TOSHIBA

TX00 ペリフェラルドライバ使用例 (TMPM061)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M061UE-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。

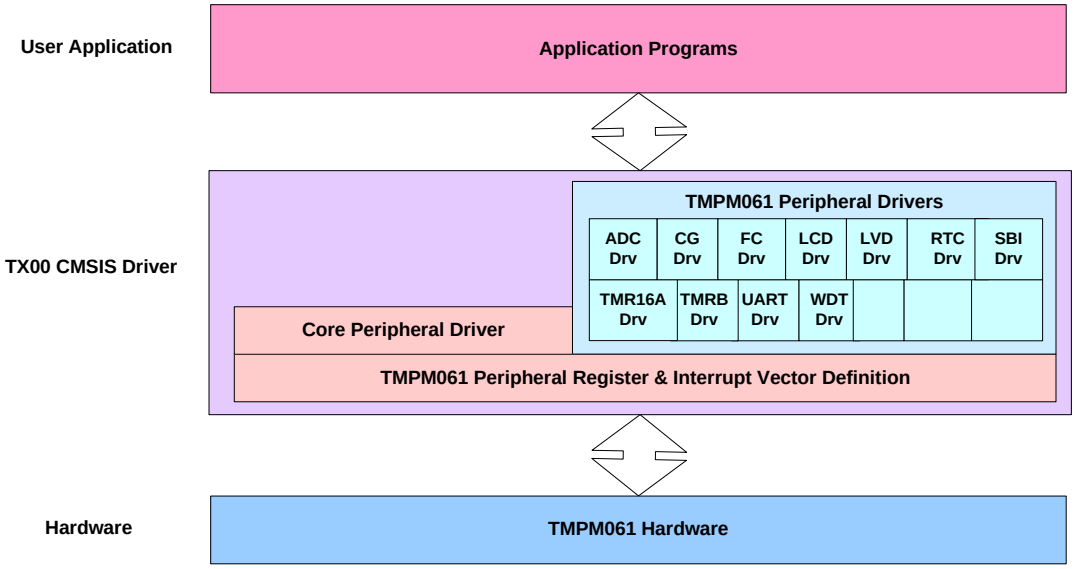
目次

1 はしがき

本サンプルプログラムは、東芝製マイコンTMPM061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

TX00 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
CG	クロックギア	使用(CG サンプル)
Standby mode	-	使用(SLEEP モードサンプル)
External interrupt	INT0	使用(CG デモにおける SLEEP モードからの復帰)
	INT1	未使用
	INT2	未使用
	INT3	未使用
SIO	SIO0	使用(UART リターゲットサンプル)
	SIO1	未使用
	SIO2	未使用

機能	チャネル	使用/未使用
	SIO3	使用(UART リターゲットサンプル、SIO サンプル)
16-bit timerB	TMRB0	使用(TMRB 汎用タイマサンプル、PPG サンプル)
	TMRB1	未使用
16-bit timerA	TMR16A0	使用(TMR16A 汎用タイマサンプル)
	TMR16A1	未使用
	TMR16A2	未使用
	TMR16A3	未使用
	TMR16A4	未使用
	TMR16A5	未使用
	TMR16A6	未使用
RTC	RTC	使用(RTC サンプル)
ADC	AIN0	使用(ADC サンプル)
	AIN1	未使用
DSADC	DSAIN0	使用(DSADC サンプル)
	DSAIN1	使用(DSADC サンプル)
	DSAIN2	使用(DSADC サンプル)
SBI	SBI0	使用(マスタ送信/スレーブ受信)
WDT	WDT0	使用(WDT サンプル)
LCD	LCD	使用(LCD サンプル)
LVD	LVD	使用(LVD サンプル)

4 端子用途

本サンプルプログラムは、開発環境に東芝製TMPM061用評価ボードを用いてテストされています。以下に、端子用途を説明します。

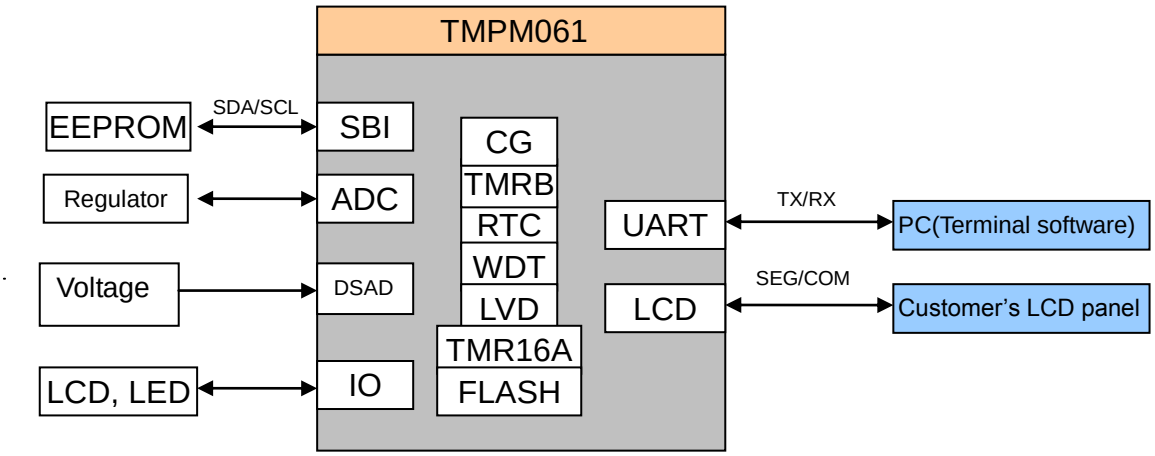
No	Name	Usage
1	VREFH	10bit ADコンバータ用基準電源。(DVDD3と接続)
2	AVDD3	10bit ADコンバータ用電源供給。(DVDD3と接続)
3	MODE	MODE端子。(GNDと接続)
4	XT1	未使用
5	XT2	未使用
6	#RESET	プルアップ、ノイズフィルタ付リセット端子。(Lowアクティブ)
7	X1	高速発振子
8	DVSS	GND端子
9	X2	高速発振子
10	DVDD3	電源端子
11	INT2/TB0OUT/PG0	PPG出力用TB0OUT
12	PH0/IROUT0/TXD0/	UART0 TX / LED0/SIO
13	PH1/RXD0	UART0 RX/ LED1/SIO
14	PH2/SCLK0/#CTS0/T16A0OUT	LED2/SCLK0
15	PH3/TXD1/IROUT1/#DBGEN0	未使用
16	PH4/RXD1	未使用
17	PH5/SCLK1/#CTS1/T16A1OUT	未使用
18	PI0/TXD2/IROUT2/#DBGEN1	未使用
19	PI1/RXD2	未使用
20	PI2/SCLK2/#CTS2/T16A2OUT	未使用
21	PI3/TB0IN	未使用
22	COUT	1.5V出力
23	RVSS	GND端子
24	RVDD3	電源端子
25	PI4/TXD30	TXD3
26	PI5/RXD30	RXD3
27	PI6/SCLK30/#CTS30/T16A5OUT	SCLK3
28	PJ0/SDA0/SO0/#BOOT	SBI用SDA0
29	PJ1/SCL0/SI0	SBI用SCL0
30	PJ2/SCK0/INT1	未使用
31	PJ3/RTCOUT	未使用

No	Name	Usage
32	PJ4/T16A3OUT/SCOUT	未使用
33	PJ5/TB1IN/XTCLKIN	未使用
34	PE7/SEG39/SWDIO0	SEG39
35	PE6/SEG38/SWCLK0	SEG38
36	PE5/SEG37/TXD31	SEG37
37	PE4/SEG36/RXD31	SEG36/SWS
38	PE3/SEG35/SCLK31/#CTS31	SEG35/SW37
39	PE2/SEG34/T16A6OUT	SEG34/SW26
40	PE1/SEG33/SWDIO1	SEG33/SW15
41	PE0/SEG32/SWCLK1	SEG32/SW04
42	PD7/SEG31	SEG31/LCDDB7
43	PD6/SEG30	SEG30/LCDDB6
44	PD5/SEG29	SEG29/LCDDB5
45	PD4/SEG28	SEG28/LCDDB4
46	PD3/SEG27	SEG27/LCDDB3
47	PD2/SEG26	SEG26/LCDDB2
48	PD1/SEG25	SEG25/LCDDB1
49	PD0/SEG24	SEG24/LCDDB0
50	PC7/SEG23	SEG23
51	PC6/SEG22	SEG22
52	PC5/SEG21	SEG21
53	PC4/SEG20	SEG20
54	PC3/SEG19	SEG19
55	PC2/SEG18	SEG18/LDRS
56	PC1/SEG17	SEG17/LCDRW
57	PC0/SEG16	SEG16/LCDE
58	PB7/SEG15	SEG15
59	PB6/SEG14	SEG14
60	PB5/SEG13	SEG13
61	PB4/SEG12	SEG12
62	PB3/SEG11	SEG11
63	PB2/SEG10	SEG10
64	PB1/SEG9	SEG9
65	PB0/SEG8	SEG8
66	PA7/SEG7	SEG7
67	PA6/SEG6	SEG6
68	PA5/SEG5	SEG5
69	PA4/SEG4	SEG4
70	PA3/SEG3	SEG3
71	PA2/SEG2	SEG2
72	PA1/SEG1	SEG1

No	Name	Usage
73	PA0/SEG0	SEG0
74	DVDD3	電源端子
75	DVSS	GND端子
76	COM0	LCDコモン出力端子
77	COM1	LCDコモン出力端子
78	COM2	LCDコモン出力端子
79	COM3	LCDコモン出力端子
80	PK0/INT3/LV1	未使用
81	PK1/TB1OUT/LV2	未使用
82	VLC	LCD電源端子
83	AGNDREF0	24bit ΔΣAD変換用GND端子
84	DAIN0+	24bit ΔΣAD変換用アナログ入力端子
85	DAIN0-	24bit ΔΣAD変換用アナログ入力端子
86	VREFIN0	24bit ΔΣAD変換用電源端子
87	AGNDREF1	24bit ΔΣAD変換用GND端子
88	DAIN1+	24bit ΔΣAD変換用アナログ入力端子
89	DAIN1-	24bit ΔΣAD変換用アナログ入力端子
90	VREFIN1	24bit ΔΣAD変換用電源端子
91	AGNDREF2	24bit ΔΣAD変換用GND端子
92	DAIN2+	24bit ΔΣAD変換用アナログ入力端子
93	DAIN2-	24bit ΔΣAD変換用アナログ入力端子
94	VREFIN2	24bit ΔΣAD変換用GND端子
95	DSRVSS	基準電圧回路用GND端子
96	DSRVDD3	アンプ用電源端子
97	SRVDD	基準電圧用電源端子
98	PF0/AIN0	ADC用AIN0
99	PF1/AIN1/INT0	CGサンプルにおけるSLEEPモード解除用INT0
100	AVSS	10bit AD変換用GND端子

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:
   - TMPM061 用評価ボード(非売品)
2. 開発ツール:
   - IAR:
     1) J-Link: IAR J-Link-7.0
     2) IDE: IAR Embed Workbench for ARM version 6.40
   - KEIL:
     1) u-Link: Realview ULINK2
     2) IDE: KEIL uVision MDK version 4.21

6 機能

動作モード選択

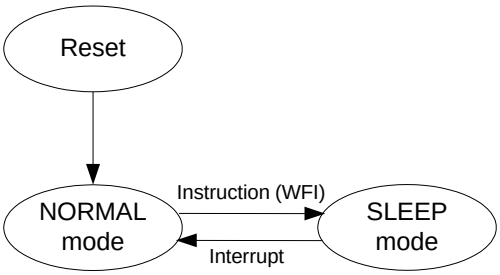
本デバイスは 5 つの動作モードがあります: NORMAL, SLOW, IDLE, SLEEP, STOP
IDLE, SLEEP, STOP モードは低消費電力モードです。
低消費電力モードに移行するには、システム制御レジスタ STBYCR0<STBY2:0>にて IDLE, SLEEP, STOP モードを選択し、WFI (Wait For Interrupt) 命令を実行します。

- **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(fosc1、または fosc2)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。低速クロックを動作させることも可能です。
補足: サンプルプログラムでは SLEEP モードのみとなります。

- **SLEEP モード:**

高速クロックが停止するモードです。低速クロックで動作する RTC のカウンタは動作を継続します。
SLEEP モードが解除されると、SLEEP 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。
SLEEP モードの解除要因は、RTC 割り込みと外部割込みです。
補足: SLEEP モードのサンプルプログラムは、CG ドライバを使用しています。



6-1 ADC

ADC 入力チャネル AIN0(pin98)に接続された *VR2* の値を変換します。この電圧を ADC で測定し、測定結果を標準出力に表示します。 標準出力を UART0 に接続すると、ホストと UART0 を接続しておけば、変換結果をターミナルソフト表示することができます。

UART の設定: ボーレート = 115200, パリティ無し, 8bit データ, 1bit のストップビット

pin1(VREFH)と pin2(AVDD3)を pin10、または pin74 (DVDD3)と接続します。
pin100(AVSS)を pin8、または pin75(DVSS)と接続します。

6-2 CG

6-2-1 動作モード変更

動作モードを変更します。NORMAL と STOP の 2 モードを使用します。2 つのモードを切り替えるためにスイッチを使用します。

現在のモード	アクション(スイッチ)	動作	LED 表示
NORMAL	SW1	NORMAL→ STOP	LED 0,1,2,3 を消灯します。
STOP	SW1	STOP →NORMAL	LED 0,1,2,3 を点灯します。

補足: PF1/ PIN 99 (INT0)を PJ24/PIN4 と接続します。

6-3 DSADC

この例は、DSADC と UART のドライバを使用したサンプルプログラムです。
3 つの DSAD 変換の値を取得し、その値を U 標準出力します。

以下の例を行います。

1. DSADC 用ソフトウェアトリガ
2. DSADC 状態と変換結果の取得

補足:

端子接続:

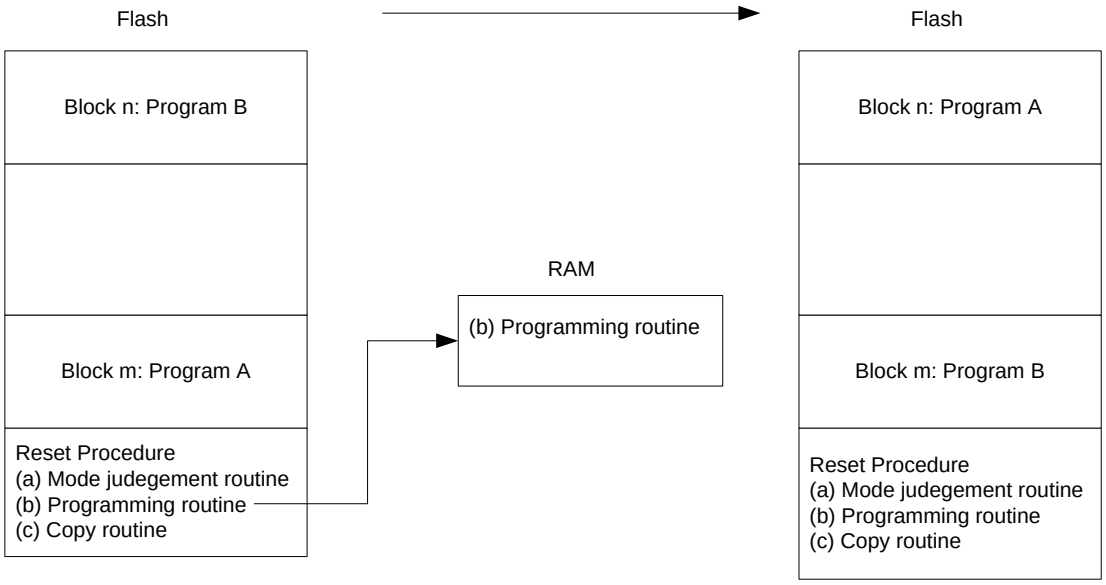
1. 入力電圧
   (a) pin84(DAIN0+)/pin85(DAIN0-)と電圧(-0.375V<入力電圧<+0.375V)を接続
   (b) pin88(DAIN1+)/pin89(DAIN1-)と電圧(-0.375V<入力電圧<+0.375V)を接続
   (c) pin92(DAIN0+)/pin93(DAIN0-)と電圧(-0.375V<入力電圧<+0.375V)を接続
2. pin83(AGNDREF0), pin87(AGNDREF1), pin91(AGNDREF2)を GND に接続します。
3. pin86(VREFIN0)と 1uF コンデンサが接続された pin'+'を接続し、コンデンサが接続された pin'-'と pin83(AGNDREF0)を接続します。
4. pin 90(VREFIN1)と 1uF コンデンサが接続された pin'+'を接続し、コンデンサが接続された pin'-'と pin87(AGNDREF1)を接続します。
5. pin94(VREFIN2)と 1uF コンデンサが接続された pin'+'を接続し、コンデンサが接続された '-'と pin91(AGNDREF2)を接続します。
6. pin96(DSRVDD3)と pi 97(SRVDD)を Vcc と接続します。
7. pin95(DSRVSS)を GND と接続します。

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: LED0 点滅、LCD 表示"DEMO A"
B: LED1 点滅、LCD 表示"DEMO B")
初期状態は、プログラム A が最初に動作します。
ーSW0 キーはリセット動作のモード判定に使います。

SW0 キーが押された場合 → ユーザブートモードです。

SW0 キーが押されていない場合 → Normal モードです。



・動作シーケンス

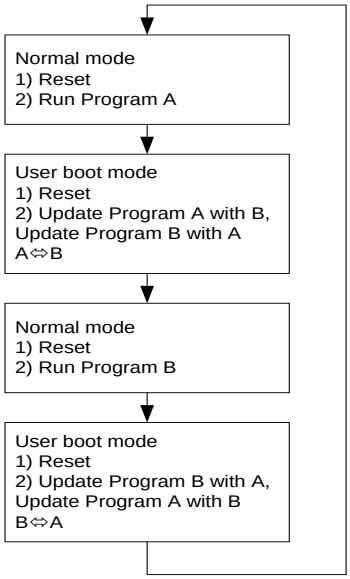
(1) 電源投入
評価ボードのリセット動作が行われます。
初期プログラム A がフラッシュ ROM へ格納され動作します。
LED0 が点滅し、LCD に "DEMO A" を表示します。

(2) SW0 キーを押している間にリセットを押し、その後 LCD に"User Boot Mode"が表示されるまでに SW0 キーを離します。
リセット処理ルーチンを実行します: 書き込みプログラムが転送ルーチンによって RAM へ転送されます。その後、フラッシュ内のプログラム A と B が入れ替わります。
この間、LCD に次のような情報を出力します。"RAM transferring ……", "FLASH swapping ……", "Finished Restart ……"

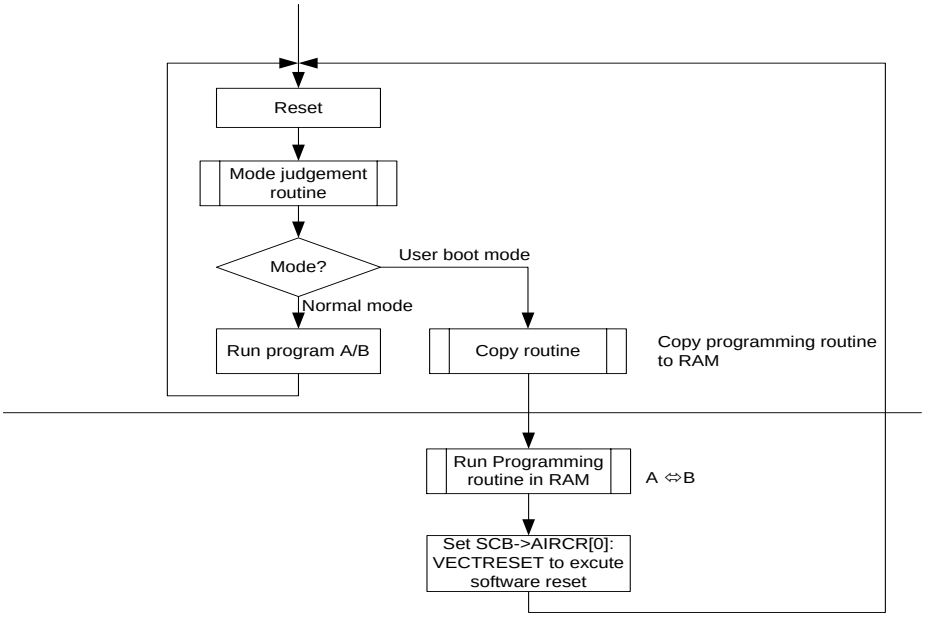
(3) LCD に “Finished Restart ……”が出力された場合、評価ボードは自動的にソフトウェアによってリセットします。
その後、フラッシュ ROM へ格納されたプログラム B が動作します。
LED1 が点滅し、LED に”DEMO B”を表示します。

(4) SW0 を押しながらリセットします。LCD に“User Boot Mode”が出力されるまでに SW0 を話してください。その後 SW0 を離すと、手順(2)を行います。

(5) LCD に “Finished Restart ……”を出力すると、評価ボードはソフトウェアによって自動的にリセットします。
その後、プログラム A がフラッシュ ROM へ格納され、動作します。
LED0 が点滅し、LCD に“DEMO B”を表示します。



・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、LCD に下記を出力します。

User Boot Mode

(リセットシーケンス)
RAM 転送もしくはフラッシュ書き込み中は、LCD に現在の処理を出力します。
LCD 表示例

RAM transferring ……

(コピールーチン)

FLASH swapping ……

(プログラミングルーチン)

Finished Restart ……

(プログラミングルーチン)

6-5 LCD

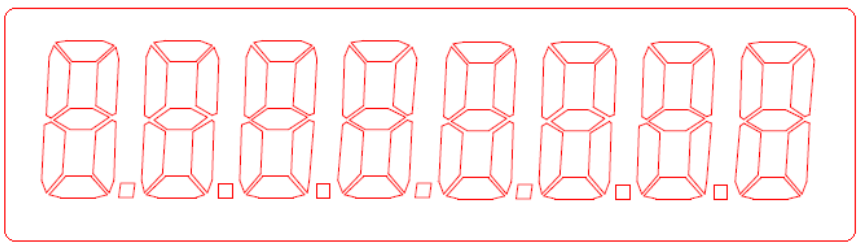
M061 の LCD ドライバを使った 2 つの LCD サンプルがあります。

6-5-1 LCD demo1

外部の LED を使い、文字列‘12345678’の 8 桁を順番に表示します。
表示順は以下の通りです。

1. 右づめで 1 を表示します。
2. 右づめで 12 を表示します。
3. 右づめで 123 を表示します。
4. 右づめで 1234 を表示します。
5. 右づめで 12345 を表示します。
6. 右づめで 123456 を表示します。
7. 右づめで 1234567 を表示します。
8. 右づめで 12345678 を表示します。
9. 上記ステップ 1 に戻ります。

以下が簡易化した LCD パネルです。(デューティ 1/4, 1/3 バイアス)



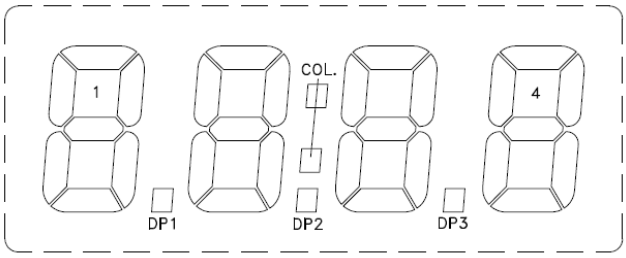
6-5-2 M061-SK LCD demo

TMPM061-SK のスティック LCD(型名: S5026D) を使い、文字列 ‘0123456789’ の 10 桁を順

番に表示します。
表示順は以下の通りです。

1. 表示　　0
2. 表示　　0 1
3. 表示　　0 1 2
4. 表示　　0 1 2 3
5. 表示　　1 2 3.4
6. 表示　　2 3.4 5
7. 表示　　3.4 5 6
8. 表示　　4 5 6 7
9. 表示　　5 6 7 8
10. 表示　6 7 8 9
11. ステップ 1 に戻る

以下が簡易化した LCD パネルです。(スタティック LCD):



6-6　LVD

この例は、LVD によって電圧の状態を検出するサンプルプログラムです。

電圧が通常の場合は LED0 を点滅させ、LED1 を OFF します。
電圧が検出電圧より低い場合は INTLVD を発生させ、LVD 割り込みハンドラ内で LED0 を OFF し、LED1 を点滅させます。

6-7　RTC

評価ボード:

この例は、内蔵 RTC を使用して LCD に日時を表示するサンプルプログラムです。
初期設定では 2010/10/22, 時刻の設定が 12:50:55, 24 時間表示です。
更新間隔: 1 秒
LCD 表示:

2010/10/22 12:50:55

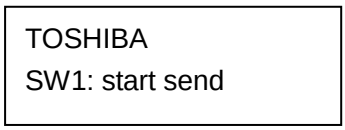
スタータキット:

この例は、LCD に時間と分を表示するサンプルプログラムです。
初期設定では 2010/10/22, 時刻の設定が 12:50:55, 24 時間表示です。
更新間隔: 1 秒
LCD 表示:

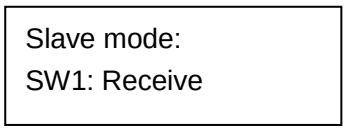
6-8 SBI

この例は、I2C バスの 1 本を I2C マスタとして、もう片方を I2C スレーブとして動作させ、SBI 割り込みを使用して I2C バスのリード/ライトを行うサンプルプログラムです。(評価ボード上の 2 本の I2C バス(SBI0 と SBI1)を接続します)
I2C スレーブ(SBI1)は I2C マスタ(SBI0)から“TOSHIBA”を受け取り、LCD に表示します。
プログラム開始時の LCD は次の通りです。

評価ボードの SW0 を ON するとマスタとして動作し、他方の評価ボードの SW2 を ON するとスレーブとして動作します。2 つの評価ボードの表示は次の通りです。

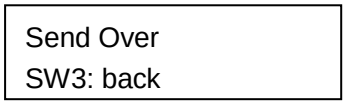


マスタ側の評価ボード

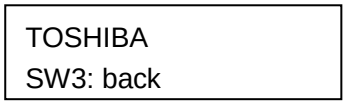


スレーブ側の評価ボード

2 つの評価ボードの SW1 を ON すると、スレーブ側が取得した文字列 “TOSHIBA” を表示します。



マスタ側の評価ボード



スレーブ側の評価ボード

補足: このサンプルソフトを実行ために評価ボードを以下のように接続します。
Connect the two pins PJ1 of M061 Evaluation board (SCL).
Connect the two pins PJ0 of M061 Evaluation board (SDA).
Connect two ground pins of the boards to make the two boards have the same ground.

補足: このサンプルソフトを実行ために評価ボードを以下のように接続します。
2 つの評価ボードの PJ1(SCL)端子同士を接続します。
2 つの評価ボードの PJ0(SDA)端子同士を接続します。

6-9 TMR16A

6-9-1 汎用タイマ

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

6-10 TMRB

6-10-1 汎用タイマ

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

6-10-2 PPG 波形出力

この例は、SW1 を使い、デューティ可変の PPG(プログラマブル矩形波)の波形を出力するサンプルプログラムです。
PPG 波形は、タイマ出力端子をオシロスコープに接続することで確認できます。
デューティは 5 段階(10%, 25%, 50%, 75%, 90%)に変更できます。
電源投入すると PPG モードに入り、PPG 波形出力を開始します。
キーを押すことでデューティーを変更します。
10% → 25% → 50% → 75% → 90% → 10%

6-11 SIO/UART

6-11-1 リターゲット

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

6-11-2 SIO

この例は、TMPM061 の SIO モジュール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 3 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 RXD3、TXD3 と RXD0、sclk0 と sclk3 を接続してください)

*補足: この例では、スレーブ側を最初に動作させ、その後、マスタ側を動作させてください。

6-12 WDT

ウォッチドッグタイマは、高速クロックが停止する STOP モードでは使用できません。リセット後にウォッチドッグタイマは有効となり、SystemInit()関数で無効になります。

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

DEMO1:

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

DEMO2:

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

7 ソフトウェア

本ソフトウェアは、TMPM061 MCU の主要機能を評価ボード上でデモ、動作確認するためのサンプルプログラムです。

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

ワークスペース構造とプロジェクト名は、以下の通りです:

```
\---M061
    +---Libraries
    |   +---TX00_CMSIS
    |   |   |   system_TMPM061.c
    |   |   |   system_TMPM061.h
    |   |   |   TMPM061.h
    |   |   \---startup
    |   |       +---arm
    |   |       |       startup_TMPM061.s
    |   |       \---iar
    |   |               startup_TMPM061.s
    |   \---TX00_Periph_Driver
    |       +---inc
    |       |       tmpm061_adc.h
    |       |       tmpm061_cg.h
    |       |       tmpm061_dsad.h
    |       |       tmpm061_fc.h
    |       |       tmpm061_lcd.h
    |       |       tmpm061_lvd.h
    |       |       tmpm061_rtc.h
    |       |       tmpm061_sbi.h
    |       |       tmpm061_tmr16a.h
    |       |       tmpm061_tmrb.h
    |       |       tmpm061_uart.h
    |       |       tmpm061_wdt.h
    |       |       tx00_common.h
    |       \---src
    |               tmpm061_adc.c
    |               tmpm061_cg.c
    |               tmpm061_dsad.c
    |               tmpm061_fc.c
```

```
|                   tmpm061_lcd.c
|                   tmpm061_lvd.c
|                   tmpm061_rtc.c
|                   tmpm061_sbi.c
|                   tmpm061_tmr16a.c
|                   tmpm061_tmrb.c
|                   tmpm061_uart.c
|                   tmpm061_wdt.c
\---Project
    +---例:s                //ここでは 1 つの例みのを記載します。
    |    +---ADC
    |    |    \---ADC_Data_Read
    |    |         +---App
    |    |         |         main.c
    |    |         +---IAR
    |    |         |         ADC_Data_Read.ewd
    |    |         |         ADC_Data_Read.ewp
    |    |         |         ADC_Data_Read.eww
    |    |         \---KEIL
    |    |                    ADC_Data_Read.uvopt
    |    |                    ADC_Data_Read.uvproj
    |    \---Workspace
    |         +---IAR
    |         |         例:s_for_M061_Driver_IAR.eww
    |         \---KEIL
    |                   例:s_for_M061_Driver_Keil.uvmpw
    \---Template
         +---IAR
         |         TMPM061_Flash.icf
         |         TMPM061_RAM.icf
         \---KEIL
                   tmpm061.sct
```

7-1 ADC

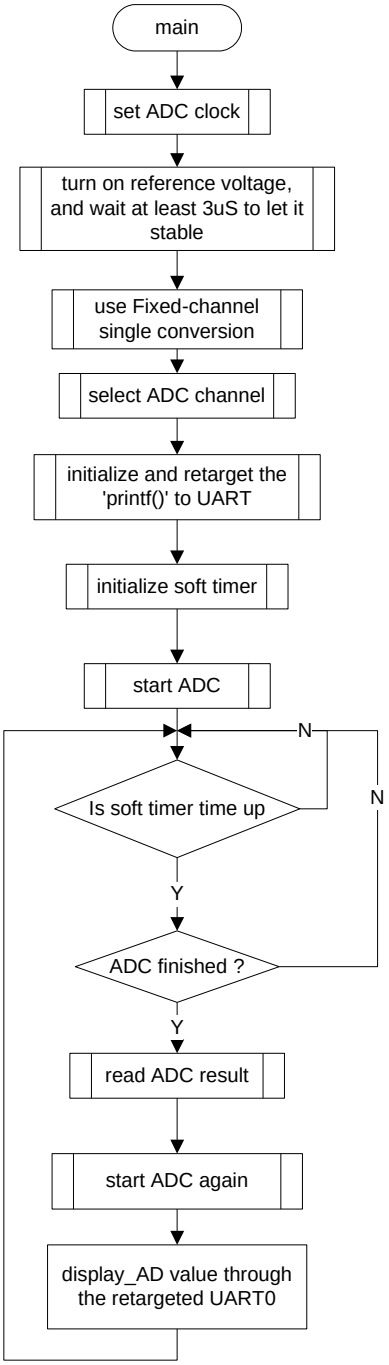
7-1-1 例: ADC Data Read

TX00 ペリフェラルドライバ(ADC, UART)を用いたサンプルプログラムです。

この例では以下を行います。

1. ADC 設定と初期化を行います。
2. AD 変換の開始と AD 変換結果の読み出しを行います。

- **フローチャート:**



- **サンプルプログラムのコードと説明:**

最初に AD コンバータクロックの選択を行い、次にリファレンス電源を ON します。

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

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

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

チャネル固定シングル変換の設定を行い、AD チャネルを選択します。

```
/* 3. use Fixed-channel single conversion */
ADC_SetScanMode(DISABLE);
ADC_SetRepeatMode(DISABLE);

/* 4. select ADC channel */
ADC_SetInputChannel(ADC_CHANNEL);
```

その後、AD 変換を開始します。

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

“printf()”とソフトウェアタイマの初期化を行います。

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

/* initialize soft timer */
softTimer_Init();
```

AD 変換を開始し、AD 変換状態をチェックします。AD 変換フラグがセットされると、ADC_Display()をコールし、次の AD 変換を開始します。

```
while (1U) {
    softTimer_Run();
    if(softT.flag_TimeUp) {
        softT.flag_TimeUp = 0U;

        /* check ADC module state */
        adcState = ADC_GetConvertState();
        if (adcState == DONE) {
            /* read ADC result when it is finished */
            adResult = ADC_GetConvertResult(ADC_REG_RESULT);
            myResult = (uint32_t)adResult.Bit.ADResult ;
            ADC_Start();
            display_AD(myResult);
        }
    } else {
        /* Do nothing */
    }

}
```

display_AD()関数では、 “printf()”を使いホスト PC に AD 変換結果を送信します。

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

7-2 CG

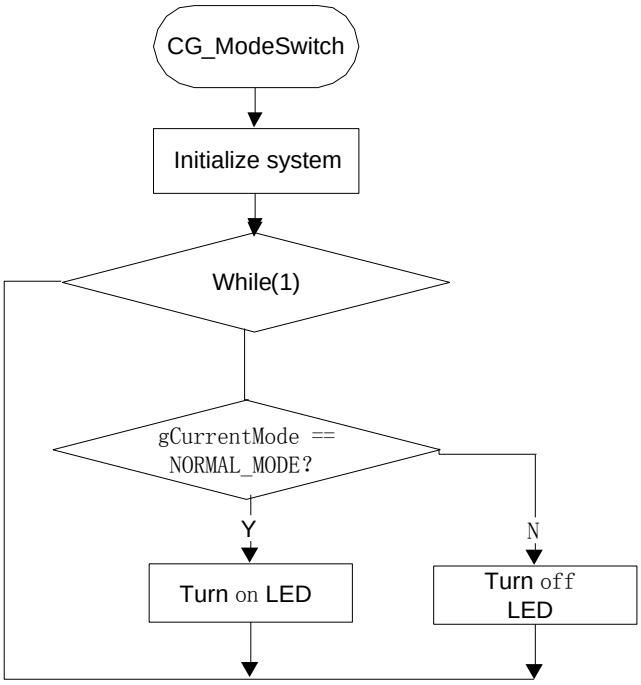
7-2-1 例: パワーモード変更

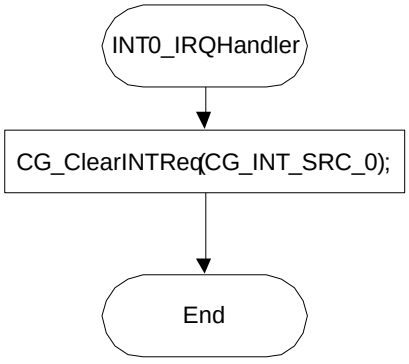
TX00 ペリフェラル・ドライバ(CG)を用いたサンプルプログラムです。

以下の例が含まれます:

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

- **フローチャート**





- **サンプルプログラムのコードと説明:**

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

(リセット後)CG の通常設定:

以下は NORMAL モードで CG の設定を行うプログラムです。 高速発振器は 16MHz を想定しています。

```
/* Switch over from IHOSC to EHOSC*/
switchFromIHOSCtoEHOSC();

/* 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 stop */
CG_SetSTBYMode(CG_STBY_MODE_STOP);
/* Set pin status in stop mode to "active" */
CG_SetPinStateInStopMode(ENABLE);
```

STOP モード設定:

NOMAL モードから STOP モードへ移行します。
NOMAL モードから STOP モードへ移行するには SW0 を ON します。

```
/* Set CG module: Normal ->Stop mode */
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC2, CG_WUODR_EXT);
/* Enter stop mode */
__WFI();
```

NOMAL モードへ移行する準備を行います。INT0 以外を無効化します。

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

低消費電力モードの解除要因として INT0 を使用します。STOP モードから NORMAL モードへ移行するには SW1 を ON します。

```
void INT0_IRQHandler(void)
{
    CG_ClearINTReq(CG_INT_SRC_0);
}
```

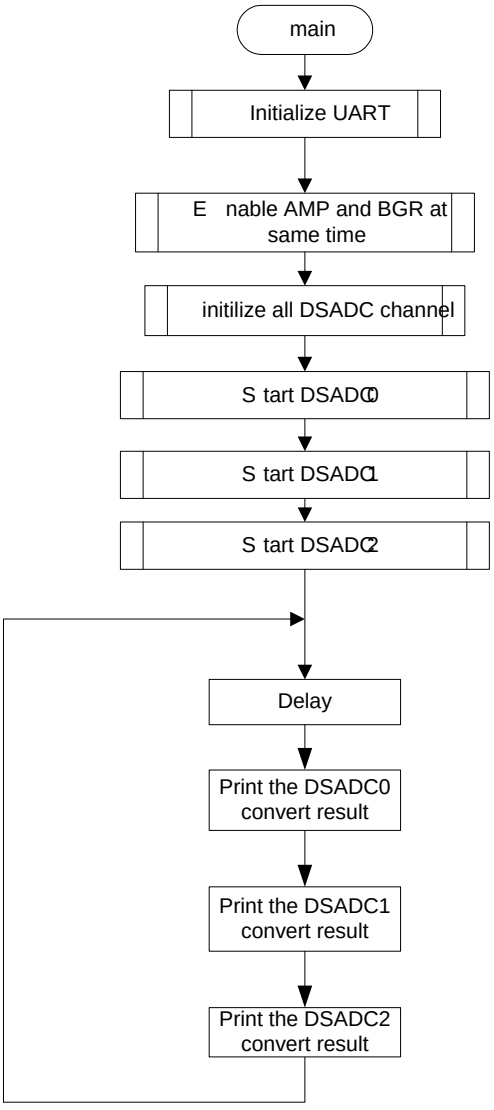
7-3 DSADC

7-3-1 例: DSADC Data Read

この例では以下を行います。

1. DSAD 変換用ソフトウェアトリガを設定します。
2. DSAD 変換状態の取得と変換結果の読み出しを行います。

- **フローチャート:**



- **サンプルプログラムのコードと説明:**

まず、UART を初期化します。

```
Retarget_Init();
```

次に AMP と BGR を許可します。

```
TSB_TEMP->EN = 0x03U;
while(f1ms != 1U){    }; /* delay some time about 1ms*/
```

DSADC の全てのチャネルを初期化します。

```
for(i = 0U; i < (sizeof(t_DSADx)>>2U); ++ i){            /*initilize all DSADC channel*/
        DSADCx = t_DSADx[i];
        DSADC_SWReset(DSADCx);
        DSADC_SetClkSupply(DSADCx,ENABLE);      /*Clock feed to DSADC*/

        InitStruct.Clk = DSADC_FC_DIVIDE_LEVEL_1;
        InitStruct.BiasEn = ENABLE;
        InitStruct.ModulatorEn = ENABLE;
        InitStruct.SyncMode=DSADC_SYNC_MODE;

        InitStruct.Repeatmode = DSADC_REPEAT_MODE;    /*Repeat mode*/
        InitStruct.Amplifier = DSADC_GAIN_8x;
        InitStruct.Offset = 0x03U;
        InitStruct.CorrectEn = ENABLE;
        DSADC_Init(DSADCx,&InitStruct);
    }
```

変換結果を取得し、UART 経由で変換値を出力します。

```
while(1U){
    if(f1s == 1U){                  /*every 1s*/
        f1s = 0u;
        result = DSADC_GetConvertResult(TSB_DSAD0);
        printf("DSAD0: %d\r\n",result);

        result = DSADC_GetConvertResult(TSB_DSAD1);
        printf("DSAD1: %d\r\n",result);
        Delay(1000);
        result = DSADC_GetConvertResult(TSB_DSAD2);
        printf("DSAD2: %d\r\n",result);
    }
```

補足 1: 端子接続:

1. 入力電圧
   (a) pin84(DAIN0+)/pin85(DAIN0-)と電圧(-0.375V<入力電圧<+0.375V)を接続
   (b) pin88(DAIN1+)/pin89(DAIN1-)と電圧(-0.375V<入力電圧<+0.375V)を接続
   (c) pin92(DAIN0+)/pin93(DAIN0-)と電圧(-0.375V<入力電圧<+0.375V)を接続
2. pin83(AGNDREF0), pin87(AGNDREF1), pin91(AGNDREF2)を GND に接続します。
3. pin86(VREFIN0)と 1uF コンデンサが接続された pin'+'を接続し、コンデンサが接続された pin'-'と pin83(AGNDREF0)を接続します。
4. pin 90(VREFIN1)と 1uF コンデンサが接続された pin'+'を接続し、コンデンサが接続された pin'-'と pin87(AGNDREF1)を接続します。
5. pin94(VREFIN2)と 1uF コンデンサが接続された pin'+'を接続し、コンデンサが接続された

'-'と pin91(AGNDREF2)を接続します。

6. pin96(DSRVDD3)と pi 97(SRVDD)を Vcc と接続します。
7. pin95(DSRVSS)を GND と接続します。

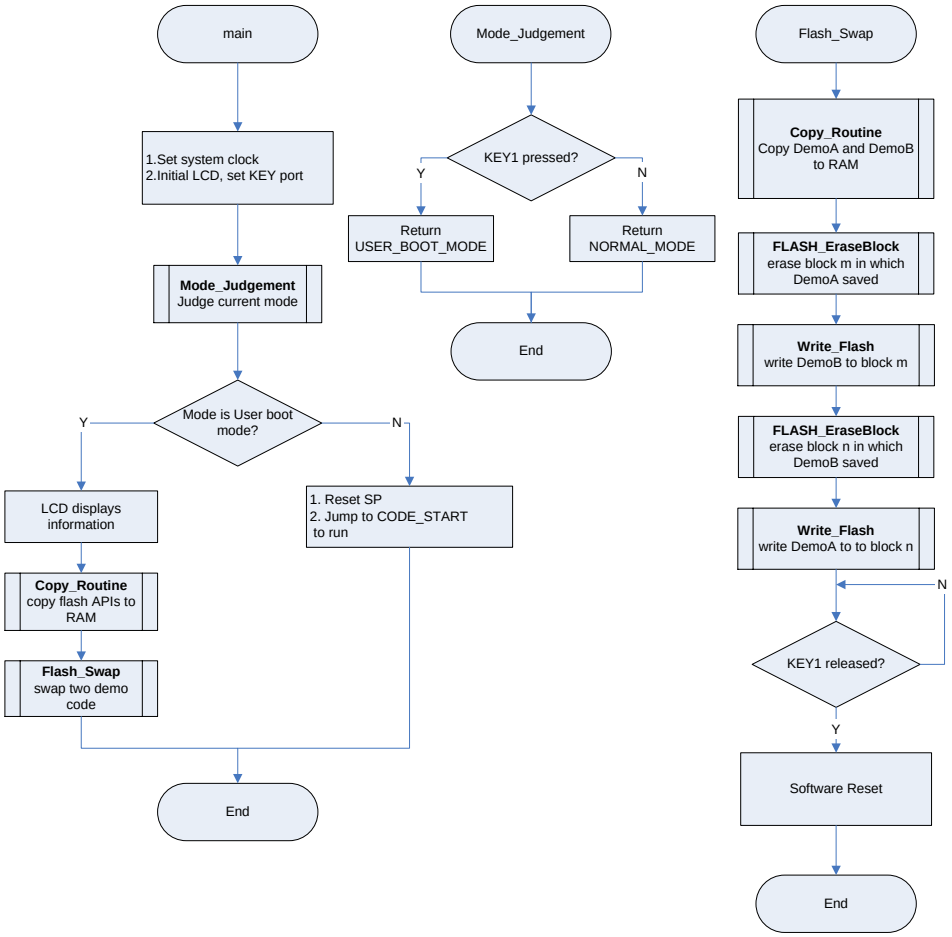
7-4 FLASH

TX00 ペリフェラルドライバ(FLASH)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. FLASH メモリのオンボードプログラミング (書き込み/消去)
2. 動作モード:シングルチップモード (ノーマルモード、ユーザーブートモード)
3. ユーザーブートモードを使用し、Flash メモリのコードを更新。

- **フローチャート**



- **サンプルプログラムのコードと説明:**

まず SW0 と LCD を初期化します。リセット時に現在のモード判定を SW0(GPIO) で行い、LCD に現在の動作モード情報を出力します。

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

```
    LCD_Configuration();
```

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 (SW_Get(SW0) == 1U) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

リセット時に SW0 が OFF の場合、プログラムは NORMAL 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” にジャンプします。ブロックmに格納されている“CODE_START”が開始アドレスのデモ A が動作します (LED0 が点滅)。

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

リセット時に SW0 が ON の場合、プログラムはユーザブートモードになります。LCD にモード情報を表示します。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、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。

```
            LCD_Display("User Boot Mode", "");          /* enter user boot mode */
            LCD_Display("RAM transferring", "......");

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

            LCD_Display("Flash swapping", "......");
```

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,まずデモ A、デモ 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。デモ A とデモ 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 */

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

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

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

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

デモ A とデモ B のスワップ動作が終了したことを示す情報を LCD に表示します。SW0 が OFF の場合、SCB->AIRCR レジスタにて自動的にソフトウェアリセットを行います。そして、再び NORMAL モードで動作します。デモ A とデモ B が入れ替わったため、アドレス “CODE_START”はデモ B の開始アドレスとなり、デモ B が実行されます。(LED1 が点滅します)

```
    LCD_Display("Finished", "Restart.....");


    while (SW_Get(SW0) == 1U) {
    }

    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 “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。それ以外の場合は、自動的ブロック削除命令を送り、ブロックを削除します。

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

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

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

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

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

7-5 LCD

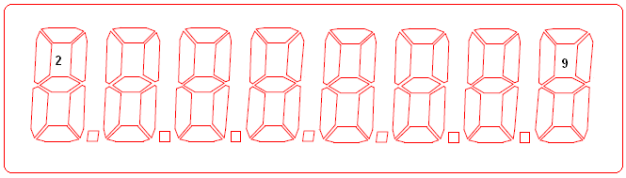
7-5-1 例: LCD Demo1

TX00 ペリフェラルドライバ(LCD)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. LCD の設定と初期化を行います。
2. LCD と同じ桁を表示します。

LCD の詳細情報は以下です。



PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	0
COM1	S15	2D	T1	3D	T2	4D	S1									COM1	/	/	/	
COM2	2E	2C	3E	3C	4E	4C	S2									/	COM2	/	/	
COM3	2G	2B	3G	3B	4G	4B	S3									/	/	COM3	/	
COM4	2F	2A	3F	3A	4F	4A	S4									/	/	/	COM4	

PIN	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
COM1			9D	P7	8D	P6	7D	P5	6D	P4				5D	T3		
COM2			9C	9E	8C	8E	7C	7E	6C	6E				5C	5E		
COM3			9B	9G	8B	8G	7B	7G	6B	6G				5B	5G		
COM4			9A	9F	8A	8F	7A	7F	6A	6F				5A	5F		

接続表:

LCD pin 36 ----> MCU Seg0　　LCD pin 35 ----> MCU Seg1

LCD pin 34 ----> MCU Seg2　　LCD pin 33 ----> MCU Seg3

....

LCD pin 19 to 16　---> MCU Com3 to Com0　…

LCD pin 15 ----> MCU Seg17　　LCD pin 14 ----> MCU Seg18

LCD pin 13 ----> MCU Seg19　　LCD pin 12 ----> MCU Seg20

...

LCD pin 02 ----> MCU Seg30　　LCD pin 01 ----> MCU Seg31

上記の情報によって、以下のフォントテーブルができます。

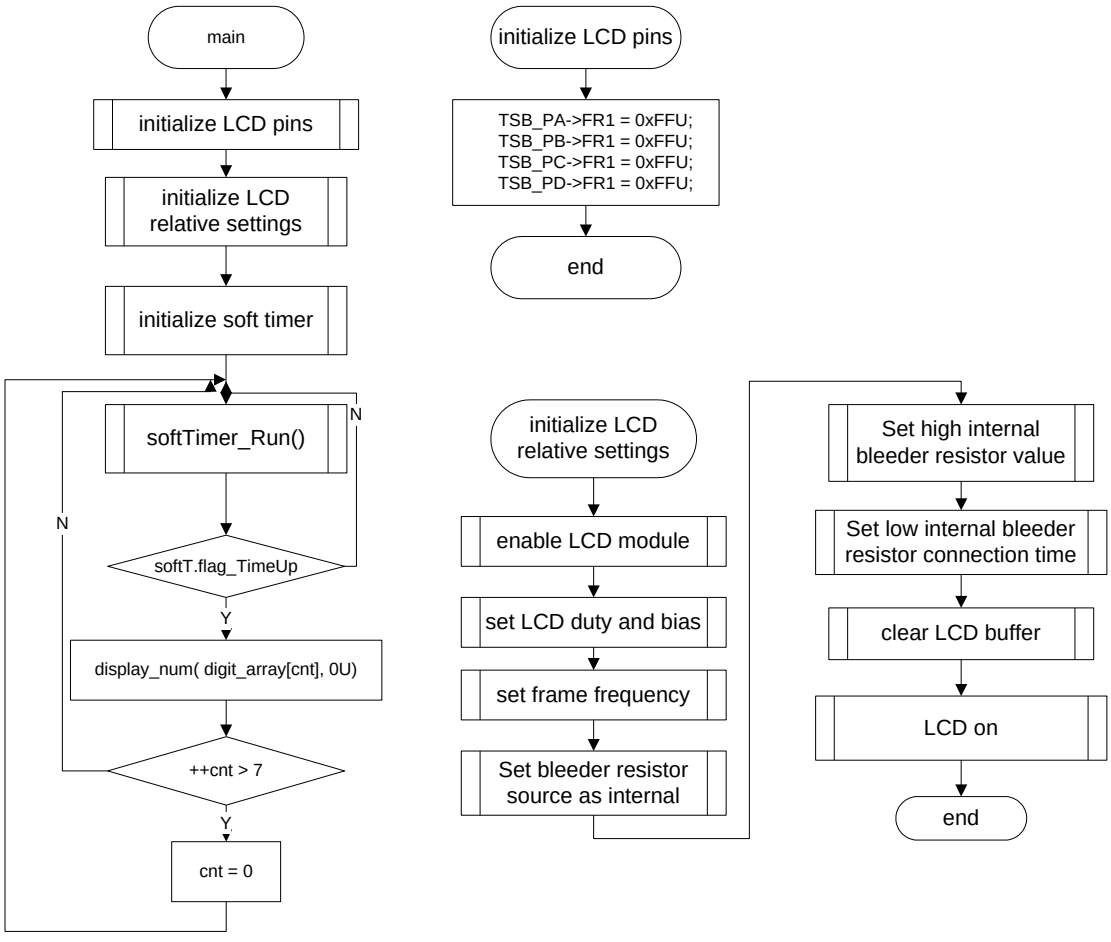
```
const uint8_t number_font_table[] = {
      0xAF,     /* digit '0' */
      0x06,     /* digit '1' */
      0x6D,     /* digit '2' */
      0x4F,     /* digit '3' */
      0xC6,     /* digit '4' */
      0xCB,     /* digit '5' */
```

```
    0xEB,    /* digit '6' */
    0x0E,    /* digit '7' */
    0xEF,    /* digit '8' */
    0xCF,    /* digit '9' */
    0x00     /* ' ' (SPACE) */
};
```

フォントデータと LCD バッファの関係は下表です。

Drive method	Bits 7/3	Bits 6/2	Bits 5/1	Bits 4/0
1/4 duty	COM3	COM2	COM1	COM0
1/3 duty	-	COM2	COM1	COM0
1/2 duty	-	-	COM1	COM0
Static	-	-	-	COM0

- **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、lcd_port_init()関数をコールし、ポートを LCD に設定します。

```
void lcd_port_init(void)
{
    /* Port init to SEG and COM */
    TSB_PA->FR1 = 0xFFU;
    TSB_PB->FR1 = 0xFFU;
    TSB_PC->FR1 = 0xFFU;
    TSB_PD->FR1 = 0xFFU;
}
```

その後、lcd_configure_init()関数をコールし、LCD を初期化します。

```
    /* enable LCD module */
    LCD_Enable();

    /* set LCD duty and bias */
    LCD_SetDutyBias(LCD_DUTY4_BIAS3);              /* 1/4 duty, 1/3 bias */

    /* set frame frequency */
    LCD_SetBaseFreq(LCD_FS_DIVIDE_2_POWER_8); /* F_base = fs / (2^8) */

    /* Set bleeder resistor source as internal */
    LCD_SetBleederSource(LCD_BLEEDER_RESISTOR_INTERNAL);

    /* Set high internal bleeder resistor value */
    LCD_SetInternalBleeder(LCD_BLEEDER_RESISTOR_500K);

    /* Set low internal bleeder resistor connection time */
    LCD_SetLowBleederTime(LCD_2_POWER_6_DEVIDE_BASE_FREQ);

    /* clear LCD buffer */
    lcd_clear();

    /* LCD on */
    LCD_SetDisplay(ENABLE);
```

その後、ソフトウェアタイマを初期化します。

```
softTimer_init();
```

最後に、while(1)によるループ処理を行い、その中でソフトウェアタイマを開始し、タイマが上限に達すると display_num()関数をコールして、文字列を表示します。

```
    while (1U) {

        softTimer_Run();
        if(softT.flag_TimeUp) {
            softT.flag_TimeUp = 0;

            display_num( digit_array[cnt], 0U);
            if( ++cnt > 7 ) {
                cnt = 0;
            } else {
                /* Do nothing */
            }
        } else {
            /* Do nothing */
        }
```

```
        }
```

display_num()関数の中で、表示データに制限を掛け、最上位の 0 を表示しないようにしています。その後、set_digit()関数をコールし、LCD バッファにフォントデータを転送します。
display_num()関数の詳細は、lcd061.c を参照してください。

set_digit()関数のコードは以下です。

```
    void set_digit(uint8_t digit_loc, uint8_t digit, uint8_t dot)
    {
        LCD_font8 font = { 0U };
        LCD_BufIndex idx = LCD_BUF_SEG14;

        if( digit > SPACE ) {
            digit = SPACE;
        } else {
            /* Do nothing */
        }
        font.All = number_font_table[digit];
        if( dot ){
           font.All |= 0x10U;
        } else {
           /* Do nothing */
        }

        if( digit_loc > MAX_DIGITS - 1U ) {
            digit_loc = MAX_DIGITS - 1U;
        } else {
            /* Do nothing */
        }

        if( digit_loc <= 3U ){
            idx -= (LCD_BufIndex)(digit_loc * 2 );
            LCD_WriteBuf( idx, font.Bit.low);
            LCD_WriteBuf( (LCD_BufIndex)(idx - 1U) , font.Bit.high);
        } else if ( digit_loc == 4U ) {
            LCD_WriteBuf( LCD_BUF_SEG03, font.Bit.low);
            LCD_WriteBuf( LCD_BUF_SEG02, font.Bit.high);
        } else {
            idx = (LCD_BufIndex)(LCD_BUF_SEG2726 + (LCD_BufIndex)(digit_loc -
5U));
            LCD_WriteBuf(idx, font.All);
        }

    }
```

LCD の各 8 桁の接続表が同じ順番にならないよう、LCD_WriteBuf()関数をコールし、セグメントデータ領域にフォントデータをライトします。

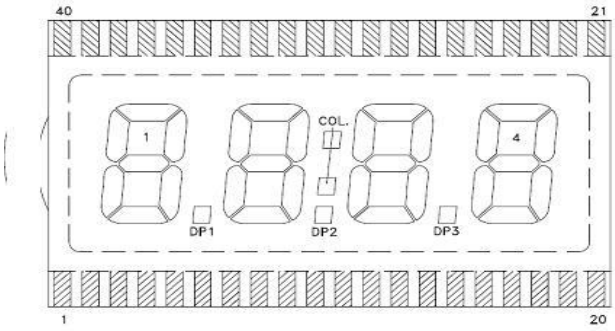
7-5-2 例: M061-SK LCD

TX00 ペリフェラルドライバ(LCD)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。

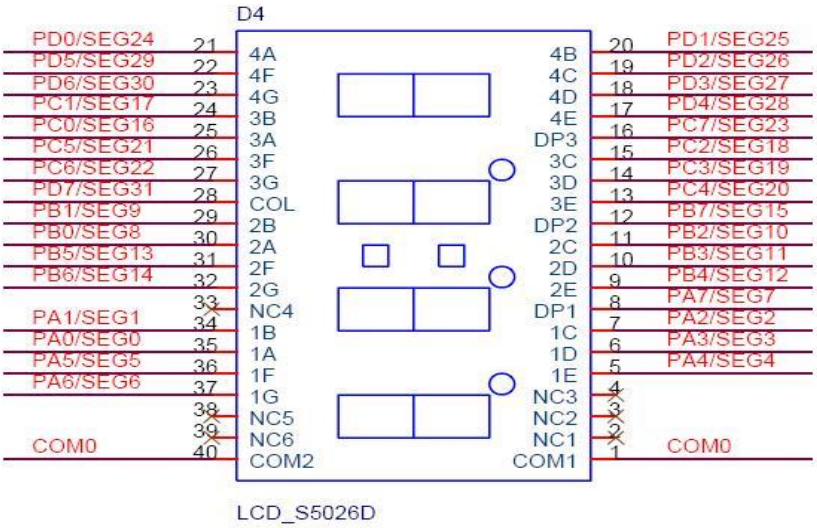
1. LCD の設定と初期化を行います。
2. LCD と同じ桁を表示します。

LCD の詳細情報は以下です。



PIN	1	2	3	4	5	6	7	8	9	10
COM	COM	/	/	/	1E	1D	1C	DP1	2E	2D
PIN	11	12	13	14	15	16	17	18	19	20
COM	2C	DP2	3E	3D	3C	DP3	4E	4D	4C	4B
PIN	21	22	23	24	25	26	27	28	29	30
COM	4A	4F	4G	3B	3A	3F	3G	COL.	2B	2A
PIN	31	32	33	34	35	36	37	38	39	40
COM	2F	2G	/	1B	1A	1F	1G	/	/	COM

接続表:



上記の情報によって、以下のフォントテーブルができます。

```
const uint8_t number_font_table[] = {
    0x3F,   /* digit '0' */
    0x06,   /* digit '1' */
    0x5B,   /* digit '2' */
    0x4F,   /* digit '3' */
    0x66,   /* digit '4' */
    0x6D,   /* digit '5' */
    0x7D,   /* digit '6' */
```

```
    0x07,     /* digit '7' */
    0x7F,     /* digit '8' */
    0x6F      /* digit '9' */
};
```

フォントデータと LCD バッファの関係は下表です。

Drive method	Bits 7/3	Bits 6/2	Bits 5/1	Bits 4/0
1/4 duty	COM3	COM2	COM1	COM0
1/3 duty	-	COM2	COM1	COM0
1/2 duty	-	-	COM1	COM0
Static	-	-	-	COM0

- **フローチャート:**
  M061-SK LCD demo のフローチャートは、“display_num(digit_array[cnt], 0U);”関数の代わりに“display_4digit(&str[idx], dot);”関数をコールする点を除き、上記“例: LCD Demo1”と同じため、割愛します。

- **サンプルプログラムのコードと説明:**

まず、lcd_port_init()関数をコールし、ポートを LCD に設定します。

```
void lcd_port_init(void)
{
    /* Port init to SEG and COM */
    TSB_PA->FR1 = 0xFFU;
    TSB_PB->FR1 = 0xFFU;
    TSB_PC->FR1 = 0xFFU;
    TSB_PD->FR1 = 0xFFU;
}
```

その後、lCD_Init()関数をコールし、LCD を初期化します。

```
/* enable low-speed oscillator for LCD */
if(CG_GetFsState() == DISABLE)
{
    CG_SetFs(ENABLE);
}
lcd_port_init();
lcd_configure_init();
```

その後、lCD_Init()関数をコールし、LCD を初期化します。

```
/* enable LCD module */
LCD_Enable();

/* set LCD duty and bias */
LCD_SetDutyBias(LCD_STATIC);          /* static LCD on M061-SK board */

/* set frame frequency */
LCD_SetBaseFreq(LCD_FS_DIVIDE_2_POWER_8);   /* F_base = fs / (2^8) */
```

```
/* Set bleeder resistor source as internal */
LCD_SetBleederSource(LCD_BLEEDER_RESISTOR_INTERNAL);

/* Set high internal bleeder resistor value */
LCD_SetInternalBleeder(LCD_BLEEDER_RESISTOR_500K);

/* Set low internal bleeder resistor connection time */
LCD_SetLowBleederTime(LCD_2_POWER_6_DEVIDE_BASE_FREQ);

/* clear LCD buffer */
lcd_clear();

/* LCD on */
LCD_SetDisplay(ENABLE);
```

その後、ソフトウェアタイマを初期化します。

```
softTimer_init();
```

最後に、while(1)によるループ処理を行い、その中でソフトウェアタイマを開始し、タイマが上限に達すると display_num()関数をコールして、文字列を表示します。

```
while (1U) {
    softTimer_Run();
    if(softT.flag_TimeUp) {
        softT.flag_TimeUp = 0;

        if(idx > 3U && idx < 7U){
            dot = 7U - idx;
        } else {
            dot = 0U;
        }
        display_4digit( &str[idx], dot );

        idx++;
        if( idx > 9U ) {
            idx = 0U;
        } else {
            /* Do nothing */
        }
    } else {
        /* Do nothing */
    }
}
```

スタティックLCD のため、LCD バッファの bit0 のみ、つまりフォントの 8bit データを 8 つの LCD バッファレジスタの bit0 に設定するため、display_4digit()関数のコードを以下に示します。

```
void display_4digit(unsigned char *p, uint8_t dot_pos)
{
    uint8_t tmp = 0U, loop = 0U;
    uint8_t idx = 0U, dat = 0U;

    LCD_BufIndex buf_head = LCD_BUF_SEG0100;

    for( ; loop < 4U; loop++) { /* total 4 digit need to display */
        tmp = *(p + loop);
        if( tmp == ' ' ) {          /* is SPACE, hide it */
```

```
                tmp = 0U;
            } else {
                tmp = number_font_table [ tmp - '0' ];
                if( dot_pos == loop + 1U) { /* if dot_pos isn't '0', should display dot */
                    tmp |= 0x80U;
                } else {
                    /* Do nothing */
                }
            }

    /* now 'tmp' store the font data, we will set its 8 bit info to 8 LCD Segment buffer */
            for(idx = 0U; idx < 4U; idx++) {
                dat = 0U;
                if( tmp & 0x01U ) {
                    dat |= 0x01U;
                } else {
                    /* Do nothing */
                }
                if( tmp & 0x02U ) {
                    dat |= 0x10U;
                } else {
                    /* Do nothing */
                }
                /* set 2 segment at one time */
                LCD_WriteBuf( (LCD_BufIndex)(buf_head + idx) , dat );
                tmp >>= 2U;
            }
            buf_head += 4U;       /* for the static LCD, one digit '8' need 8 seg, */
                /*    but we process 2 seg at one time, so, only need to add 4 here */
        }
    }
```

7-6 LVD

7-6-1 LVD Demo

TX03 ペリフェラルドライバ(LVD)を用いて電圧の状態を確認するサンプルプログラムです。

電圧が通常の場合は LED0 を点滅させ、LED1 を OFF します。
電圧が検出電圧より低い場合は INTLVD を発生させ、LVD 割り込みハンドラ内で LED0 を OFF し、LED1 を点滅させます。

検証ステップ

1. 3.1V より大きい電圧に設定します。LED0 が点滅します。
2. 2.9V より小さい電圧に設定します。LED0 が OFF し、LED1 が点滅します。
3. 3.1V に戻します。LED0 が点滅し、LED1 が OFF します。

補足: 電圧は M061 の電源端子に接続されているため、留意してください。
以下のように接続します。
電源電圧 '+' と M061 の pin10 または 74 と接続します。
電源電圧 '-' と M061 の pin8 または 75 と接続します。

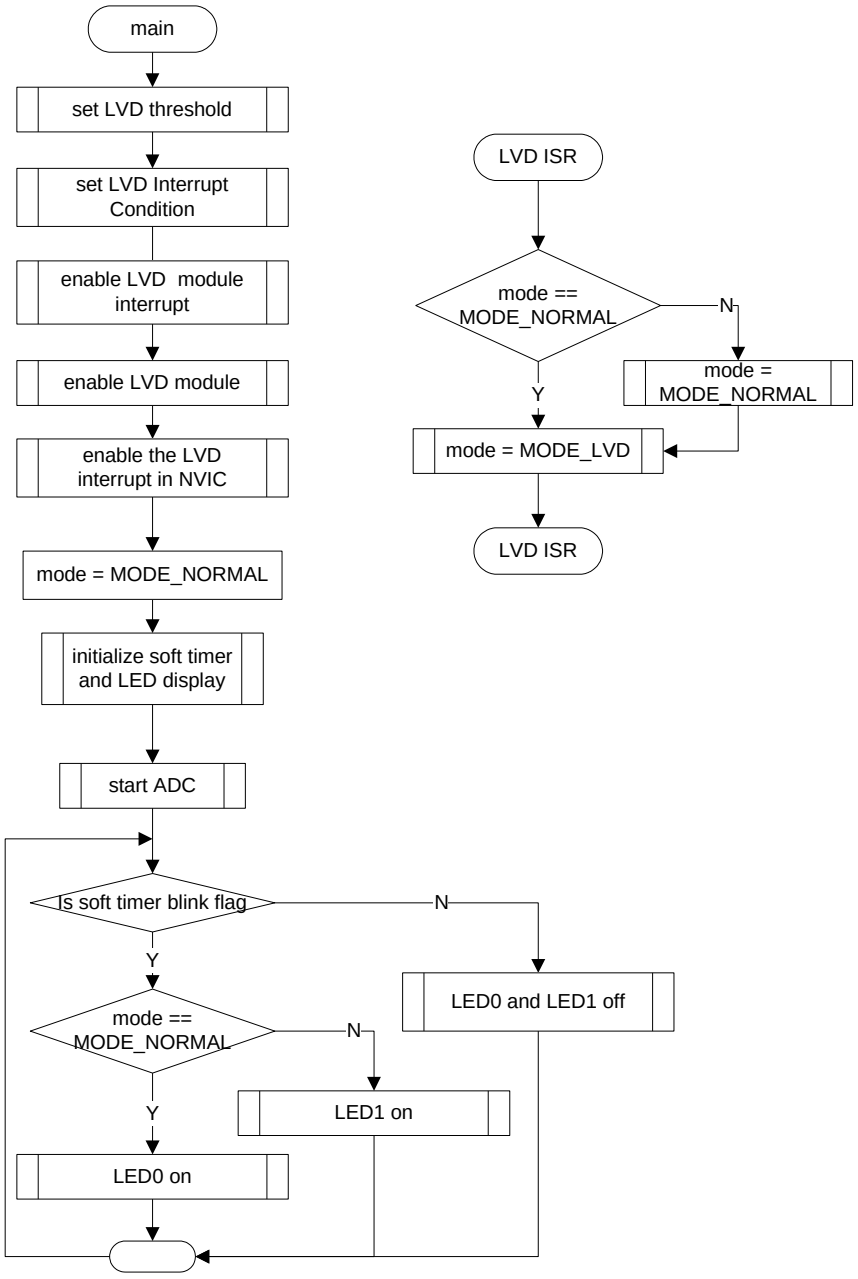
7-6-2 例: LVD Demo

TX00 ペリフェラルドライバ(LVD)を用いた簡単な例です。

この例では以下を行います。

1. LVD の設定と初期化を行います。
2. LVD 割込み

- **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、LVD の閾値を 3.00 +/- 0.2V に設定し、LVD 割込み状態を設定します。

```
/* 1. set LVD threshold as 3.00 +/- 0.2V */
LVD_SetVoltage(LVD_DETECT_VOLTAGE_300);

/* 2. set LVD Interrupt Condition: both side */
LVD_SetINTCondition(LVD_INTSEL_LOWER_UPPER);
```

その後、LVD 割込みと LVD 動作を許可します。

```
/* 3. enable LVD module interrupt */
LVD_SetINTOutput(ENABLE);

/* 4. enable LVD module */
LVD_Enable();
```

NVIC の LVD 割込みを許可します。

```
NVIC_EnableIRQ(INTLVD_IRQn);
```

その後、LED 表示の初期化とソフトウェアタイマの設定を行います。

```
/* when power is up, mode is 'normal' */
mode = MODE_NORMAL;

/* initialize LEDs on board before display something */
LED_Init();

/* initialize soft timer */
softTimer_Init();
```

“printf()”とソフトウェアタイマの初期化を行います。

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

/* initialize soft timer */
softTimer_Init();
```

main ループの中で、点滅フラグをセットすると、LED0 または LED1 が現状のモードにあわせて変化します。

```
while (1U) {
    softTimer_Run();
    if(softT.flag_Blink) {
        if( mode == MODE_NORMAL ) {
            LED_On(LED0);
        } else if ( mode == MODE_LVD ) {
            LED_On(LED1);
        } else {
            /* Do nothing */
        }
    } else {
        LED_Off(LED0);
        LED_Off(LED1);
    }

}
```

以下は LVD 割込みハンドラです。電圧が通常状態から 2.9V に低下(上記 1)すると、LVD 割込みが発生します。

```
void INTLVD_IRQHandler(void)
{
    /* the first time enter this routine, power supply is lower than threshold */
    if( mode == MODE_NORMAL ) {
        mode = MODE_LVD;
    } else if ( mode == MODE_LVD ) { /* the second time enter thsi routine, power
supply is rising and higher than threshold */
        mode = MODE_NORMAL;
    } else {
        /* Do nothing */
    }
}
```

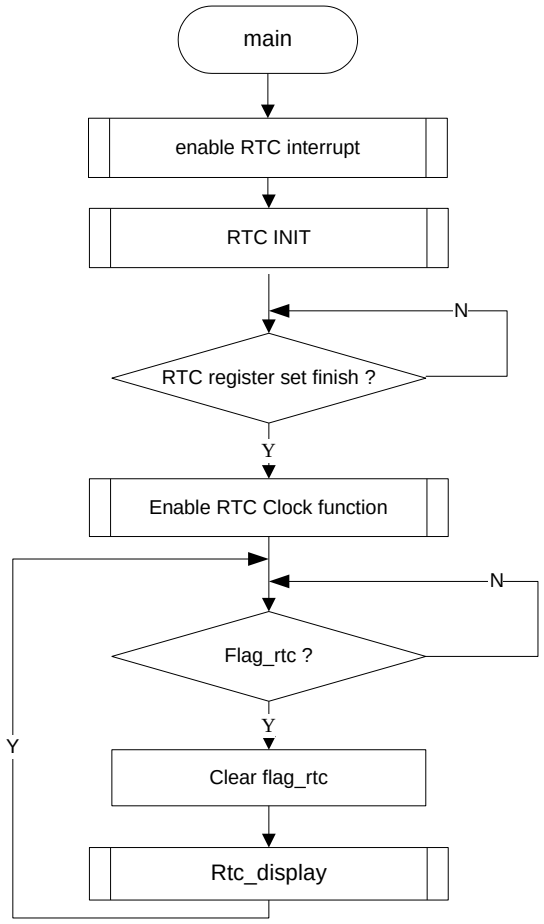
7-7 RTC

TX00 ペリフェラルドライバ (RTC, CG) のサンプルプログラムです。

この例は以下を行います。

1. RTC 初期化
2. RTC データと時間の取得

- **フローチャート:**



- **サンプルプログラムのコードと説明:**

まず、RTC の初期化で、RTC_DateTypeDef と RTC_TimeTypeDef 構造を作成し、全データ項目を設定します。ここでは 2010/10/22 12:50:55, 24 時間表示フォーマットを初期設定としています。

```
Date_Struct.LeapYear = RTC_LEAP_YEAR_3;
Date_Struct.Year = (uint8_t) 10U;
Date_Struct.Month = (uint8_t) 10U;
Date_Struct.Date = (uint8_t) 22U;
Date_Struct.Day = RTC_FRI;
```

```
Time_Struct.HourMode = RTC_24_HOUR_MODE;
Time_Struct.Hour = (uint8_t) 12U;
Time_Struct.Min = (uint8_t) 50U;
Time_Struct.Sec = (uint8_t) 55U;
```

クロックとアラーム機能を禁止します。

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

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

```
RTC_ResetClockSec();
RTC_SetAlarmOutput(RTC_PULSE_1_HZ);
RTC_SetRTCINT(ENABLE);
```

RTC の時間と日付の値を設定します。

```
RTC_SetTimeValue(&Time_Struct);
RTC_SetDateValue(&Date_Struct);
```

上記項目を設定後、RTC 割り込みを許可します。 RTC レジスタ設定の終了を待ち、RTC 時計機能を許可します。

```
NVIC_EnableIRQ(INTRTC_IRQn);
RTC_EnableClock();
```

RTC 割り込みにおいて、RTC 割り込みを秒単位で発生するようにします。その後、RTC 割り込み要求をクリアします。

```
fRTC_1HZ_INT = 1U;
CG_ClearINTReq(CG_INT_SRC_RTC);
```

割り込みが発生すると、RTC データと時間値を LCD に転送します。
以下は、RTC データと時間値をどのように取得するかを示します。

```
Year = RTC_GetYear();
Month = RTC_GetMonth();
Date = RTC_GetDate(RTC_CLOCK_MODE);
Hour = RTC_GetHour(RTC_CLOCK_MODE);
Min = RTC_GetMin(RTC_CLOCK_MODE);
Sec = RTC_GetSec();
```

7-8 SBI

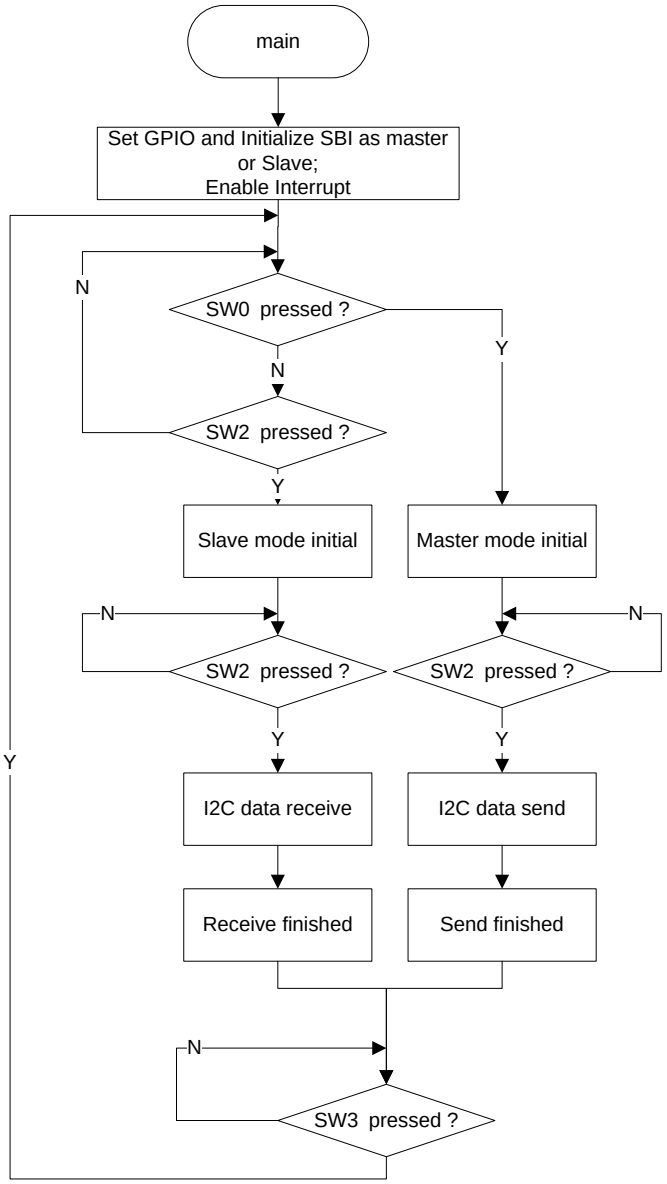
7-8-1 例: SBI Slave

TX00 ペリフェラルドライバ (SBI) のプログラムサンプルです。

この例は以下を行います。

1. SBI を設定します。
2. I2C マスタによるデータ送信を行います。
3. I2C スレーブによるデータ受信を行います。

- **フローチャート**



- **サンプルプログラムのコードと説明:**

まず、GPIO を SBI I2C に設定します。

```
TSB_PJ->FR1 |= 0x03U;
TSB_PJ->CR |= 0x03U;
TSB_PJ->IE |= 0x03U;
TSB_PJ->OD |= 0x03U;
TSB_PJ->PUP |= 0x03U;
```

SBI 動作を許可し、マスタ/スレーブの設定、および INTI2C を許可します。

```
/* Master mode */
myI2Cm.I2CSelfAddr = SELF_ADDR;
myI2Cm.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2Cm.I2CACKState = ENABLE;
myI2Cm.I2CClkDiv = SBI_I2C_CLK_DIV_328;
/* Slave mode */
myI2Cs.I2CSelfAddr = SLAVE_ADDR;
myI2Cs.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2Cs.I2CACKState = ENABLE;
myI2Cs.I2CClkDiv = SBI_I2C_CLK_DIV_328;

NVIC_EnableIRQ(INTI2C_IRQn);
```

上記設定を行った後、I2C 送信を開始します。
マスタモードの初期化を行い、送信バッファとバッファ長の設定、受信バッファのクリアを行います。

```
case MODE_SBI_MASTER_INITIAL:
        /* Initialize SBI channel to Master mode */
        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';

    SBI_Enable(TSB_SBI);
    SBI_SWReset(TSB_SBI);
    SBI_InitI2C(TSB_SBI, &myI2Cm);
    gSBIMode = MODE_SBI_I2C_START;
      break;
```

スレーブモードの初期化を行い、受信バッファをクリアします。

```
case MODE_SBI_ SLAVE_INITIAL:
       /* Initialize SBI channel to Slave mode */
    gI2CWCnt = 0U;
      for (gICnt = 0U; gICnt < 8U; gICnt++) {
            gI2CRxData[gICnt] = 0U;
      }

    SBI_Enable(TSB_SBI);
    SBI_SWReset(TSB_SBI);
    SBI_InitI2C(TSB_SBI, &myI2Cs);
    gSBIMode = MODE_SBI_I2C_ TRX;
      break;
```

I2C バスが空いているかどうか確認し、SBI_SetSendData()関数を使って“SLAVE_ADDR”データの設定、および SBI データバッファの送信方向“SBI_I2C_SEND”の設定を行います。その後、SBI_GenerateI2CStart(TSB_SBI)を使用して、I2C 動作を開始します。

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

INTI2C 割込みハンドラ内でデータ転送を行います。

INTI2C ハンドラ内で、I2C バス状態を取得し、その値で I2C マスタ送信プロセスを決定します。I2C マスタ送信中は、SBI_SetSendData() で次のデータを送信し、I2C プロセス終了時には、SBI_GenerateI2CStop() で I2C を停止します。

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

INTI2C ハンドラで、I2C バス状態を取得し、その値で I2C スレーブ受信プロセスを決定します。SBI バッファの受信データ読み出しは SBI_GetReceiveData() 関数を用いて行います。I2C 停止状態はマスタによって制御します。

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

7-9 TMR16A

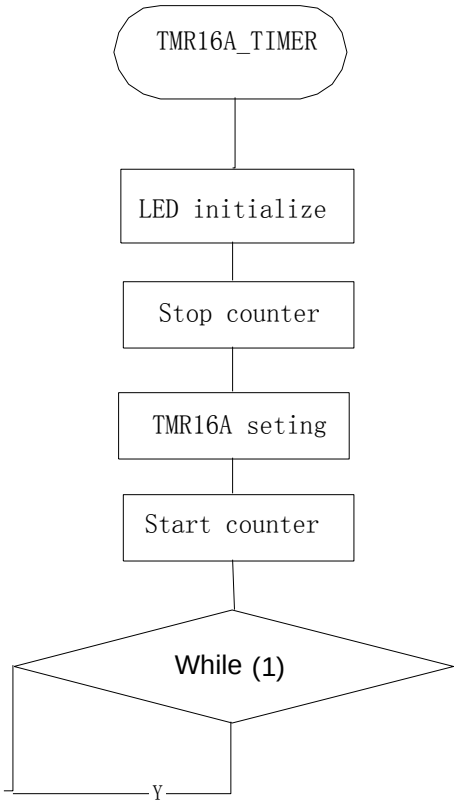
7-9-1 例: 汎用タイマ

TX00 ペリフェラルドライバ (TMR16A) のプログラムサンプルです。

この例は以下を行います。

1. TMR16A 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:**

まず、評価ボード上の LED を初期化し、LED を On します。

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

このサンプルプログラムでは、周期の設定を TMR16A_1MS マクロにて 0x3E80U を定義することで 1ms の周期に設定します。このカウント値は以下により算出されます。

ftmrb = fphiT0 = fsys = fc = 16MHz、Ttmrb = 1/16us, 1ms*16 us = 16000 = 0x3E80 (クロック設定の詳細は CG 章を参照してください。)
その後、カウントを開始します。

```
        TMR16A_SetRunState(TSB_T16A0, TMR16A_STOP);    /* counter stops*/
        TMR16A_SetSrcClk(TSB_T16A0, TMR16A_SYSCK);     /* set source clock is system clock */
        TMR16A_ChangeCycle(TSB_T16A0, TMR16A_1MS);      /* Set a counter value is 1ms*/
        /* Before starting counter operation, set "0x0000" to T16AxCP<CP> to clear the counter*/
        TMR16A_ClrCaptureValue(TSB_T16A0);
        NVIC_EnableIRQ(INTT16A0_IRQn);
        TMR16A_SetRunState(TSB_T16A0, TMR16A_RUN);
```

main 関数の“While(1)”ループの中で、割り込み発生を待ちます。割り込みルーチンの中で、カウントアップし、500ms をカウントすると LED の状態を反転させ、再度カウントアップします。

```
    tbcount++;
    if (tbcount >= 500U) {                              /* 500ms is up */
        tbcount = 0U;
        /* reverse LED output */
        ledon = (ledon == 0U) ? 1U : 0U;
        if (0U == ledon) {
            LedOff(LED11);
        } else {
            LedOn(LED11);
        }
    } else {
        /* do nothing */
    }
```

7-10 TMRB

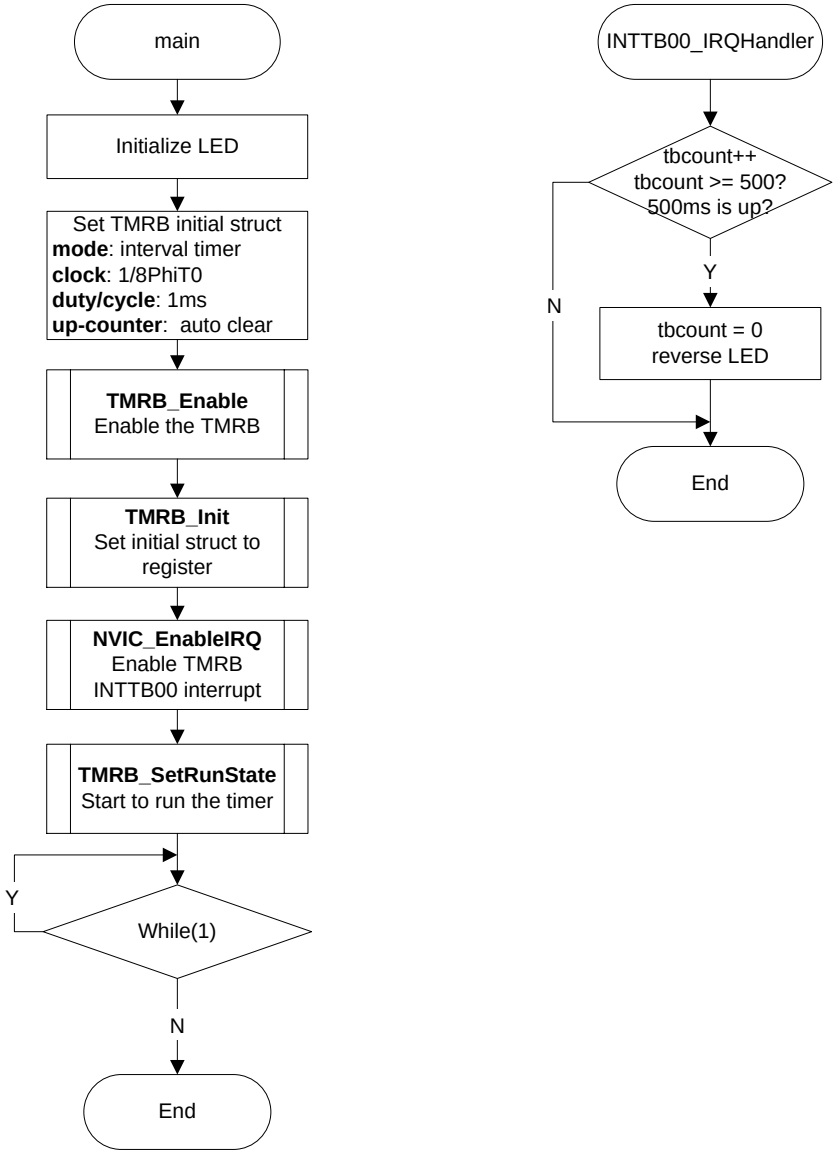
7-10-1 例: 汎用タイマ

TX00 ペリフェラルドライバ (TMRB) のプログラムサンプルです。

この例は以下を行います。

1. TMRB0 の初期化を行います。
2. 1ms の汎用タイマを生成します。

- **フローチャート**



- **サンプルプログラムのコードと説明**

最初に LED を初期化し、LED を On します。

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

TMRB 設定用の構造体を用意し TMRB モード、クロック、アップカウンタクリア方法、周期とデューティを設定します。このデモでは、1ms の周期とデューティを設定します。TMRB_1MS マクロは、0x7d0 です。(φT0=fsys=16MHz, ftmrb = 1/8φT0 = 2MHz, Ttmrb = 0.5us, 1ms/0.5us = 2000 = 0x7d0 (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。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(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(LED11);
    } else {
        LedOn(LED11);
    }
} else {
    /* do nothing */
}
```

7-10-2　例: PPG 出力

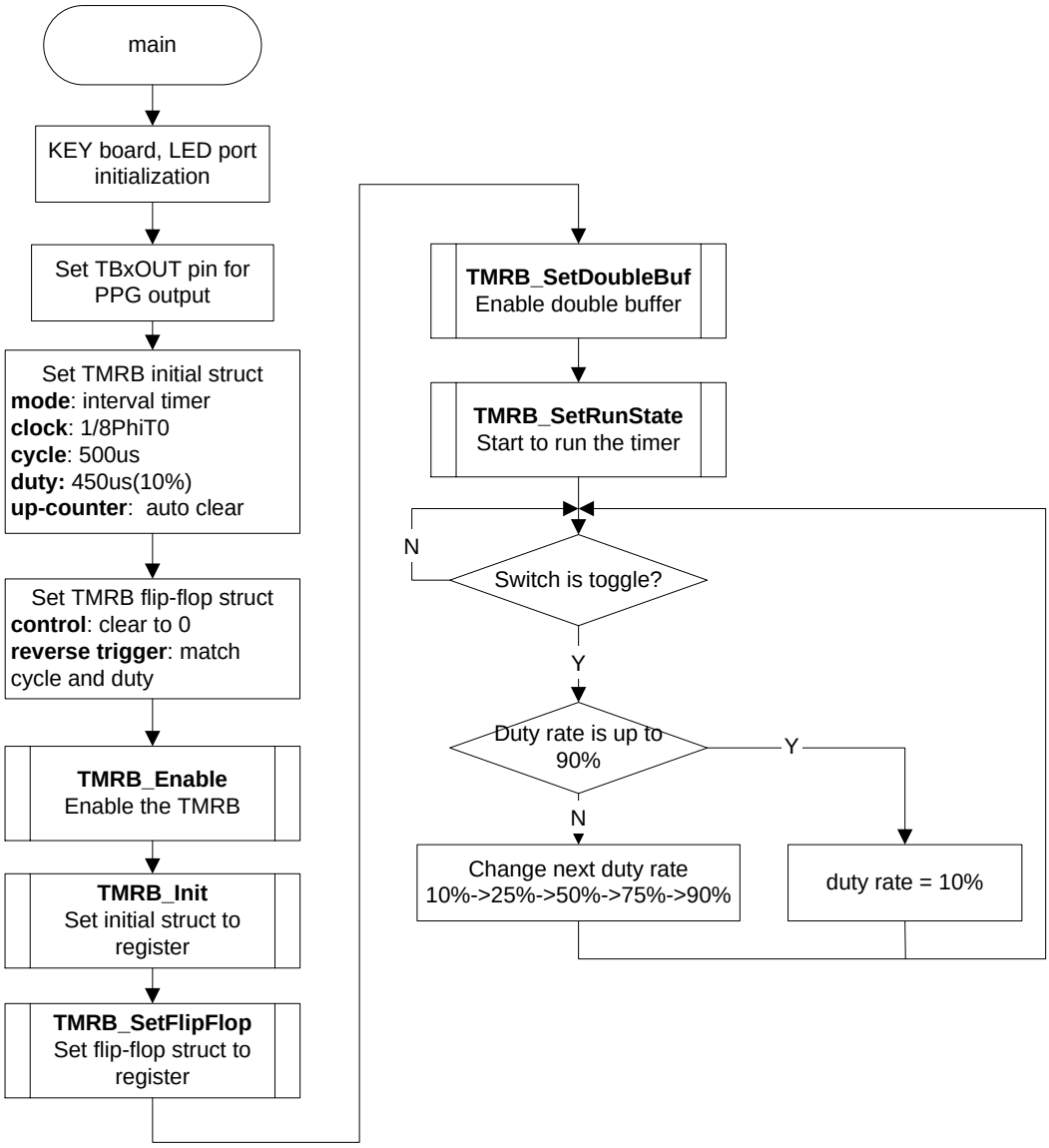
ペリフェラルドライバ (TMRB, GPIO) を使用したサンプルプログラムです。

以下の例が含まれます。

1. TMRB0 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 PPG 出力用に PG0 を TB0OUT に設定します。

```
TSB_PG->CR |= 0x01;
TSB_PG->FR1 |= 0x01;
```

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、TMRB0TIME マクロを定義してあります。このマクロは 0x03e8 です。(φT0 = fsys = fc = 16MHz, ftmrb = 1/8 φT0 = 2MHz, Ttmrb = 0.5us, 500us/0.5us = 1000 = 0x03e8 (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 = TMRB0TIME;        /* trailing timing is 500us */
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
    m_tmrb.LeadingTiming = LeadingTiming[Rate]; /* leading timing, initial value 10% */
```

フリップ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_TB0);
    TMRB_Init(TSB_TB0, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB0, &PPGFFInital);
    /* enable double buffer */
    TMRB_SetDoubleBuf(TSB_TB0,ENABLE, TMRB_WRITE_REG_SEPARATE);
    TMRB_SetRunState(TSB_TB0, TMRB_RUN);
```

スイッチ状態の変化を待ちます。

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

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。

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

    TMRB_ChangeLeadingTiming(TSB_TB0,   LeadingTiming[Rate]);        /*   change
leading timing rate */
```

デューティの算出方法:

Trailing Timing = 500us, ftmrb = 1/8 fphiT0 = 2MHz, Ttmrb = 0.5us (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 = 0x384

LeadingTiming = 25%: High 幅は 500*25% = 125us, Low 幅は 500-125 = 375us, カウンタ値= 375us/Ttmrb = 0x2EEU

LeadingTiming = 50%: High 幅は 500*50% = 250us, Low 幅は 500-250 = 250us, カウンタ値= 250us/Ttmrb = 0x1F4U

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

LeadingTiming = 90%: High 幅は 500*90% = 450us, Low 幅は 500-450 = 50us, カウンタ値 = 50us/Ttmrb = 0x64U

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] = { 0x384U, 0x2EEU, 0x1F4U, 0xFAU, 0x64U };
/* leading timing: 10%, 25%, 50%, 75%, 90% */
```

7-11 SIO/UART

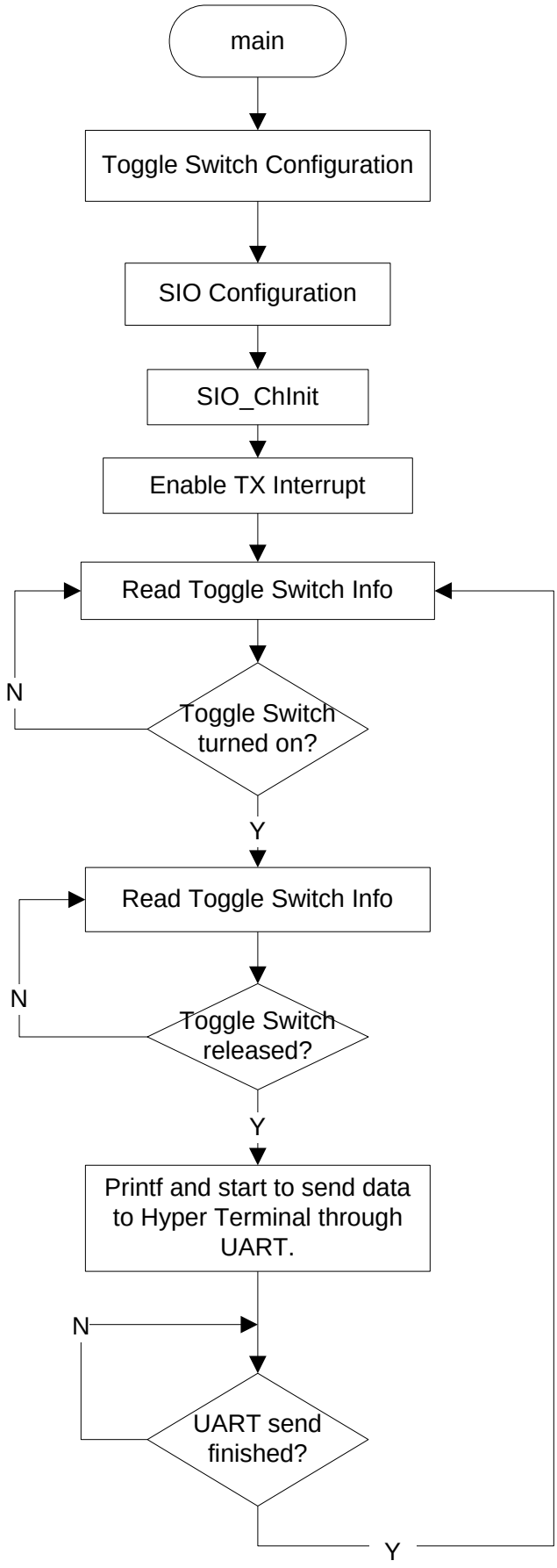
7-11-1 例: リターゲット

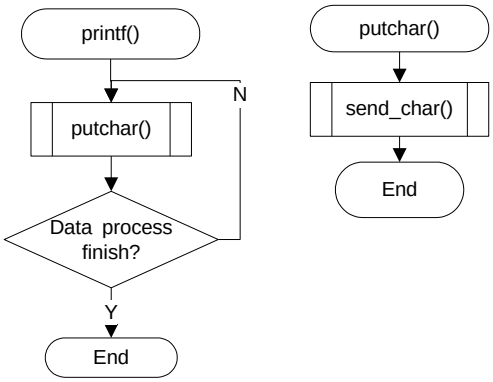
ペリフェラルドライバ (UART)を用いたサンプルプログラムです。

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. UART に printf()関数をリターゲット

- **フローチャート**


main
Toggle Switch Configuration
SIO Configuration
SIO_ChInit
Enable TX Interrupt
Read Toggle Switch Info
N
Toggle Switch turned on?
Y
Read Toggle Switch Info
N
Toggle Switch released?
Y
Printf and start to send data to Hyper Terminal through UART.
N
UART send finished?
Y



- **サンプルプログラムのコードと説明:**

GPIO を UART0 に設定します。

```
TSB_PH->CR |= GPIO_BIT_0;
TSB_PH->FR1 |= GPIO_BIT_0;
TSB_PH->FR1 |= GPIO_BIT_1;
TSB_PH->IE |= GPIO_BIT_1;
```

UART_InitTypeDef 構造体を準備します。データを設定後、UART0 を初期化します。以下は設定例です。

```
UART_InitTypeDef myUART;

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

上記設定を行い、その後、UART0 の送信割り込みを有効にします。

```
NVIC_EnableIRQ(RETARGET_INT)
```

データ送信を行います。UART0 経由で printf()にてデータを送信します。

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

最後に UART0 の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(UART_RETARGET,          gSIOTxBuffer[gSIORdIndex++]); /* send data */
        fSIO_INT = SET;              /* SIO1 INT is enable */
    } else {
        /* disable SIO1 INT */
        fSIO_INT = CLEAR;
        NVIC_DisableIRQ(RETARGET_INT);
```

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

IARコンパイラではprintf()関数がputchar()関数をコールし、RealViewコンパイラではfputc()関数をコールし、UART0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(UART_RETARGET, gSIOTxBuffer[gSIORdIndex++]);
            NVIC_EnableIRQ(RETARGET_INT);
        }

        return ch;
}
```

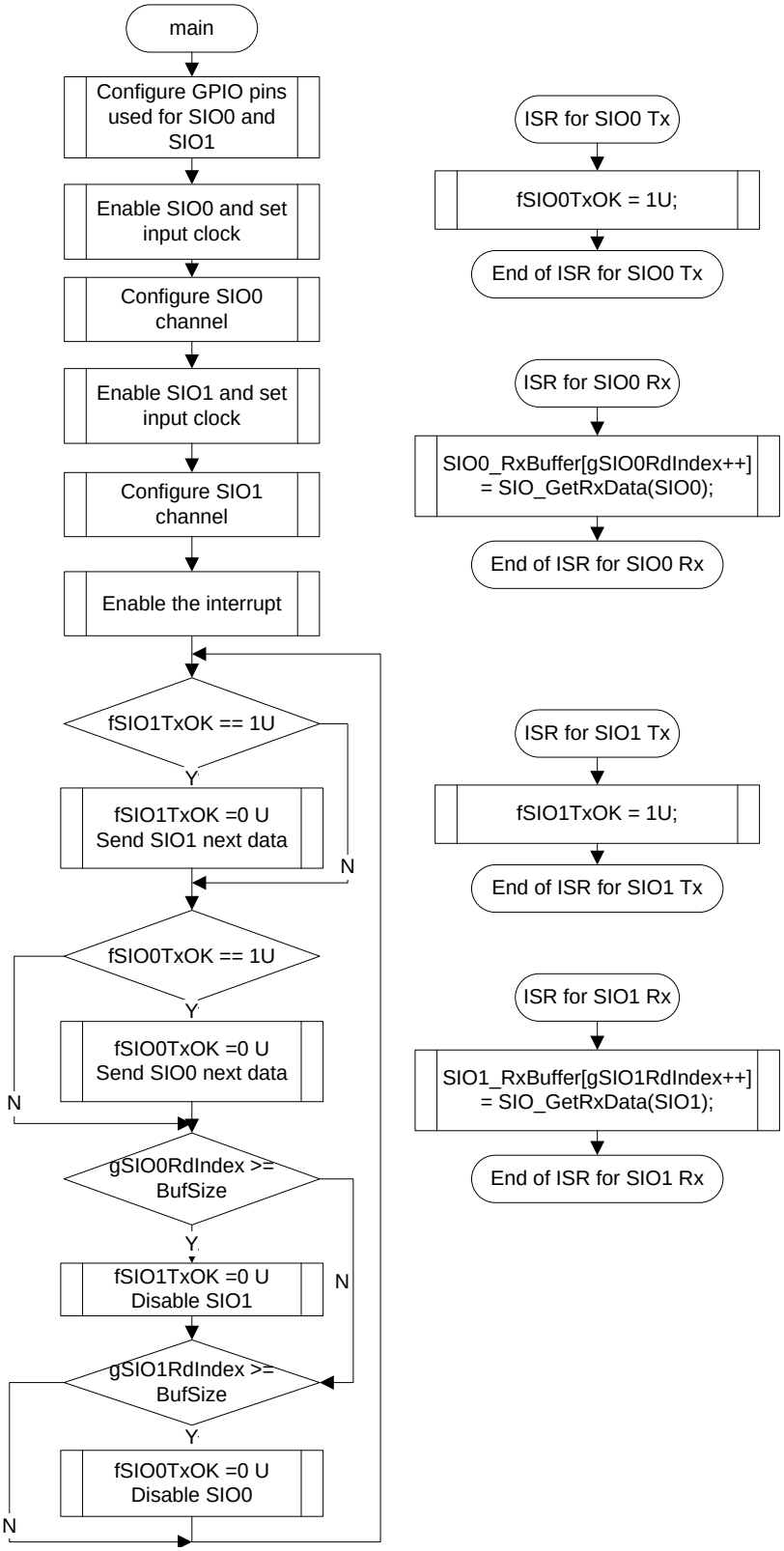
7-11-2　例: SIO

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-SIO3 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);
```

SIO3 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 SIO3 channel */
SIO_Enable(SIO3);

/*initialize the SIO3 struct */
SIO3_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO3_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO3_Init.TransferDir = SIO_LSB_FRIST;
SIO3_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO3_Init.DoubleBuffer = SIO_WBUF_ENABLE;

SIO_Init(SIO3, SIO_CLK_SCLKINPUT, &SIO3_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 SIO3 Channel RX interrupt */
NVIC_EnableIRQ(INTRX3_IRQn);

/* Enable SIO3 Channel TX interrupt */
NVIC_EnableIRQ(INTTX3_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) {
        /* SIO3 send data from TXD3*/
        if (fSIO3TxOK == 1U) {
            fSIO3TxOK = 0U;
            SIO_SetTxData(SIO3, SIO3_TxBuffer[gSIO3WrIndex++]);
        } 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) {
```

```
            fSIO3TxOK = 0U;
            SIO_Disable(SIO3);
        } else {
            /*Do Nothing */
        }
        /*SIO3 receive data end */
        if (gSIO3RdIndex >= 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 SIO3 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 INTTX3_IRQHandler(void)
{
    fSIO3TxOK = 1U;
}
```

以下は SIO3 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 INTRX3_IRQHandler(void)
{
    SIO0_RxBuffer[gSIO3RdIndex++] = SIO_GetRxData(SIO3);
}
```

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)
    {
    }
```

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();
```