使用
73	PK1/AINB12/INTF	未使用
74	PK0/AINB11/INTE	未使用

No	Name	Usage
75	PJ7/AINB10/INTD	未使用
76	PJ6/AINB9/INTC	未使用
77	PJ5/AINB8	未使用
78	PJ4/AINB7	未使用
79	PJ3/AINB6	未使用
80	PJ2/AINB5	未使用
81	PJ1/AINB4	未使用
82	PJ0/AINB3	未使用
83	AVSSB/VREFLB	AGND
84	AVDD5B/VREFHB	A+5V
85	PI3/AINA11/AINB2	未使用
86	PI2/AINA10/AINB1	未使用
87	PI1/AINA9/AINB0	未使用
88	AVSSA/VREFLA	AGND
89	AVDD5A/VREFHA	A+5V
90	PI0/AINA8	未使用
91	PH7/AINA7	未使用
92	PH6/AINA6	未使用
93	PH5/AINA5	未使用
94	PH4/AINA4	未使用
95	PH3/AINA3	未使用
96	PH2/AINA2/INT2	未使用
97	PH1/AINA1/INT1	未使用
98	PH0/AINA0/INT0	アナログ入力、VR1と接続
99	AMPVSS	AGND
100	AMPVDD5	A+5V

＊ 東芝製M370評価ボード(非売品)を用いたVLTD用プログラムで使用する端子です。

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:
   1) IAR 社製 TMPM370-SK 評価ボード
   2) 東芝製 TMPM370 評価ボード(非売品)

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 5.5 version
   - KEIL 社:
     1) U-Link: Realview ULINK2
     2) IDE: KEIL uVision 4.14

補足: ENC/PMD サンプル動作環境は次の通りです。

- IAR:
  1) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
  2) IDE: IAR Embedded workbench 6.50.1 version
- KEIL:
  1) IDE: KEIL uVision 4.60

6 機能

6-1 動作モード選択

TMPM370 には 3 つの動作モードがあります: NORMAL, IDLE, STOP モード

- **NORMAL モード:**

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

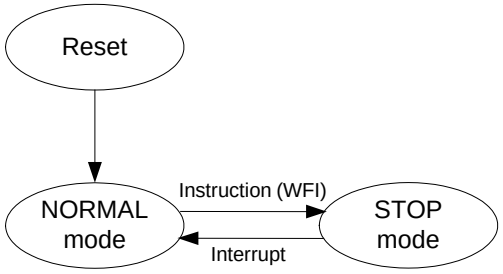
IDLE モードと STOP モードは低消費電力モードです。
低消費電力モードへ移行するには、システ制御レジスタ CGSTBYCR<STBY2:0>にて IDLE モードまたは STOP モードを選択し、WFI (Wait For Interrupt)命令を実行します。

注: STOP モードのみサンプルプログラムに含まれます。

- **STOP モード:**

STOPモードでは、すべての内蔵回路が停止します。
STOPモードが解除されると、STOPモードへ移行する直前の動作モードへ復帰し、動作を開始します。
STOPモードでは、CGSTBYCR<DRVE>の設定により端子の状態を設定することができます。
外部割込みのみ(INT0～INTF) STOPモードを解除できます。

注: STOP モードのサンプルプログラムは CG の例に含まれています。



6-2 ADC

PH0(AINA0)に接続された VR1 の値を変更します。この電圧は測定され、stdout へプリントします。stdout は UART にリターゲットされているので、UART0 と PC を接続しておくことで、AD 変換結果をターミナルソフトで確認することができます。

6-3 CG

6-3-1 パワーモード変更

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

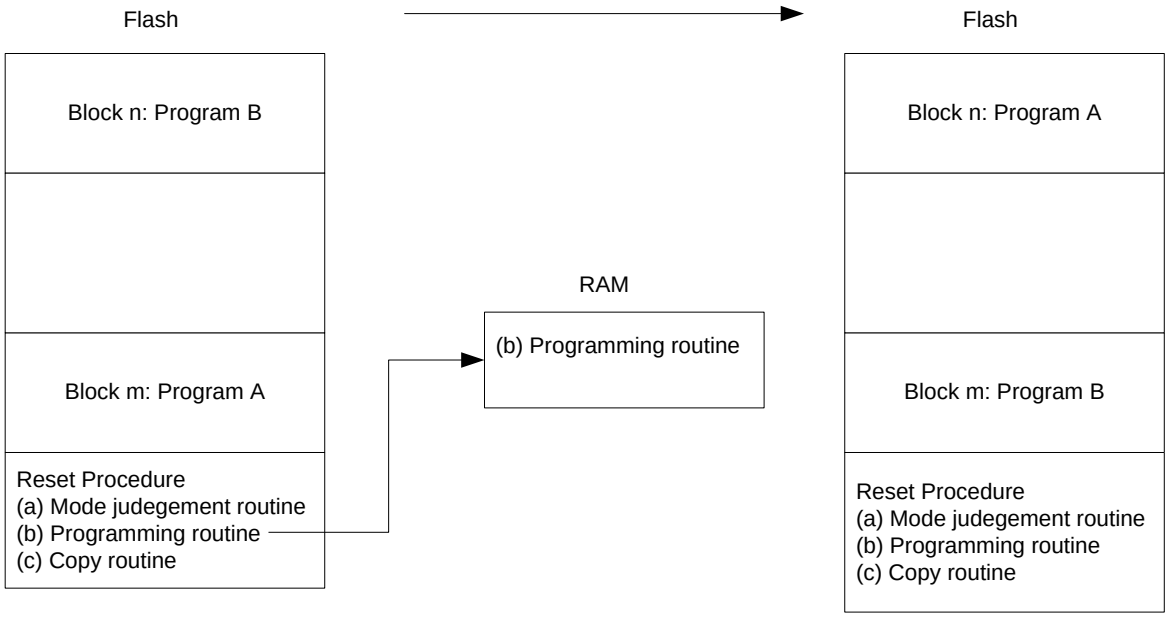
現在のモード	アクション (Key)	動作	LED 表示
NORMAL	SW2 を押す	NORMAL→ STOP	LED オフ
STOP	SW1 を押す	STOP →NORMAL	LED オン

6-4 FLASH

このアプリケーションは、内蔵フラッシュメモリの消去及び再書き込み操作を行います。
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ー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 点滅、B: LED2 点滅)
初期状態は、プログラム A が最初に動作します。
ーSW1 キーはリセット動作のモード判定に使います。

SW1 キーが押された場合 → ユーザブートモードです。

SW1 キーが押されていない場合 → Normal モードです。

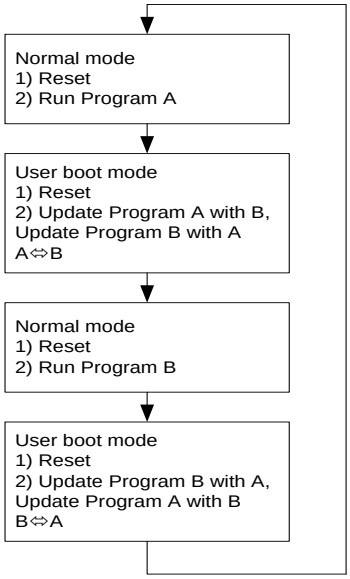


動作手順

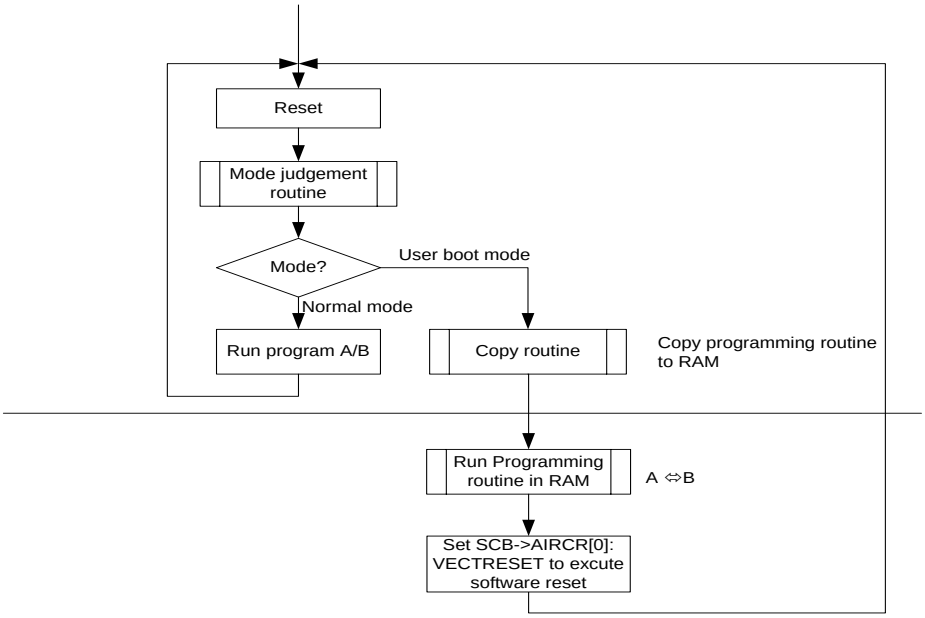
(1) 電源投入
評価ボードのリセット動作が行われます。
初期プログラム A がフラッシュ ROM へ格納され動作します。
LED1 が点滅します。

(2) SW1 キーを押している間にリセットを押してください。その後、LED3 が点灯したら SW1 キーを離してください
評価ボードのリセット動作が動きます: 書き込みルーチンが転送ルーチンで RAM へ転送されます。
書き込みルーチンが動作し、フラッシュ ROM 上のプログラム A と B が入れ替わります。

(3) しばらくすると、入れ替わりが終了し、LED3 が消灯します。
評価ボードは自動的にソフトウェアによってリセットします。
その後、プログラム B がフラッシュ ROM へ格納され動作します。
LED2 が点滅します。



・フローチャート



6-5 GPIO

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

6-6 OFD

この例では、OFD 機能の初期化を行います。

6-7 SIO/UART

6-7-1 リターゲット

C 言語の標準入出力ライブラリの標準入力(stdin)、標準出力(stdout)をターゲットのハードウェアに合わせて変更することをリターゲットと呼びます。
サンプルプログラムでは、標準入力と標準出力を、共にUART0に設定します。アプリケーションから printf()関数を使って UART0 から出力を行います。

6-7-2 UART FIFO

この例では、UART0 から"TMPM3701"というデータを FIFO を使用した UART1 へ送信します。また同時に UART0 は"TMPM3702"というデータを FIFO を使用した UART1 から受信します。
ResetIdx()関数の前にブレークポイントを設定してください。RxBuffer=”TMPM3702”となり、RxBuffer1="TMPM3701"となると設定したブレークポイントで停止します。

6-7-3 SIO

この例では、TMPM370 の SIO モジュールを使用して同期式の送受信を行います。
SIO のチャネル 0 とチャネル 1 を使用し、これらのチャネル間で同期式データ送信を行います。
(TXD0 と RXD1、TXD1 と RXD0、sclk0 と sclk1 を接続してください)

6-8 TMRB

6-8-1 汎用タイマー

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

6-8-2 プログラマブル矩形波出力(PPG)

SW1 キーを使い、デューティ可変の PPG(プログラマブル矩形波)の波形を出力します。
デューティは 5 段階(10%, 25%, 50%, 75%, 90%)に変更できます。
電源投入すると PPG モードに入り、PPG 出力を開始します。
キーを押すことでデューティを変更します。
10% → 25% → 50% → 75% → 90% → 10%

6-9 VLTD

この例では、VLTD によって検出される電圧状態を実装しています。
電圧が検出電圧より低い場合はハード的に MCU リセットが行われ、ソフト的に PC0 に HIGH レベルを出力します。

6-10 WDT

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

本ドライバサンプルの流れは以下です。

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

   DEMO1:

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

   DEMO2:

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

6-11 ENC

この例では、ENC ドライバを使用してホールマウスの回転を検知します。

動作シーケンス:

1. ホイールマウスエンコーダーの端子 A(図 1 参照)と端子 32 (ENCA0)を接続します。
2. ホイールマウスエンコーダーの端子 B(図 1 参照)と端子 33(ENCB0)を接続します。
3. USB マウスと PC を接続し、ホイールマウスエンコーダーの Com(図 1)を 5V と接続します。
4. プログラムを実行します(LED1 が点滅します)。
5. マウスのホイールを前に回転すると LED1 はより短い間隔で点滅します。点滅間隔が最も短くなると、点滅間隔は初めの間隔に戻ります。
6. マウスのホイールを後ろに回転すると LED1 はより長い間隔で点滅します。点滅間隔が最も長くなると、点滅間隔は初めの間隔に戻ります。

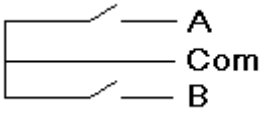


図 1

6-12 PMD

6-12-1 例: 位相出力

この例は、PMD ドライバを使用して位相出力を行います。

動作シーケンス:
1. プロジェクトを開き、DEMO_U_PHASE、または DEMO_3_PHASE と定義されたサンプルを選択します。プログラムを実行します。
2. EMG1 (pin 47)を DVSS (pin 64)とプルダウン接続します。(LED1 が点灯します)これは EMG 保護状態であることを意味します。
3. EMG1 (pin 47)を DVDD5 (pin 63)とプルアップ接続します。(LED1 が消灯します)これは PMD が通常動作であることを意味します。
4. UO1 (pin 39) をオシロスコープに接続し、波形を確認します。
5. XO1 (pin 40) をオシロスコープに接続し、波形を確認します。
6. VO1 (pin 41) をオシロスコープに接続し、波形を確認します。
7. YO1 (pin 44) をオシロスコープに接続し、波形を確認します。
8. WO1 (pin 45) をオシロスコープに接続し、波形を確認します。
9. ZO1 (pin 46) をオシロスコープに接続し、波形を確認します。

DEMO_U_PHASE 定義を選択した場合、位相波形は同じです。
UO1 のデューティは 25%、出力電圧は 5V、周期は 410us です。XO1 は UO1 を反転させた形です。VO1 と WO1 は UO1 と同じで、YO1 と ZO1 は XO1 と同じです。

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

7 ソフトウェア

本ソフトウェアは、、TMPM370 MCU の主要機能を IAR TMPM370-SK 評価ボード上で動作確認するためのサンプルプログラムです。

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

ワークスペース構造とプロジェクト名は、以下の通りです:

IAR EWARM:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm370_wdt_int.c
        │   tmpm370_wdt_int.h
        │
        ├─TX03_CMSIS
        │   │   system_TMPM370.c
        │   │   system_TMPM370.h
        │   │   TMPM370.h
        │   │
        │   └─startup
        │         startup_TMPM370.s
        │
        ├─TX03_Periph_Driver
        │   ├─inc
        │   │   tmpm370_cg.h
        │   │   tmpm370_gpio.h
        │   │   tmpm370_wdt.h
        │   │   tx03_common.h
        │   │
        │   └─src
        │         tmpm370_cg.c
        │         tmpm370_gpio.c
        │         tmpm370_wdt.c
        │
        └─TMPM370-SK
                led.c
                led.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm370_wdt_int.c
        │
        ├─TX03_CMSIS
        │   │   system_TMPM370.c
        │   │   system_TMPM370.h
        │   │   TMPM370.h
        │   │
        │   └─startup
        │         startup_TMPM370.s
```

```
│
├─TX03_Periph_Driver
│      tmpm370_cg.c
│      tmpm370_gpio.c
│      tmpm370_wdt.c
│
└─TMPM370-SK
        led.c
```

7-1 ADC

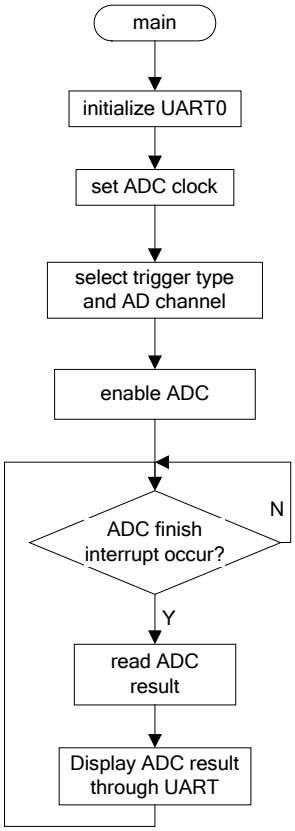
7-1-1 例: ADC データ読み出し

TX03 ペリフェラルドライバ(ADC, GPIO, 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. ADC 設定と初期化
2. ソフトウェアトリガによる AD 変換を開始し、AD 変換結果を読み出します。

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

```
/* set ADC clock */
ADC_SetClk(TSB_ADA, ADC_HOLD_FIX, ADC_FC_DIVIDE_LEVEL_16);

/* select trigger type and AD channel, this time software trigger is used */
/* VR1 is connected to AIN0, remember to input with macro TRG_ENABLE() */
ADC_SetSWTrg(TSB_ADA, ADC_REG0, TRG_ENABLE(ADC_AIN0));

/* enable ADC module */
ADC_Enable(TSB_ADA);
```

AD 変換を開始します:

```
ADC_Start(TSB_ADA, ADC_TRG_SW);
```

AD 変換を行い、その後、ADC モジュールの状態を確認します。AD 変換終了時、ADC_Display()関数をコールし、他の AD 変換を開始します。

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

    If (adcState == DONE) {
        ADC_Display();

        /* Start another AD conversion */
        ADC_Start(TSB_ADA, ADC_TRG_SW);
    }
}
```

ADC_Display()関数では、AD 変換結果である ADREG の値を読み出します。

```
ADC_Result result = ADC_GetConvertResult(TSB_ADA, ADC_REG0);
```

7-2 CG

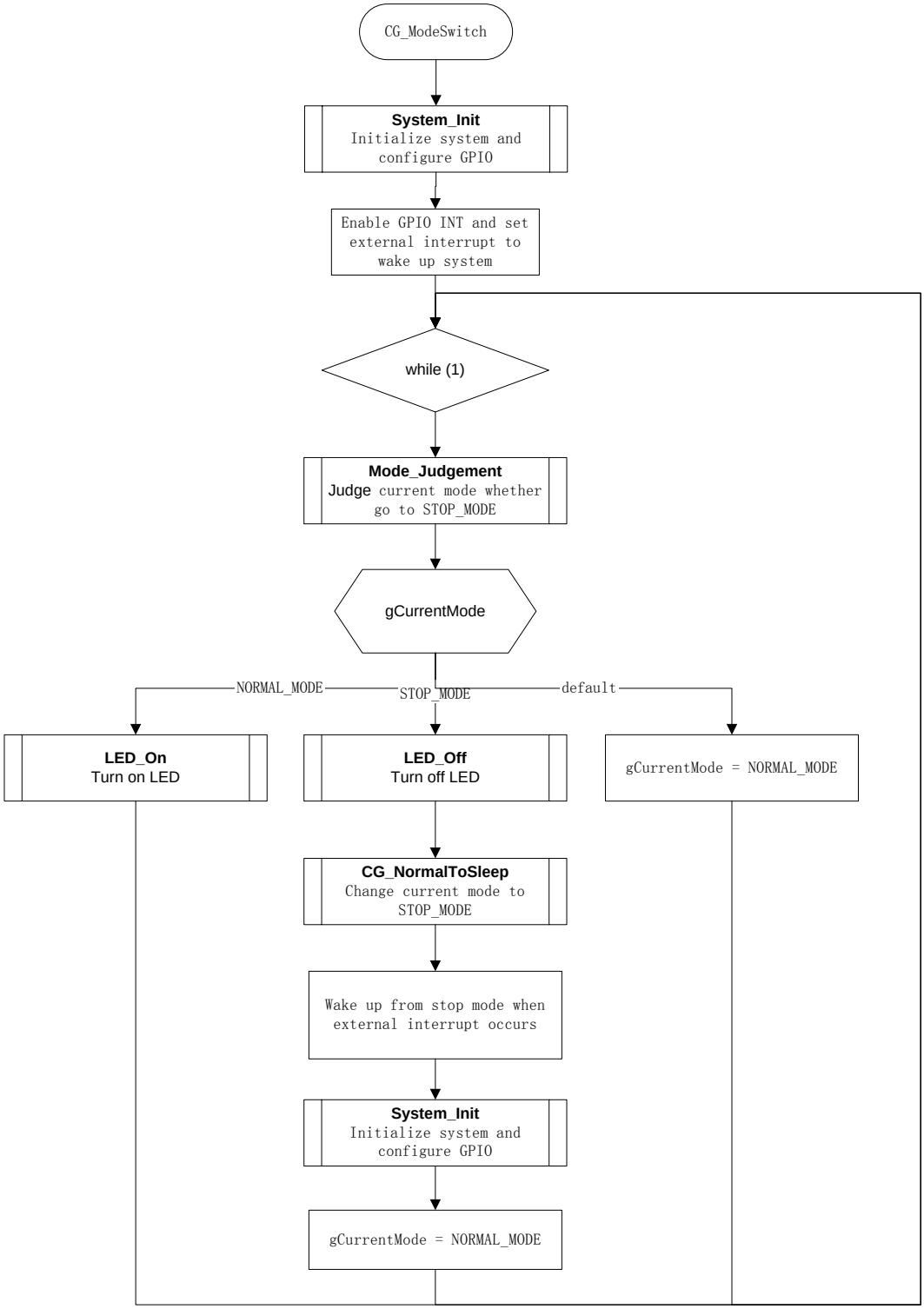
7-2-1 例: パワーモード変更

TX03 ペリフェラル・ドライバ(CG, GPIO)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. 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**

(リセット後)CG の通常設定

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

サンプルプログラムのコード:

```
/* Set fgear = fc/2 */
CG_SetFgearLevel(CG_DIVIDE_2);
/* Set fperiph to fgear */
CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
CG_SetPhiT0Level(CG_DIVIDE_64);
/* Enable high-speed oscillator */
CG_SetFosc(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();
```

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と外部割込み INT3 を設定し、システムを起動します。割り込み保留要求をクリアし、INT3 を許可します。最後に、__WFI() 命令を使用し STOP モードに入ります。

```
/* Set CG module: Normal ->Stop mode */
/* Disable interrupts */
__disable_irq();
/* Enable GPIO INT */
CG_ClearINTReq(CG_INT_SRC_3);
/* Set external interrupt to wake up system */
CG_SetSTBYReleaseINTSrc(CG_INT_SRC_3,
                        CG_INT_ACTIVE_STATE_FALLING, ENABLE);
NVIC_ClearPendingIRQ(INT3_IRQn);
NVIC_EnableIRQ(INT3_IRQn);
__enable_irq();

CG_SetWarmUpTime(CG_WUODR_EXT);
__DSB();
/* Enter stop mode */
__WFI();
```

マルチクロック回路の許可

PLL 、および fc のソース設定。

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

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

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

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

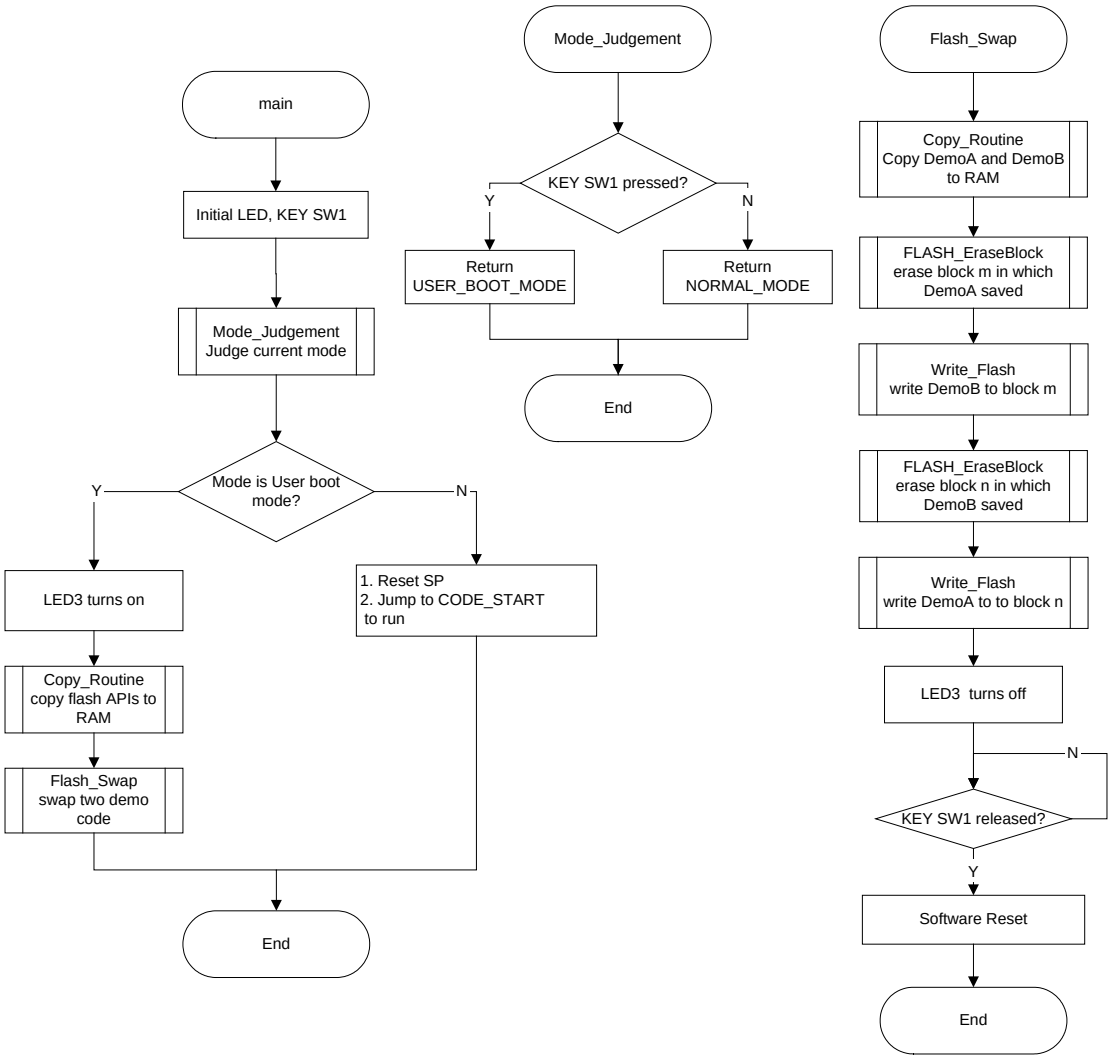
7-3 FLASH

TX03 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。

内容は下記になります。

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**

KEY SW1 と LED を初期化。リセット時、および LED に現在の動作モード情報の表示時、KEY ポート (GPIO) で現在のモードを判断します。

```
Init_Led();
SW1_Init();
```

リセット後にどのモードになってるのか判断するために、Mode_Judgement() 関数を使用します。

```
uint8_t Mode_Judgement(void)
{
    return (SW1_Get() == 1U) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

リセット時に KEY SW1 が押下されない場合、ルーチンプログラムはノーマルモードになります。その後、ルーチンプログラムは SP をリセットし、“CODE_START” にジャンプし動作させます。ブロック m に属するサンプル A は“CODE_START” に保存されているため、サンプル 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 KEY SW1 が押下された場合、ルーチンプログラムはブートモードになります。LED3 は点滅してブートモードであることを表示します。その後、ルーチンプログラムは 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”コピーします。プログラム実行中は、同時に Flash メモリは削除/書き込みを実行できないためです。

```
    Led_On(LED3);                     /* enter user boot mode */
    Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
    /* copy flash API to RAM */
```

Flash 動作用 API を RAM にコピー後、ルーチンプログラムは Flash_Swap()関数にジャンプします。この関数は、上記の Copy_Routine() を使用し、Flash メモリーから 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 メモリーの除/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 のプログラムは Flash メモリ内にて差し替えられます。

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

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

    /* write B to A */
    if (FC_SUCCESS == Write_Flash(DEMO_A_FLASH, DEMO_B_RAM,
SIZE_DEMO_B)) {
```

```
            /* Do nothing */
        } else {
            return ERROR;
        }

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

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

SWAP オペレーションが完了したことを示すため、LED3 が消えます。KEY SW1 が解除されると、SCB->AIRCR レジスタを用いてソフトリセットが行われます。その後、サンプルが再度動作しノーマルモードになります。サンプル A、サンプル B が差し替えられるため、アドレス “CODE_START” はサンプル B のスタートアドレスになります。サンプル B が始動します(LED2 が点滅)。

```
    delay(0x7FFFFF);
    Led_Off(LED3);                          /* complete and restart */
    delay(0x2FFFFF);

    /* wait for Key SW1 to release*/
    while (SW1_Get() == GPIO_BIT_VALUE_1) {
    }

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

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

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

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

次に、削除が終了すると、Flash ﾄﾞﾗｲﾊﾞ FC_GetBusyState() を用いモニターリングを行いま

す。同時に、タイムアウトカウンタを使用し、動作が指定時間を越えていないか確認します。

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

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

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

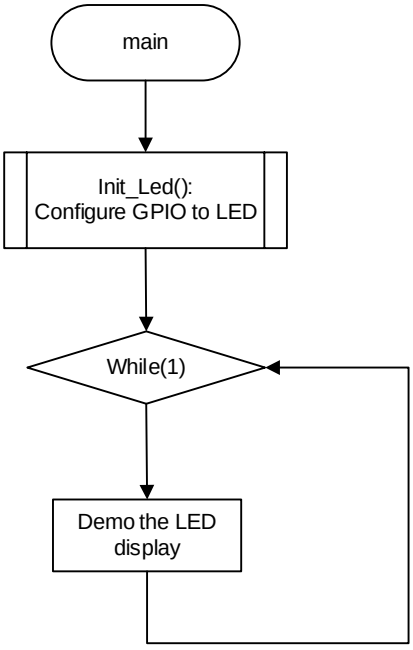
7-4 GPIO

TX03 ペリフェラルドライバ 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. 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 を LED に設定します。例えば、端子を出力端子として設定し、端子データを設定します。

```
void Init_Led()
{
    GPIO_SetOutput(LED_DATA_PORT, LED1 | LED2 | LED3);
    GPIO_WriteDataBit(LED_DATA_PORT, LED1, GPIO_BIT_VALUE_0);
    GPIO_WriteDataBit(LED_DATA_PORT, LED2, GPIO_BIT_VALUE_0);
    GPIO_WriteDataBit(LED_DATA_PORT, LED3, GPIO_BIT_VALUE_0);
}
```

プログラム動作中、LED は1つずつ点滅します。

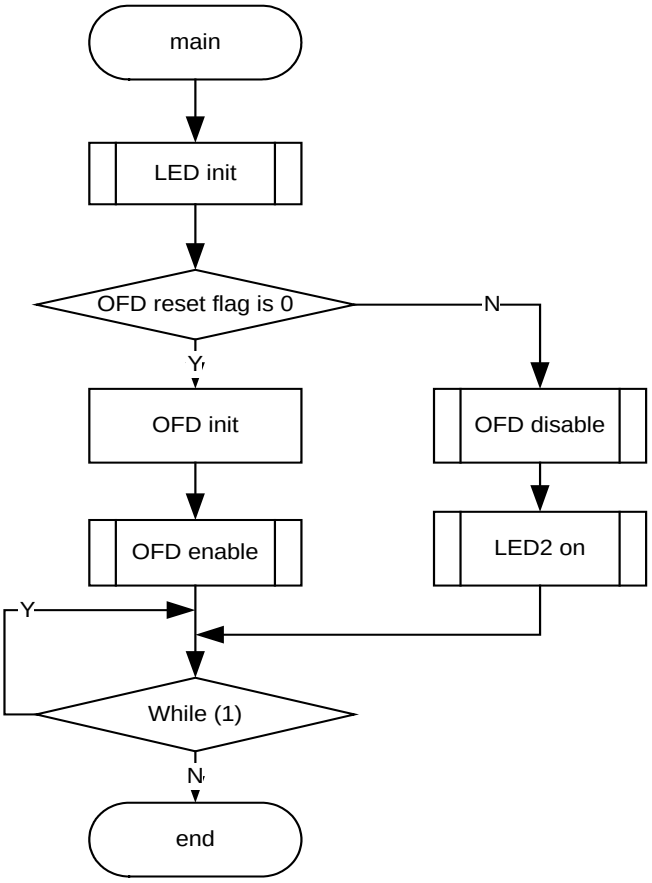
7-5 OFD

TX03 ペリフェラルドライバ(OFD, CG, 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. 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。

```
    Init_Led();
```

OFD のリセット flag を確認します。

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

例えば電源投入時のノーマルリセット処理において、OFD フラグを'0'クリアし、外部発振を許可します。そして OFD 検知周波数範囲を OFD に設定し、OFD を初期化します。
以下は外部発振を許可して PLL を ON にする CG 設定のコードです。

```
void ChangeToEHOSC(void)
{
    /* Use the external oscillation */
  TSB_CG->SYSCR  = SYSCR_Val;
  TSB_CG->OSCCR  = OSCCR_Val;
  TSB_CG->STBYCR = STBYCR_Val;

  if (TSB_CG_OSCCR_PLLON) {    /* If PLL enabled */
    TSB_CG_OSCCR_WUEON = 1U;
    while(TSB_CG_OSCCR_WUEF) {      /* Warm-up          */
        /* Do nothing */
    }
  }
  TSB_CG->PLLSEL = PLLSEL_Val;
  TSB_CG->CKSEL  = CKSEL_Val;
}
```

OFD の初期化のために、まずレジスタへの書き込みを許可します。

```
    OFD_SetRegWriteMode(ENABLE);
```

検知周波数を設定します。

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

DEMO2 を定義すると異常な周波数範囲が設定されます。

```
    OFD_SetDetectionFrequency(OFD_PLL_ON,
        OFD_HIGHER_COUNT_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(例えば外部発振が範囲外になる、DEMO2 が定義されている)と OFD は CG リセット要因フラグをセットして、MCU をリセットします。
OFD リセットフラグを検知すると、MCU は内蔵発信にて動き始め、OFD 動作を無効化します。

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

LED1 と LED2 はシステムクロックの状態を示します。

LED の状態	意味
LED1 点滅	MCU は外部クロック発振にて正常動作しています。OFD が機能しており、異常周波数を検知することが可能です。
LED2 点滅	外部クロックが異常状態であり、OFD リセットを行い、MCU はデフォルトの内部クロックで動作します。また OFD は機能していません。

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

7-6 TMRB

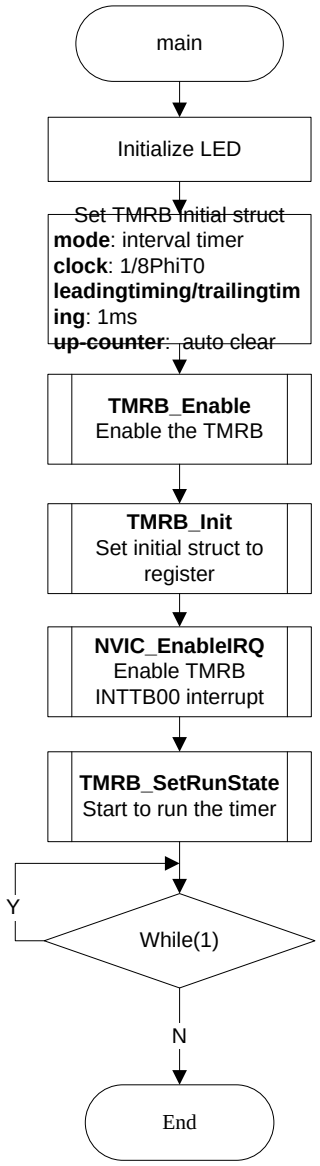
7-6-1 例: 汎用タイマー

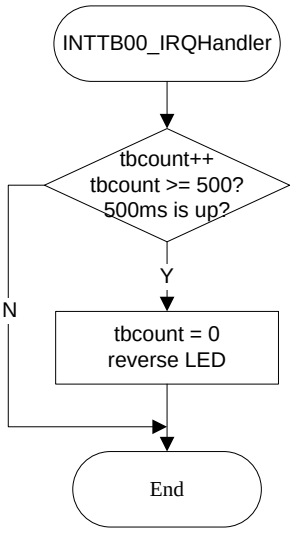
TX03 ペリフェラルドライバ(TMRB, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. TMRB の初期化
2. 1ms の汎用タイマー

- **フローチャート:**





- **サンプルプログラムのコードと説明:**

最初に LED チャネルを初期化します。

```
Init_Led();                         /* LED initialize */
```

TMRB 初期化構造体を用意し、TMRB モード、クロック、アップカウンタクリアメソッド、サイクル、デューティを設定します。本サンプルでは、サイクルとデューティを 1ms に設定し、下記条件を満たすため、マクロ TMRB_1MS は 0x1388 と同等です。 fphiT0 = fsys = fc = 10MHz * PLL = 80MHz, ftmrb = 1/8 fphiT0 = 10MHz, Ttmrb = 0.1us, 1ms/0.1us = 10000 = 0x2710 (クロック設定に関する詳細は、CG の項目をご覧ください。)

```
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。INTTB00 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(INTTB00_IRQn);                  /* enable INTTB00 interrupt */
TMRB_SetRunState(TSB_TB0, TMRB_RUN);           /* run TMRB0*/
```

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

```
tbcount++;
if (tbcount >= 500U) {                         /* 500ms is up */
    tbcount = 0U;
    /* reverse LED output */
    ledon = (ledon == 0U) ? 1U : 0U;
    if (0U == ledon) {
        Led_Off(LED1);
    } else {
        Led_On(LED1);
    }
} else {
    /* do nothing */
}
```

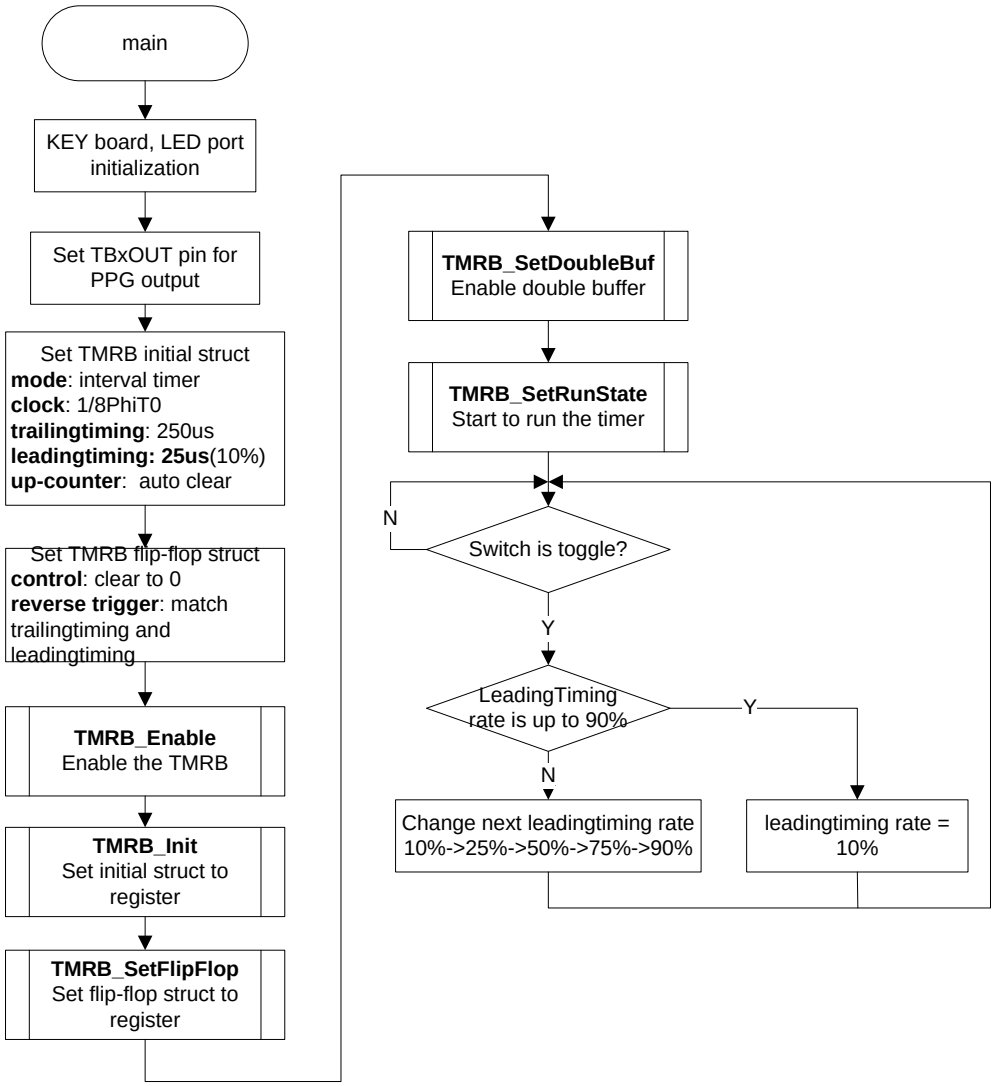
7-6-2 例: プログラマブル矩形波(PPG)出力

TX03 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. TMRB の初期化
2. PPG 機能の設定と開始
3. PPG デューティの長生

- **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、KEY と LED を初期化し、 PF1 を PPG 出力の TB7OUT に設定します。

```
GPIO_SetInput(KEYPORT, GPIO_BIT_0);                    /* set KEY port to input */

Init_Led();                          /* LED initialize */

/* Set PF1 as TB7OUT for PPG output */
GPIO_SetOutput(GPIO_PF, GPIO_BIT_1);
GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_1, GPIO_BIT_1);
GPIO_SetPullUp(GPIO_PF, GPIO_BIT_1, ENABLE);          /* Enable pull up */
```

TMRB 初期化構造体を用意し、TMRB モード、クロック、アップカウンタクリアメソッド、 サイクル、デューティを設定します。本サンプルでは、サイクルとデューティを 250us に設定し、下記条件を満たすため、マクロ TMRB7TIME は 0x09C4 と同等です。fphiT0 = fsys = fc = 10MHz * PLL = 80MHz, ftmrb = 1/8 fphiT0 = 10MHz, Ttmrb = 0.1us, 250us/0.1us = 2500 = 0x09C4 (クロック設定に関する詳細は、CG の項目をご覧ください。)

```
TMRB_InitTypeDef m_tmrb;
```

```
m_tmrb.Mode = TMRB_INTERVAL_TIMER;        /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;           /* 1/8PhiT0 */
m_tmrb.TrailingTiming = TMRB7TIME;        /* trailing timing is 250us */
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
m_tmrb.LeadingTiming = LeadingTiming[Rate]; /* leading timing, initial value 10% */
```

フリップフロップ初期化構造体、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と一致するよう設定します。

```
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_TB7);
TMRB_Init(TSB_TB7, &m_tmrb);
TMRB_SetFlipFlop(TSB_TB7, &PPGFFInital);
/* enable double buffer */
TMRB_SetDoubleBuf(TSB_TB7,ENABLE, TMRB_WRITE_REG_SEPARATE);
TMRB_SetRunState(TSB_TB7, TMRB_RUN);
```

キーが有効な場合、下記のようにデューティを設定してください。10%->25%->50%->75%->90%。その後 90% から、また 10% になります。

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

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

デューティ値の計算方法:
サイクル = 250us, ftmrb = 1/8 fphiT0 = 10MHz, Ttmrb = 0.1us (CG 設定により時間のパラメータは異なります。)

デューティ = 10%: 高レベル時間は 250*10% = 25us, 低レベル時間は 250-25 = 225us, カウンタ値 = 225us/Ttmrb = 0x8CA
デューティ = 25%:高レベル時間 250*25% = 62.5us, 低レベル時間は 250-62.5 = 187.5us, カウンタ値= 187.5us/Ttmrb = 0x753

デューティ = 50%:高レベル時間 250*50% = 125us, 低レベル時間は 250-125 = 125us, カウンタ値= 125us/Ttmrb = 0x4E2

デューティ = 75%:高レベル時間 250*75% = 187.5us, 低レベル時間は 250-187.5 = 62.5us, カウンタ値= 62.5us/Ttmrb = 0x271

デューティ = 90%:高レベル時間 250*90% = 225us, 低レベル時間は 250-225 = 25us, カウンタ値 e = 25us/Ttmrb = 0xFA

デューティ列の計算方法:

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

7-7 SIO/UART

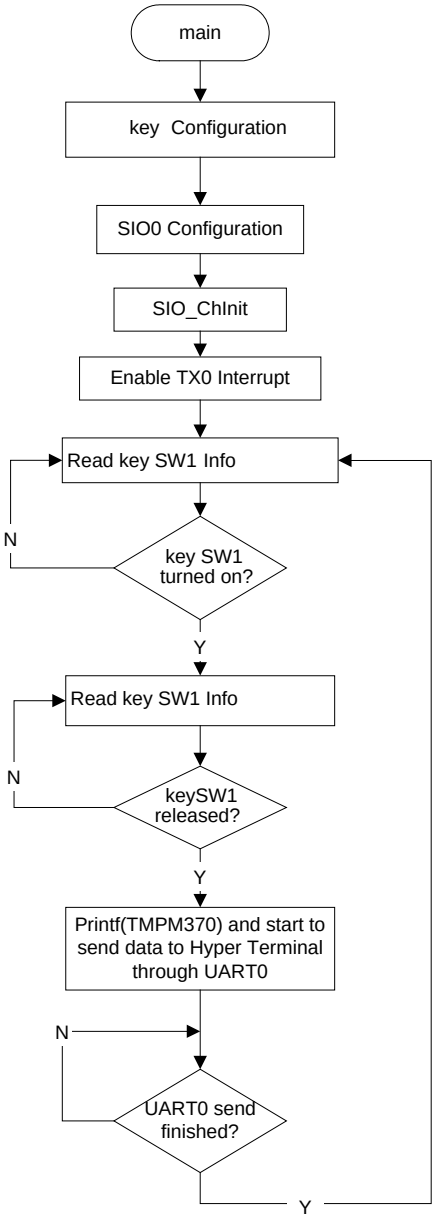
7-7-1 例: リターゲット

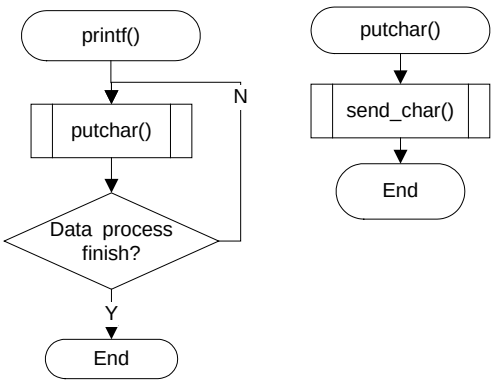
TX03 ペリフェラルドライバ (UART, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. UART 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
key Configuration
SIO0 Configuration
SIO_ChInit
Enable TX0 Interrupt
Read key SW1 Info
key SW1 turned on?
N
Y
Read key SW1 Info
keySW1 released?
N
Y
Printf(TMPM370) and start to send data to Hyper Terminal through UART0
N
UART0 send finished?
Y


printf()
putchar()
Data process finish?
N
Y
End
putchar()
send_char()
End

- **サンプルプログラムのコードと説明:**

まず、Key と SIO0 設定を行い、UART0 を初期化します。
GPIO ペリフェラルドライバを使い、GPIO を Key に設定します。

```
    GPIO_SetPullUp(TSW_PORT, GPIO_BIT_0, ENABLE);
    GPIO_SetPullUp(TSW_PORT, GPIO_BIT_1, ENABLE);
    GPIO_SetPullUp(TSW_PORT, GPIO_BIT_2, ENABLE);
    GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_0, ENABLE);
    GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_1, ENABLE);
    GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_2, ENABLE);
```

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

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

UART_InitTypeDef 構造体を生成し、データを設定します。以下は設定例です。

```
    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有効にし、UART0 を初期化します。

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

上記設定を行い、その後、UART0 の TX 割り込みを有効にします。

```
    NVIC_EnableIRQ(INTTX0_IRQn);
```

次に Key 情報を取得します。

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

SW1 キーが押されたかどうかを判定し、SW1 キーが押された場合は SW1 キーが離されるのを待ちます。

```
    if (TSW_info == TSW1) {
    /* wait for SW1 released */
    do {
        wait_TSW1 = 0U;
        TSW_info = GPIO_ReadData(TSW_PORT) & TSWBITS;
        } while (TSW_info != TSWRELEASE);
    }
```

SW1 キーが離されると、printf()関数によって UART0 を介してデータ送信を開始します。

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

printf()関数は、IAR コンパイラは putchar()関数を、RealView コンパイラは fputc()関数を呼び出して、UART0 へデータ出力します。

```
    #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;
}
```

データフローの残りのプロセスは UART0 における TX 割り込みの ISR にて終了します。

UART0 における TX 割り込みルーチン:

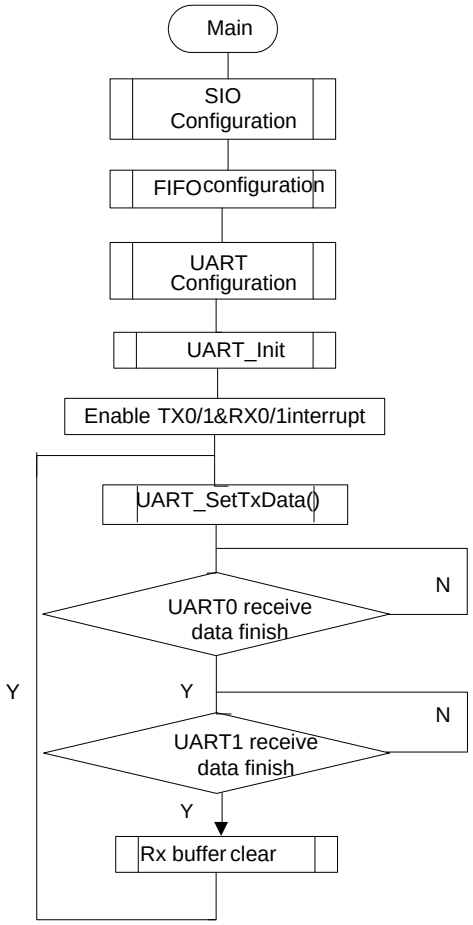
```
void INTTX0_IRQHandler(void)
{
      if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
            UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);/* send data */
            fSIO_INT = SET;               /* SIO0 INT is enable */
      } else {
            /* disable SIO0 INT */
            fSIO_INT = CLEAR;
            NVIC_DisableIRQ(INTTX0_IRQn);
            fSIOTxOK = YES;
      }
      if (gSIORdIndex >= gSIOWrIndex) {     /* reset buffer index */
            gSIOWrIndex = CLEAR;
            gSIORdIndex = CLEAR;
      } else {
            /* do nothing */
      }
}
```

7-7-2 例: UART FIFO

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

以下の例が含まれます。
1. UART と FIFO の設定と初期化
2. FIFO を使用した UART 送受信制御

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず、GPIO 設定と UART の初期化を行います。

GPIO ペリフェラルドライバを使い、GPIO を 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_PA->CR |= GPIO_BIT_5;
		TSB_PA->FR1 |= GPIO_BIT_5;
		TSB_PA->FR1 |= GPIO_BIT_6;
		TSB_PA->IE |= GPIO_BIT_6;
	}
}
```

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

```
	UART_InitTypeDef myUART;

    /* configure SIO0 for reception */
```

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

UART ペリフェラルドライバを使い、UART のチャネル 0 と 1 を初期化します。

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

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

FIFO の初期化を行います。

```
    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 と 1 の割り込みを有効にします。

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

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

データフローの残りのプロセスは UART0 と 1 の送受信割り込みの ISR にて終了します。
UART0 の送信割り込みルーチン:

```
void INTTX0_IRQHandler(void)
{
	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:

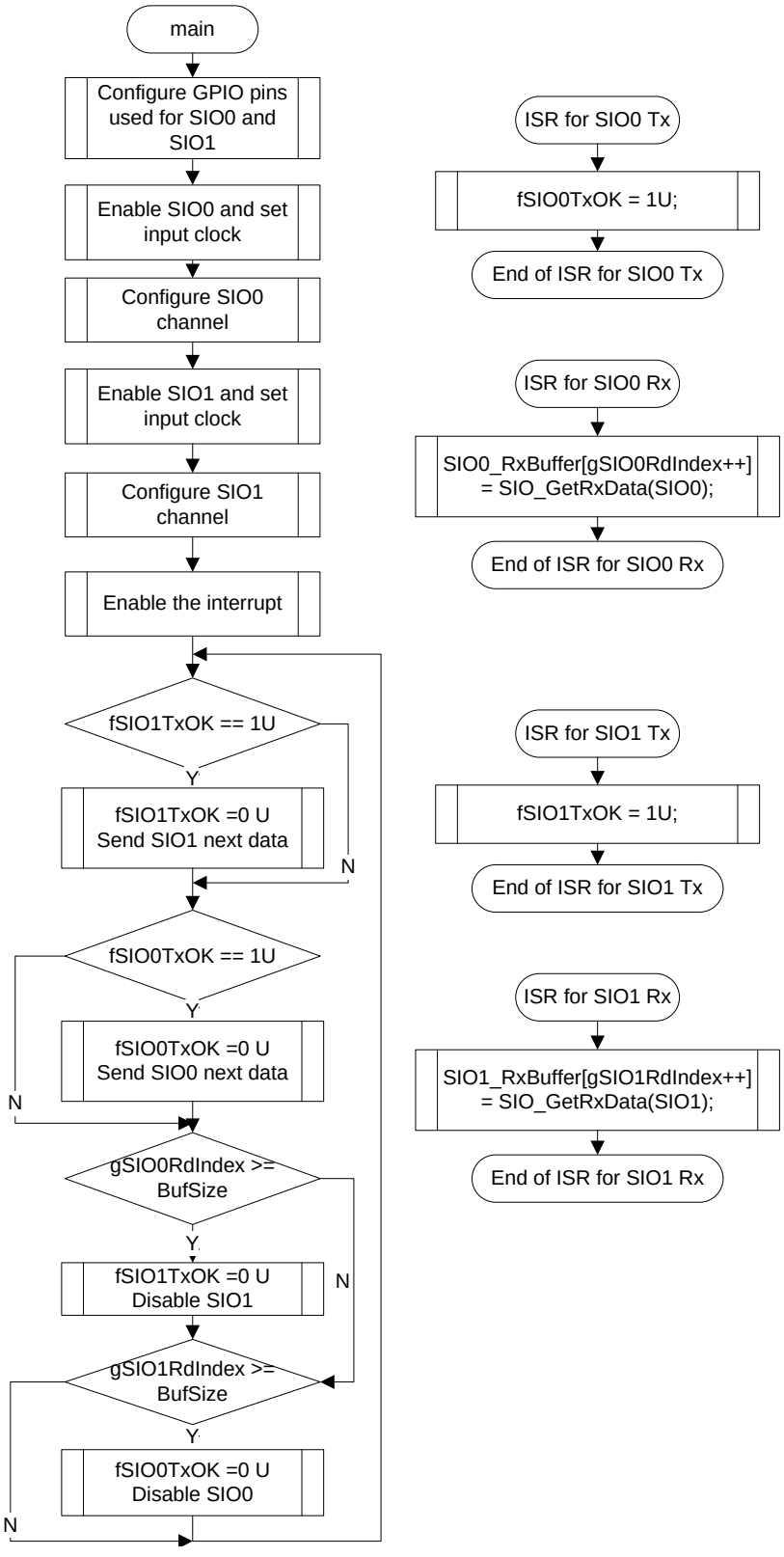
```
void INTRX1_IRQHandler(void)
{
	volatile UART_Err err;

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

7-7-3 例:SIO

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

以下の例が含まれます。
1. SIO の基本設定
2. 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 設定と SIO の初期化を行います。

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

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

/*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);
```

上記設定を行い、その後、SIO の送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、送信設定を行います。

```
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);
}
```

7-8 VLTD

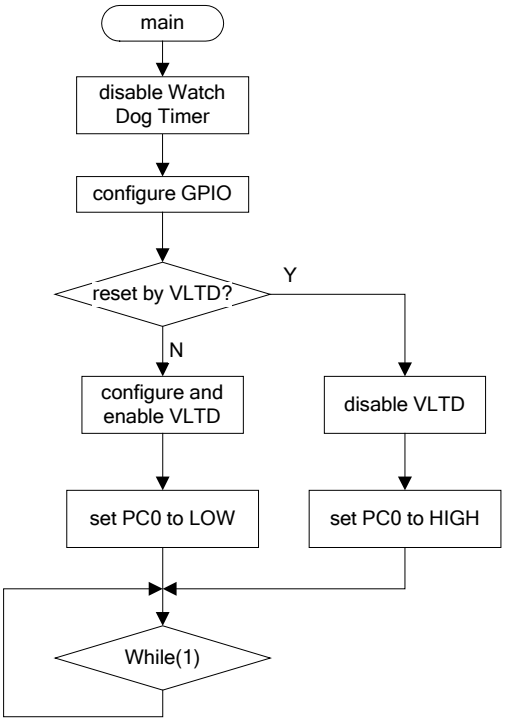
7-8-1 例: VLTD リセット

TX03 ペリフェラルドライバ(VLTD, CG, GPIO, WDT)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. VLTD 設定

- **フローチャート:**



- **サンプルプログラムのコードと説明:**

まず、ウォッチドッグタイマを禁止します。その後、PC0 を出力ポートに設定します。

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

その後、VLTD リセットフラグを読み出し、もし VLTD リセットされていなければ、VLTD の設定と有効化を行い、PC0 を LOW レベルにします。

```
if (CG_GetResetFlag().Bit.VLTDReset == 0U) {
        VLTD_SetVoltage(VLTD_DETECT_VOLTAGE_46);
        VLTD_Enable();
        GPIO_WriteData(GPIO_PC, 0x00);
```

VLTD リセットされている場合は、VLTD を無効にし、PC0 を HIGH レベルにします。

```
} else {
    VLTD_Disable();
    GPIO_WriteData(GPIO_PC, 0x01);
}
```

7-9 WDT

TX03 ペリフェラルドライバ(WDT, CG, 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、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();
```

7-10 ENC

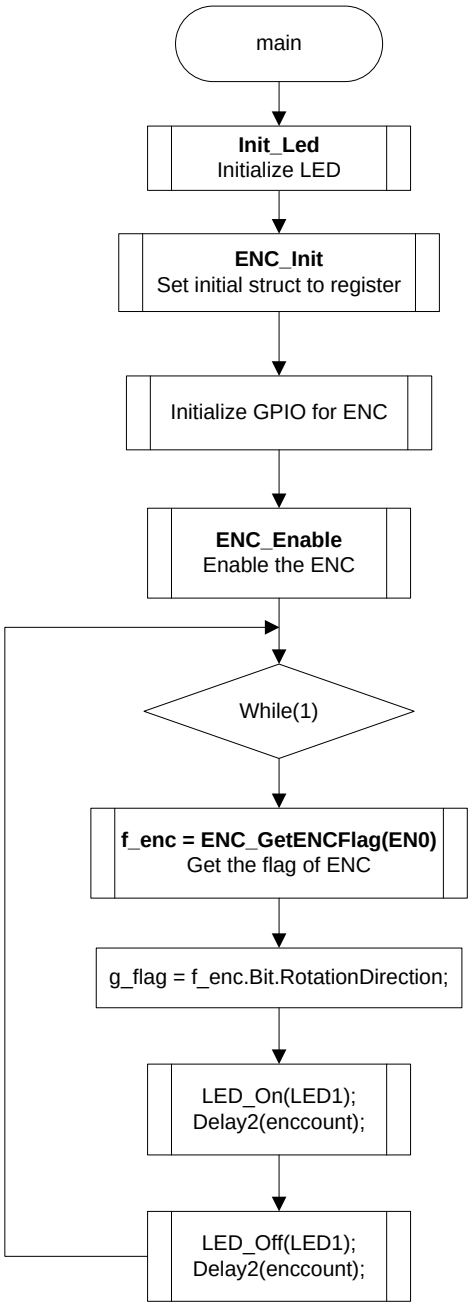
7-10-1 例: 回転検出

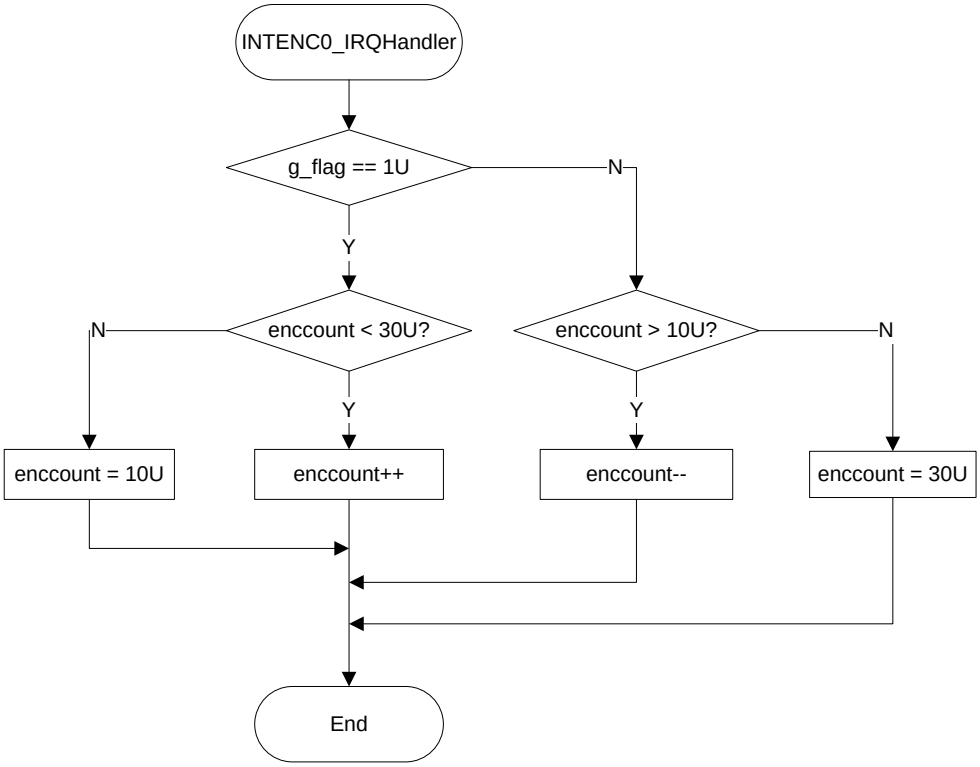
TX03 ペリフェラルドライバ(ENC, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. ENC0 の初期化
2. ホイールマウスの回転検出

- **フローチャート**


main
Init_Led
Initialize LED
ENC_Init
Set initial struct to register
Initialize GPIO for ENC
ENC_Enable
Enable the ENC
While(1)
f_enc = ENC_GetENCFlag(EN0)
Get the flag of ENC
g_flag = f_enc.Bit.RotationDirection;
LED_On(LED1);
Delay2(enccount);
LED_Off(LED1);
Delay2(enccount);



- **サンプルプログラムのコードと説明**

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

まず、評価ボード上の LED を初期化します。

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

ENC 初期化構造体を用意し、ENC0 を初期化します。

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

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

GPIO を ENC0 に設定します。PD0 と PD1 は ENC0 入力に設定します。

```
    /* GPIO initialization */
    GPIO_SetInputEnableReg(GPIO_PD, GPIO_BIT_0, ENABLE);/*Set PD0 as
input */
    GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_1, GPIO_BIT_0);/*Set
PD0 as ENCA0 */
    GPIO_SetInputEnableReg(GPIO_PD, GPIO_BIT_1, ENABLE);/*Set PD1 as
```

```
input */
    GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_1, GPIO_BIT_1);/*Set
PD1 as ENCB0 */
```

ENC0 と ENC0 割り込みを許可します。

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

ホイールマウスの回転を検出するため ENC0 の回転方向検出状態を取得します。回転数に応じて LED1 を点滅します。

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

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

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

7-11 PMD

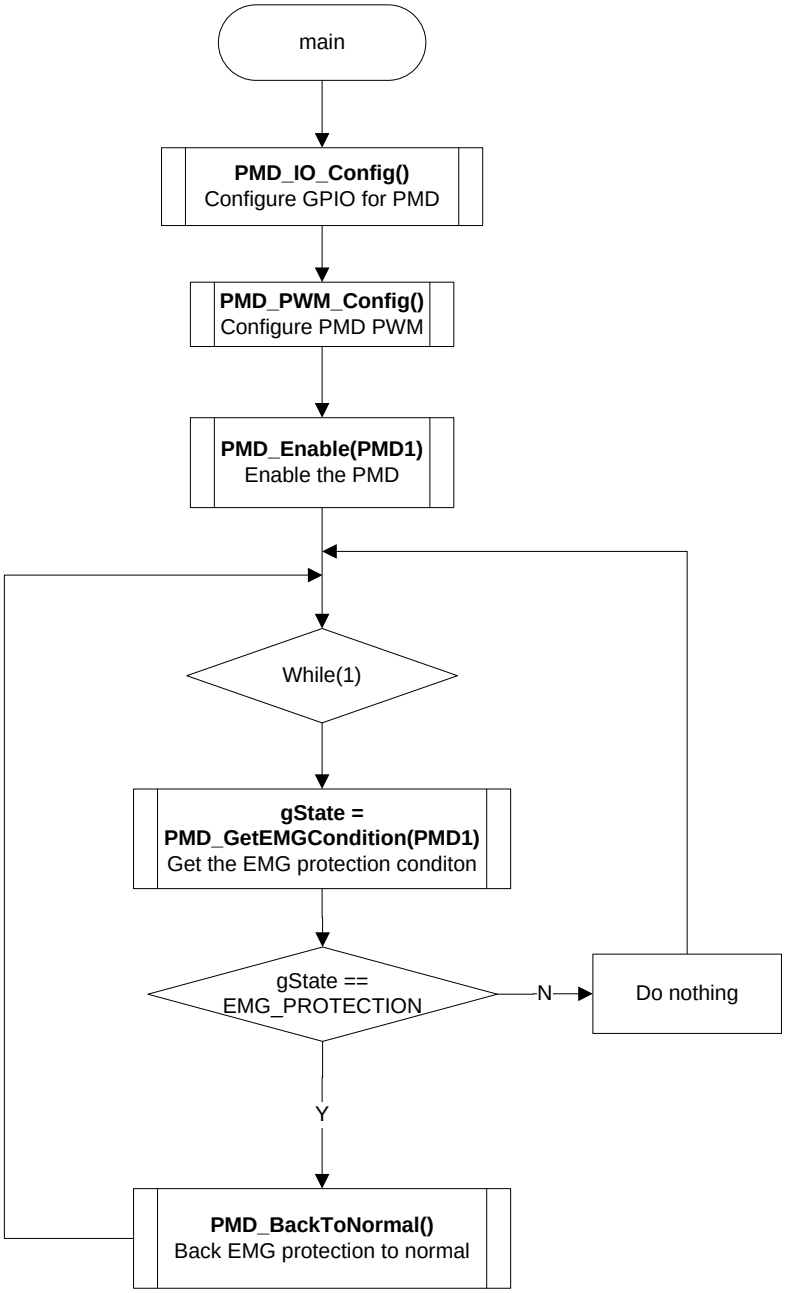
7-11-1 例: 位相出力

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

この例では以下を行います。

1. PMD と GPIO の初期化
2. EMG 保護検出出力
3. EMG 保護検出出力から通常出力への状態変化

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず LED、PMD、GPIO の初期化を行います。

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

/* GPIO configuration*/
PMD_IO_Config();

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

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

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

    PMD_Init(PMD1,&m_pmd);
```

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

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

PMD 動作を許可します。

```
    PMD_Enable(PMD1);
```

EMG 保護検出の判定を行います。保護検出状態の場合は LED1 を点灯し、EMG 保護検出出力から通常出力へ戻します。保護検出状態ではない場合は LED1 を消灯します。

```
    while(1){
        /* Get the EMG protection condition*/
        gState = PMD_GetEMGCondition(PMD1);
        /* Judge the EMG protection condition */
        if (gState == EMG_PROTECTION) {
            /* LED1 will light on if the condition is protection */
            Led_On(LED1);
            /* Back EMG protection to normal */
            PMD_BackToNormal();
            Delay(1000U);
        } else {
            /* LED1 will light off if the condition is not protection */
            Led_Off(LED1);
        }    }
```