TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM330)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M330UE-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。

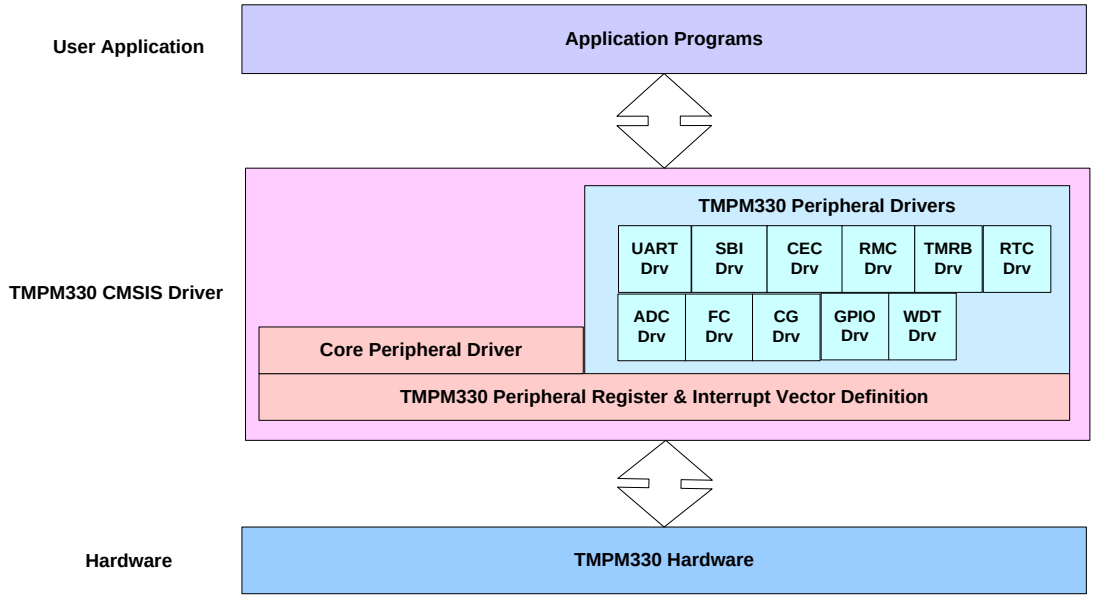
目次

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TMPM341用です。各サンプルプログラムは、主なMCU内蔵機能を単独で実行するように出来ています。サンプルプログラム内の一部を取り出して再利用することで、希望する機能を動作させることができます。

2 概要

TX03 ペリフェラルドライバを下記のように使用します。



3 使用する機能

機能	ブロック/ユニット/チャネル	使用／未使用
CG	クロックギア	使用(CG サンプル)
	PLL	4 低倍
低消費電力モード	-	SLEEP モード
SysTick	-	未使用
ウォッチドッグタイマ (WDT)	-	使用(WDT サンプル)
外部割込み (INT)	INT0	未使用
	INT1	未使用
	INT2	未使用
	INT3	未使用
	INT4	未使用
	INT5	未使用
	INT6	未使用
	INT7	未使用

機能	ブロック/ユニット/チャネル	使用／未使用
シリアルチャネル (SIO/UART)	SIO0	UART0 ,1,2 通信制御 (送信)
	SIO1	UART0, 1 通信制御(受信 / 送信)
	SIO2	UART0, 2 通信制御 (受信)
シリアルバスインタフェース (SBI)	SBI0	スレーブモード
	SBI1	未使用
	SBI2	リードライト EEPROM データ / マスタモード
CEC	-	CEC 送受信機能
リモコン判定(RMC)	RMC0	未使用
	RMC1	リモコン受信
16-bits timer	TMRB0	使用(PPG 出力)
	TMRB1	未使用
	TMRB2	未使用
	TMRB3	使用(周波数測定)
	TMRB4	未使用
	TMRB5	使用(汎用タイマ)
	TMRB6	未使用
	TMRB7	未使用
	TMRB8	使用(周波数測定)
	TMRB9	未使用
RTC	-	使用(RTC サンプル)
10-bit A/D converter	AIN0	使用(ADC データリード)
	AIN1	未使用
	AIN2	未使用
	AIN3	未使用
	AIN4	未使用
	AIN5	未使用
	AIN6	未使用
	AIN7	未使用
	AIN8	未使用
	AIN9	未使用
	AIN10	未使用
	AIN11	未使用

4 端子用途

本サンプルプログラムは、RMCとCECを除いてIAR TMPM330-SKボードを用いてテストされています。RMCとCECはM330 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。

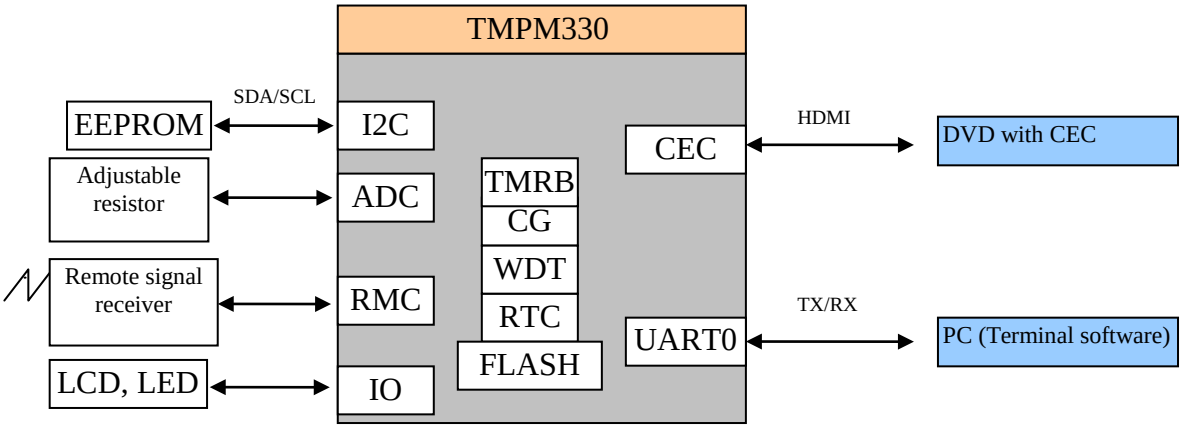
No	Name	Use
1	PD6, AN10	未使用
2	PD7, AN11	未使用
3	AVSS	GND
4	VREFH	+3.3V
5	AVCC	+3.3V
6	PG3, INT4	LED4
7	PK2, TB9OUT	未使用
8	PJ5, TB7OUT	未使用
9	PH4, TB2IN0	未使用
10	PH5, TB2IN1	未使用
11	PG7, TB8OUT	未使用
12	TEST2	未使用
13	DVSS	GND
14	DVCC	+3.3V
15	PG4, SO2, SDA2	SDA2
16	PG5, SI2, SCL2	SCL2
17	PG6, SCK2	未使用
18	TEST1	未使用
19	PF7, INT5	LCD DB7
20	PE0, TXD0	SIO0 TX
21	PE1, RXD0	SIO0 RX
22	PE2, SCLK0, CTS0	SIO0 SCLK0
23	PE4, TXD1	SIO1 TX
24	PE5, RXD1	SIO1 TX
25	PE6, SCLK1, CTS1	SIO1 SCLK1
26	PG0, SO0, SDA0	SDA0/LED1
27	PG1, SI0, SCL0	SCL0/LED2
28	PG2, SCK0	LED3
29	PB3	未使用
30	PH0, TB0IN0, BOOT	未使用
31	PH1, TB0IN1	LCD_E
32	PH2, TB1IN0	LCD_RW
33	PF0, TXD2	SIO2 TX
34	PF1, RXD2	SIO2 RX
35	PF2, SCLK2, CTS2	LCD_BL
36	PH3, TB1IN1	LCD_RS
37	PB4	KEY1
38	PI0, TB0OUT	PPG出力
39	PJ6, INT6	未使用
40	PI1, TB1OUT	未使用
41	PB5	KEY2
42	PI2, TB2OUT	未使用
43	PB6	KEY3
44	PF4, SO1, SDA1	LCD_DB4
45	PF5, SI1, SCL1	LCD_DB5
46	PF6, SCK1	LCD_DB6
47	PB7	KEY4

No	Name	Use
48	PI3, TB3OUT	未使用
49	PJ1, INT1	LED6
50	PK0, CEC	CEC (*)
51	PK1, SCOUT, ALARM	未使用
52	PI4, TB4OUT	未使用
53	PI5, TB5OUT	未使用
54	PB0, TDO, SWV	TDO
55	PA0, TMS, SWDIO	TMS
56	PA1, TCK, SWCLK	TCK
57	TEST3	未使用
58	PJ7, INT7	未使用
59	PB1, TDI	TDI
60	PB2, TRST	TRST
61	PF3, RXIN1	RMC1
62	DVCC	+3.3V
63	DVSS	GND
64	PA2, TRACECLK	未使用
65	PA3, TRACEDATA0	未使用
66	PA4, TRACEDATA1	未使用
67	PA5, TRACEDATA2	未使用
68	PA6, TRACEDATA3	未使用
69	PA7	未使用
70	PJ0, INT0	LED5
71	CVCC	+3.3V
72	X2	外部高速発振(10MHz)
73	CVSS	GND
74	X1	外部高速発振(10MHz)
75	REGVSS	GND
76	REGVCC	+3.3V
77	XT1	外部低速発振(32.768kHz)
78	XT2	外部低速発振(32.768kHz)
79	PI6, TB4IN0	未使用
80	NMI	未使用
81	MODE	GND
82	RESET	RESET
83	PI7, TB4IN1	未使用
84	PH6, TB3IN0	周波数測定用ソースクロック入力
85	PH7, TB3IN1	未使用
86	PJ2, INT2	LED7
87	PJ3, INT3	LED8
88	PJ4, TB6OUT	未使用
89	PE3, RXIN0	未使用
90	TEST4	未使用
91	PC0, AN0	AN0
92	PC1, AN1	未使用
93	PC2, AN2	未使用
94	PC3, AN3	未使用
95	PD0, AN4, TB5IN0	未使用
96	PD1, AN5, TB5IN1	未使用
97	PD2, AN6, TB6IN0	未使用
98	PD3, AN7, TB6IN1	未使用
99	PD4, AN8	未使用

No	Name	Use
100	PD5, AN9	未使用

5 開発環境

以下に開発環境の構成を示します。



1. ハードウェア:

   1) IAR TMPM330-SK ボード

   2) TOSHIBA TMPM330 評価ボード(非売品)

2. 開発ツール:

   - IAR:
     1) J-Link: IAR J-Link-ARM7.0 / J-Link 6.0
     2) IDE: IAR Embedded workbench 5.5 version
   - KEIL:
     1) U-Link: Realview ULINK2
     2) IDE: KEIL uVision 4.10

6 機能説明

6-1 動作モード

本デバイスには 4 つの動作モードがあります: NORMAL, SLOW, IDLE, SLEEP, STOP モード

- **NORMAL モード:**

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

- **SLOW モード:**

高速クロックを停止させ、CPU コアおよび周辺ハードウェアを低速クロックで動作させるモードです。NORMAL モードに比べ消費電力を低減することができます。
SLOW モードでは動作可能な周辺機能が限られます。使用できる周辺機能は、I/O ポート、リアルタイムクロック (RTC)、CEC 機能、リモコン判定機能(RMC)です。

補足: CG サンプルでは SLEEP モードを使用します。

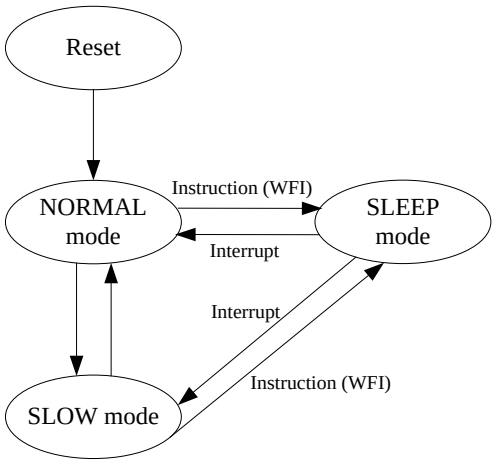
IDLE、SLEEP、STOP の各モードは低消費電力モードです。低消費電力モードへ移行するには、システムコントロールレジスタ STBYCR0<STBY2:0>にて IDLE、SLEEP、STOP のいずれかのモードを選択し、WFI (Wait For Interrupt)命令を実行します。

補足: CG サンプルでは SLEEP モードを使用します。

- **SLEEP モード:**

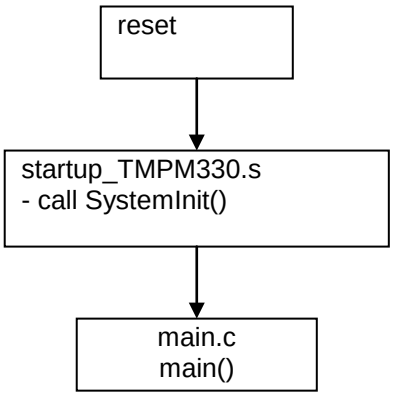
内部低速発信器と RTC、CEC 機能、リモコン判定機能が動作します。SLEEP モードが解除されると、SLEEP モードへ移行する直前の動作モードへ復帰し、動作を開始します。
RTC 割り込み、CEC 割り込み、RMC 割り込み、外部割り込みにて SLEEP モードを解除することができます。

補足: SLEEP モードのサンプルプログラムは、RMC または CEC のサンプルで使用します。



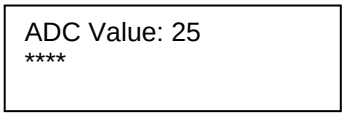
6-2 スタートアップ

スタートアップは、リセット処理からメイン処理の間に実行されます。



6-3 ADC

アナログ入力値を取得し、LCD に表示するサンプルです。



補足: 最初の行に AD 変換結果を 0~100 の間で表示します。
2 行目に進捗をパーセンテージで表示します。

6-4 CEC

CEC ラインからデータを受信します。スタートビットと 8 ビットのデータをホスト PC のターミナル I/O へ出力します。
このサンプルは、CEC メッセージデータを CEC ラインに送信し、同時にスタートビットと 8 ビットのデータをホスト PC のターミナル I/O へ出力します。
M330 評価ボードでは、セルフロジックアドレスは E です。

CEC メッセージに対する応答のため、ホスト PC のターミナル I/O にホスト PC のキーボード操作でキーを入力するか、リモコンキーを入力します。対応している CEC メッセージとトリガキーは次の通りです。

PC keyboard:	Remote key	CEC message
1	[1]	Power (pass through)

2	[2]	System standby
3	[3]	Play (pass through)
4	[4]	Stop (pass through)

Power (pass through)の CEC データサンプル:

CEC_Send: XX 44 40

CEC_Send: XX 45

System standby の CEC データサンプル:

CEC_Send: EF 36

Play (pass through) の CEC データサンプル:

CEC_Send: XX 44 44

CEC_Send: XX 45

Stop (pass through) の CEC データサンプル:

CEC_Send: XX 44 45

CEC_Send: XX 45

* **補足:** 送信アドレスデータの XX は評価環境に依存します。また受信データも評価環境に依存します。

CEC メッセージ受信時に CPU が SLEEP モードの場合は、CPU は NORMAL モードに復帰し、CEC メッセージ受信処理を行います。

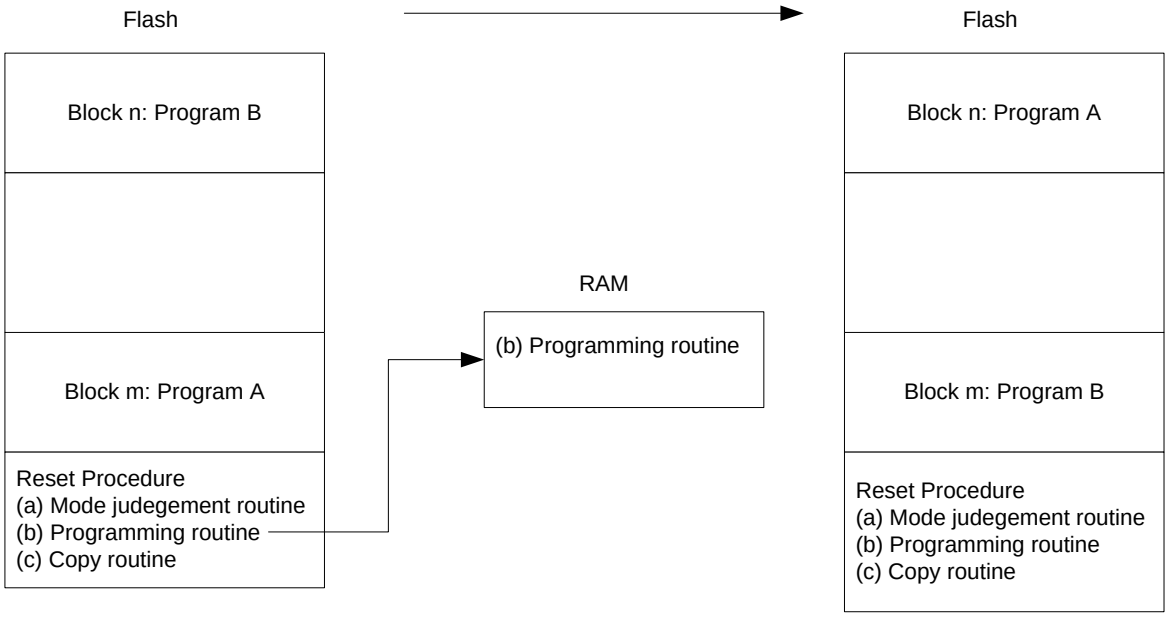
6-5 CG

CPU 動作モードを変更します。このサンプルでは NORMAL モード、SLOW モード、SLEEP モードの 3 つのモードを使用します。

6-6 FLASH

Flash のドライバ API を使用して内蔵フラッシュメモリの消去及び再書き込みを行うサンプルプログラムです。
このプログラムは、シングルチップモードのノーマルモードとユーザブートモードで実行します。
ーリセット動作: モード判定、書き込みルーチン(フラッシュ API)、転送ルーチンで構成されます。
・モード判定ルーチン: ユーザブートモードまたはノーマルモードの判定を行います。
・転送ルーチン: 書き込みルーチンを Flash から RAM へ転送します。

・書き込みルーチン: RAM 上で動作し、フラッシュ上のプログラム A とプログラム B のコードを入れ替えます。
　ープログラム A/B (リセット動作)は事前にフラッシュメモリに書き込まれています。
　ープログラム A/B (A: LED 1~4 点滅, B: LED 5~8 点滅)
初期状態は、プログラム A が動作します。
　ーKEY1 はリセット動作のモード判定に使います。
　　KEY1 が ON された状態　→ ユーザブートモードです。
　　KEY1 が OFF された状態 → Normal モードです。



動作シーケンス

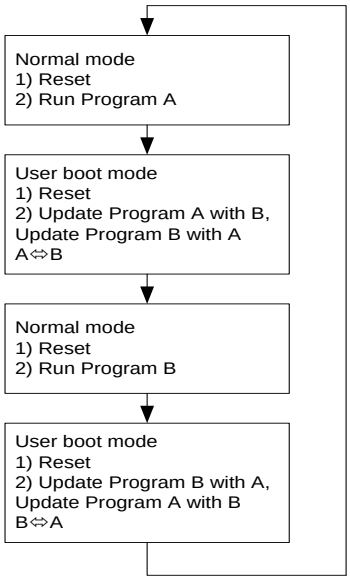
(1) 電源投入
KEY1 を OFF した状態で電源投入またはリセットボタンを押すと、まず実行開始するプログラム A が LED 1~4 を点滅し、LCD に“A is running ……”を表示します。

(2) KEY1 を ON した状態でリセットボタンを押すと、LCD に“User Boot Mode”を表示します。
この間に KEY1 を OFF すると、フラッシュ書き換えルーチンがコピールーチンにより RAM へコピーされます。
その後フラッシュ書き換えルーチンがプログラム A と B をスワップします。
この間、LCD に“RAM transferring ……”, “FLASH burning ……”, “Finished Restart ……”を表示します。

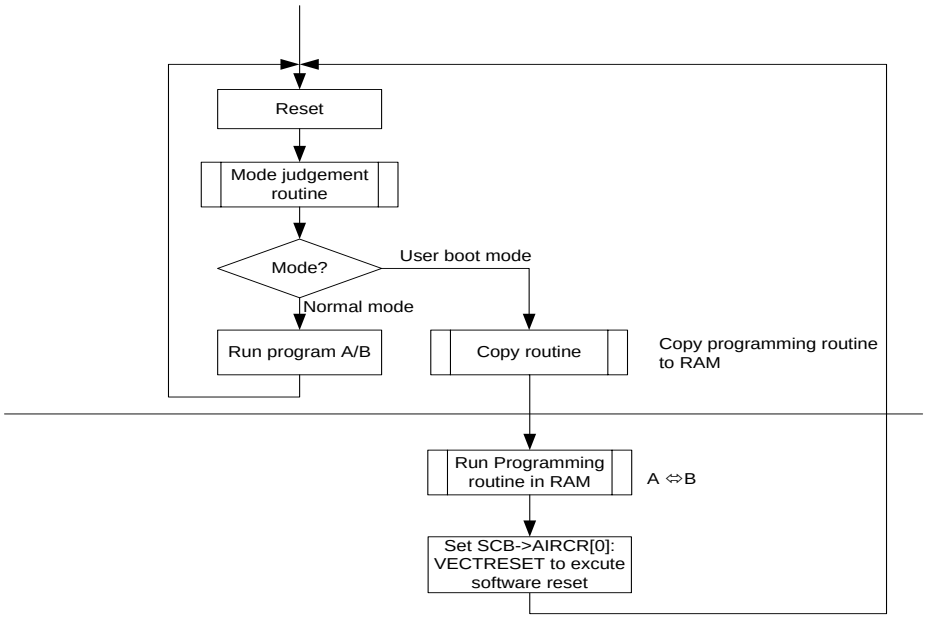
(3) LCD に“Finished Restart ……”を表示し、その後自動的にソフトウェアリセットを行います。
プログラム B が動作して LED 5~8 が点滅します。また LCD に“B is running ……”を表示します。

(4) 再度 KEY1 を ON した状態でリセットボタンを押すと、LCD に“User Boot Mode”を表示します。その間に KEY1 を OFF すると、手順(2)と同じ動作を行います。

(5) LCD に “Finished Restart ……”を表示し、その後自動的にソフトウェアリセットを行います。
実行開始するプログラム A が LED1~4 を点滅し、LCD に “A is running ……” を表示します。



サンプルプログラムのフローチャート



プログラムがユーザブートモードの場合、LCD に以下の文字列を表示します。

User Boot Mode

(リセット処理)

RAM 転送中、または Flash 書き換え中の場合、LCD には以下のように現在の状態を表示します。

RAM transferring ……	FLASH burning ……	Finished Restart ……

(コピー処理) (Flash 書き換え処理) (再起動処理)

6-7 GPIO

GPIO を LED や KEY 用に設定します。

6-8 RMC

6-8-1 RMC 受信

リモコン信号の受信を行い、デコードするサンプルプログラムです。デコードされたデータをホスト PC のターミナル I/O に表示します。カスタムコードまたはアドレスコードは、HEX 形式で表示されます。

表示形式: TMPM330 RMC サンプル
RMC_1: XX XX
RMC_1: YY YY

形式	端末ソフト表示
1	RMC_1:
2	RMC_2:
3	RMC_3:
4	RMC_4:
5	RMC_5:

6-8-2 動作モード変更

CPU 動作モードを変更します。このサンプルでは NORMAL モードと SLEEP モードの 2 つのモードを使用します。リモコンの電源ボタンを On して動作モードを移行させます。

現在のモード	アクション (Key)	動作	ターミナル表示
NORMAL	[9] [CHDOWN] [VOL DOWN]	NORMAL→SLEEP	Now, Going to Sleep
SLEEP	[7] [CH UP] [VOL UP]	SLEEP→NORMAL	Wakeup from Sleep

6-9 RTC

RTC 機能を利用し、LCD に日付や時間を表示します。
初期設定の日付: **2010/10/23 12:50:55** (24 時間表示)
更新間隔: 1 秒
LCD 表示:

2010/10/22 12:50:55

6-10 SBI

6-10-1 EEPROM リード/ライト

ボード上の EEPROM からデータリード、または EEPROM へデータライトを行います。リードデータをホスト PC のターミナル I/O へ表示します。

K1 を ON すると、EEPROM からデータをリードします。

ターミナル I/O 表示: TMPM330 SBI サンプル

0x0000: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX

0x0010: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX

…

Finish

LCD 表示:

TMPM330 EEPROM READ

リード終了後:

Read Over Press K2 to Write

K2 を OM すると、EEPROM へデータをライトします。
文字 (最大 8 文字)をターミナル I/O からライトします。その後、プログラムは EEPROM へデータライトを行います。
ライトデータとリードデータは同一にしてください。
ライト終了後:

Write Over Press K1 to Read

ベリファイするために K1 を ON して EEPROM からデータをリードします。

6-10-2 I2C スレーブ

I2C バスのスレーブモード処理を行うサンプルプログラムです。
評価ボードの 2 本の I2C バス (SBI0 & SBI2) を接続します。片方は I2C バスのマスタモードで動作し、片方は I2C バスのスレーブモードで動作します。
I2C 割り込みを利用して、I2C バスのリード/ライトを行います。
I2C のスレーブアドレス: 任意です
I2C スレーブ (SBI0) は I2C マスタ(SBI2)から “TOSHIBA” を受信し LCD に表示します。

K3 を ON すると、SBI2(マスタ)と SBI0(スレーブ)とでデータ送受信を開始します。

```
K3:SBI2 to SBI0
```

K3 を ON すると、SBI2(マスタ)から SBI0(スレーブ)へ文字列「TOSHIBA」を送信します。

```
Write Over
K1: Show SBI0
```

K1 を ON すると、SBI0(スレーブ)から文字列「TOSHIBA」を受信し、LCD に表示します。

```
TOSHIBA
SBI2 to SBI0 OK
```

補足: 本サンプルソフトを実行するためには、以下のように接続してください。
PG4/SDA2 と PG0/SDA0
PG5/SCL2 と PG1/SCL0

6-11 SIO/UART

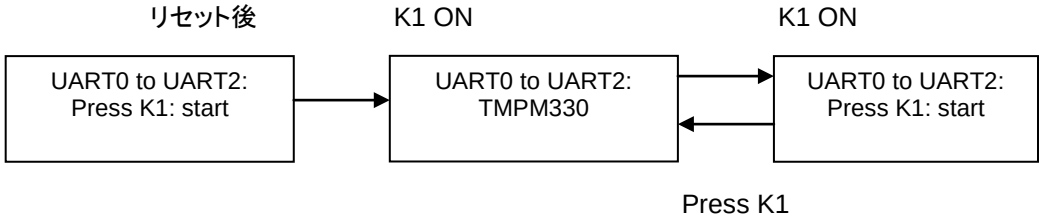
6-11-1 UART0 から UART2 へのデータ転送

UART0 を使用してデータ送信を行い、UART2 を使用してデータ受信を行います。

UART0 からは文字列 “TMPM330” を送信します。

UART2 はデータを受信し、LCD に受信データを表示します。

K1 を ON すると、送信動作を行います。その後、K1 を ON すると、再スタートします。

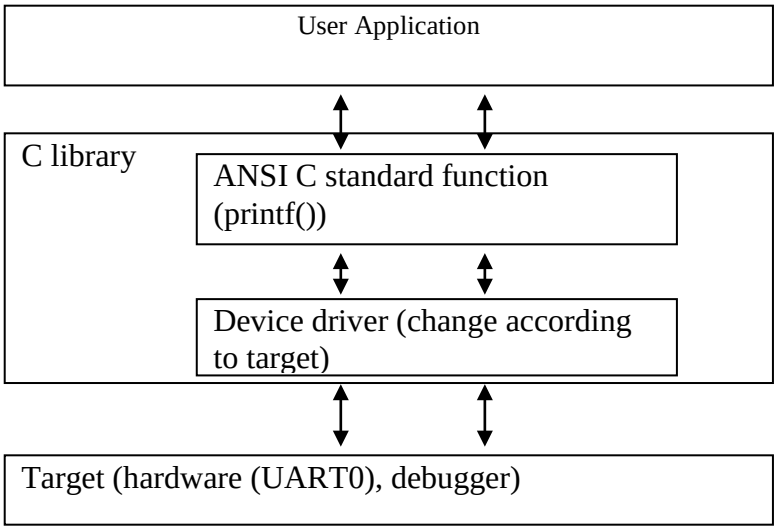


UART 設定:

通信チャネル:	UART0 と UART2
データ長:	8 ビット
パリティビット:	なり
ストップビット:	1 ビット
ボーレート:	115200bps
フロー制御:	なし

6-11-2 リターゲット

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



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

補足: getchar()は CEC サンプルで使用します。

UART 設定:

通信チャネル:	UART0
データ長:	8 ビット
パリティビット:	なり
ストップビット:	1 ビット
ボーレート:	115200bps
フロー制御:	なし

6-11-3 UART FIFO

In this sample program, UART0 sent the data "TMPM3301" to UART1 use FIFO, and at same time UART0 receive the data "TMPM3302" which send from UART1 and also use FIFO.
Set one breakpoint before the function ResetIdx(),when program stop in the breakpoint you can read the RxBuffer = "TMPM3302",and RxBuffer1 = "TMPM3301".
FIFO を UART0 から"TMPM3301"というデータを使用して UART1 へ送信し、同時に FIFO を UART1 から"TMPM3302"というデータを使用して UART0 へ送信するサンプルプログラムです。
ResetIdx()関数の前にブレークポイントを設定しておくと、RxBuffer=”TMPM3302”と RxBuffer1="TMPM3301"となった場合にブレークポイントで停止します。

6-11-4 SIO

SIO 機能を使用して同期式の送受信を行うサンプルプログラムです。
SIO のチャネル 0 とチャネル 1 を使用し、これらのチャネル間で同期式データ送信を行います。
(TXD0 と RXD1、TXD1 と RXD0、sclk0 と sclk1 を接続してください)

6-12 TMRB

6-12-1 汎用タイマ

TMRB を使って汎用タイマを実現するサンプルプログラムです。
時間周期は 1ms です。
このタイマを使い 1s(500ms でオン、500ms でオフ)間隔で LED 点滅を行います。

Interval Timer

6-12-2 PPG 出力

1 つのスイッチを使い、デューティ可変の PPG(プログラマブル矩形波)の波形を出力します。
デューティは 5 段階(10%, 25%, 50%, 75%, 90%)に変更できます。
電源投入すると PPG モードに入り、PPG 出力を開始します。
スイッチの ON/OFF を切り替えることでデューティを変更します。
(10%, 25%, 50%, 75%, 90%)

電源投入にて TMRB を PPG モードに設定し、PPG 出力を行います。

K1: Change LeadingTiming

K1 を ON すると、デューティを変更します。
(10% → 25% →50% → 75% → 90% → 10%)

LCD 表示:

PPG Output LeadingTiming: 50%

6-12-3 周波数測定

TMRB を使用して周波数測定を行います。

K3: Freq Measure

K3 を ON すると、周波数測定を行います。
LCD 表示:

Freq measure f=99999Hz

補足: PH6/TB3IN0 は周波数測定用の端子として使用します。

6-13 WDT

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

ドライバ使用方法ステップ:

1) 検出時間の設定とカウンタオーバーフロー時の動作としての WDT 割り込み設定を行い、WDT を初期化します。
2) 2 つの WDT 用サンプルがあり、DEMO2 はマクロ定義にて切り替えられます。
   DEMO1:
   タイマーオーバーフロー時に NMI 割り込みを発生し、WDT をクリアします。
   DEMO2:
   ウォッチドッグタイマ制御レジスタにクリアコードを書き込み、ウォッチドッグタイマカウンタをクリアします。

7 Software

本ソフトウェアは、主要機能は IAR TMPM330-SK ボードを使用し、一部 M330 評価ボードを使用し、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
│   │   tmpm330_wdt_int.c
│   │
│   ├─TX03_CMSIS
│   │   │   core_cm3.c
│   │   │   core_cm3.h
│   │   │   system_TMPM330.c
│   │   │   system_TMPM330.h
│   │   │   TMPM330.h
│   │   │
│   │   └─startup
│   │       startup_TMPM330.s
│   │
│   ├─TX03_Periph_Driver
│   │   ├─inc
│   │   │     tmpm330_wdt.h
│   │   │     tx03_common.h
│   │   │
│   │   └─src
│   │         tmpm330_wdt.c
│   └─TMPM330-SK
│                 key.c
│                 key.h
│                 lcd.c
│                 lcd.h
│                 led.c
│                 led.h
```

KEIL MDK:

```
├─WDT_NMI
│   ├─APP
│   │   main.c
│   │   tmpm330_wdt_int.c
│   │
│   ├─TX03_CMSIS
│   │   core_cm3.c
│   │   system_TMPM330.c
│   │   startup_TMPM330.s
│   │
│   ├─TX03_Periph_Driver
│   │   tmpm330_wdt.c
│   │
```

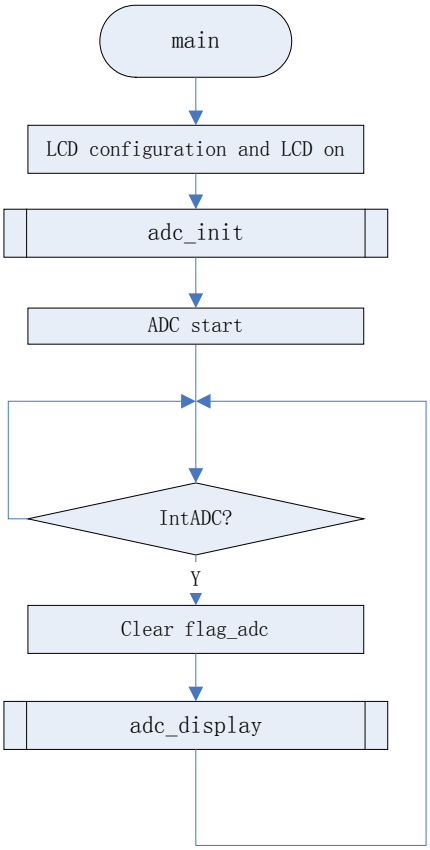
└TMPM330-SK
key.c
lcd.c
led.c

7-1 ADC

ペリフェラルドライバ(ADC)を使用したサンプルプログラムです。

以下の例が含まれます:
1. ADC 設定と初期化
2. ADC 変換結果の取得

- **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、ADC_Init()関数をコールして、リピートモードの許可と割り込み要因の設定を行い、VREF を N します。

```
ADC_SWReset();
ADC_SetInputChannel(ADC_AN_0);
ADC_SetRepeatMode(ENABLE);
ADC_SetINTMode(ADC_INT_SIGNLE);
ADC_SetVref(ENABLE);
```

AD 変換を開始します。

```
ADC_Start();
```

AD 変換を行い、その後 AD 変換終了割り込みを待ちます。AD 変換終了割り込みの発生後、
ADC_Display()関数をコールします。
ADC_Display()関数では、AD 変換結果をリードします。

```
ADC_ResultTypeDef ADC_Result;
ADC_Result = ADC_GetConvertResult(ADC_REG_08);
```

7-2 CEC

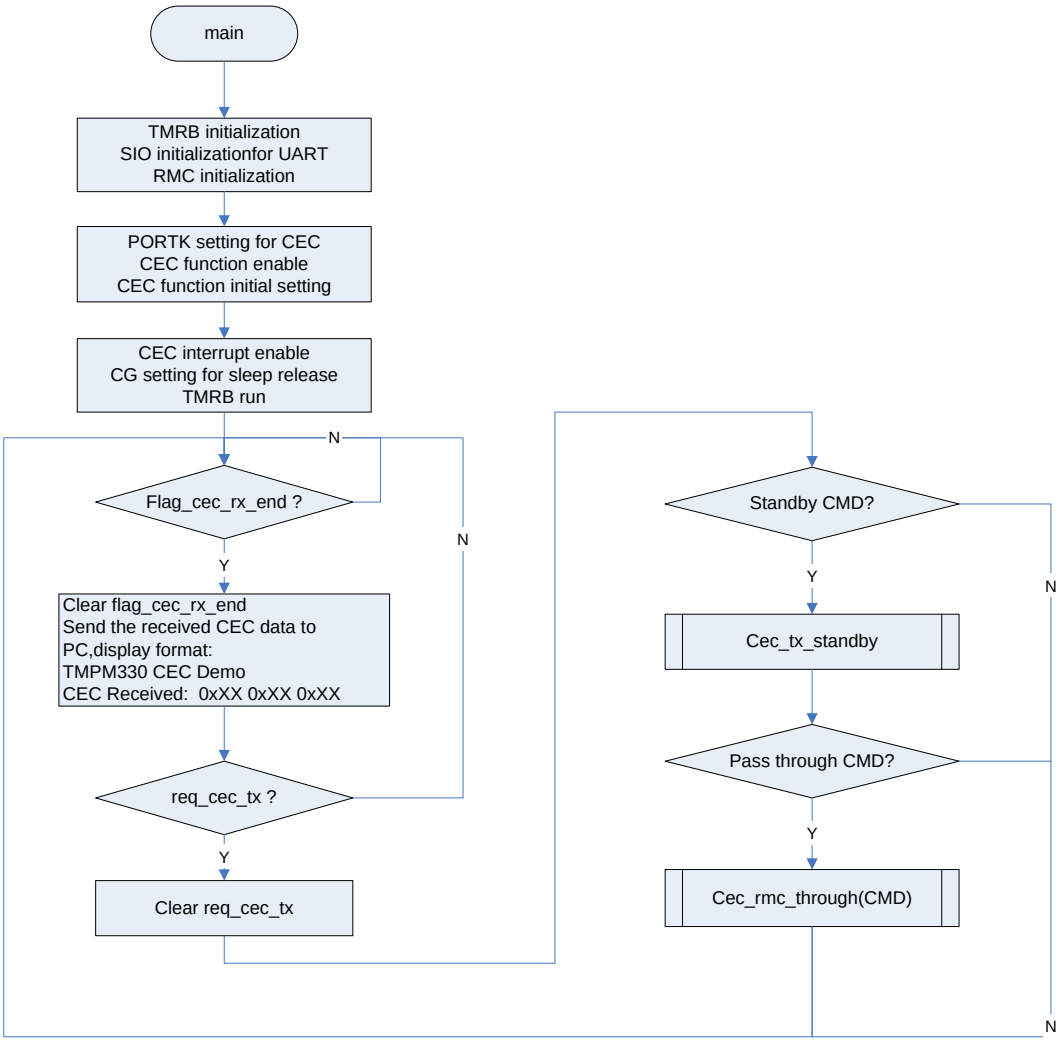
7-2-1 例: CEC 送受信

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

このサンプルでは以下を行います。

1. CEC の設定と初期化
2. CEC コマンドの送信
3. CEC データの受信

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、GPIO を CEC に設定します。

```
GPIO_SetOutputEnableReg(GPIO_PK, GPIO_BIT_0, ENABLE);
GPIO_EnableFuncReg(GPIO_PK, GPIO_FUNC_REG_1, GPIO_BIT_0);
GPIO_SetInputEnableReg(GPIO_PK, GPIO_BIT_0, ENABLE);
```

CEC を許可し、CEC の初期化を行います。

```
CEC_Enable();
CEC_SWReset();
while (CEC_GetRxState() == ENABLE);
while (CEC_GetTxState() == BUSY);

CEC_DefaultConfig();
CEC_SetLogicalAddr(CEC_TV);
CEC_SetTxBroadcast(DISABLE);
CEC_SetRxCtrl(ENABLE);

gCEC_SendStatus.All = 0U;
gCEC_SendMode = 0U;
gCEC_RcvMode = 0U;
gCEC_RcvCnt = 0U;
gCEC_RcvEnd_Flag = 0U;

gCEC_Command_Mode = CEC_CMD_MODE_IDLE;
```

データ送受信を行うため、CEC 割り込みを許可します。

```
NVIC_EnableIRQ(INTCECRX_IRQn);
NVIC_EnableIRQ(INTCECTX_IRQn);
```

上記設定後、CEC データ送受信を行います。
CEC データをホスト PC に表示します。

```
    CEC_CMD_Task();
    CEC_Receive();
    /* CEC Received data display on PC          */
    if (gCEC_RcvEnd_Flag == SET) {
        gCEC_RcvEnd_Flag = CLEAR;
        printf("CEC_Recv: ");
        for (tmp_i = 0U; tmp_i <= gCEC_RcvCnt; tmp_i++) {
            printf(" %02x", gCEC_RcvDataBuf[tmp_i]);
        }
        printf("\r\n");
        gCEC_RcvCnt = 0U;
    } else {
        /* do nothing */
    }

    /* CEC Send data display on PC          */
    if (gCEC_SndEnd_Flag == SET) {
        gCEC_SndEnd_Flag = CLEAR;
        printf("CEC_Send: ");
        for (tmp_i = 0U; tmp_i <= gCEC_SendCnt; tmp_i++) {
            printf(" %02x", gCEC_SendDataBuf[tmp_i]);
        }
        printf("\r\n");
        gCEC_SendCnt = 0U;
    } else {
        /* do nothing */
```

```
    }
```

INTCECTX ハンドラ内でデータ送信を行います。
INTCECRX ハンドラ内でデータ受信を行います。

7-3 CG

ペリフェラルドライバ(CG)を使用したサンプルプログラムです。

このサンプルでは以下を行います:

1. 基本的な CG の設定
2. NORMAL モードと SLOW モードの切り替え方法
3. マルチクロック回路の許可方法

- **サンプルプログラムのコードと説明**

CG の初期化

以下は NORMAL モードで CG の設定を行う例です。(高速発振 10MHz の場合)

```
   /* set fgear = fc/2 */
   CG_SetFgearLevel(CG_DIVIDE_2);
   /* set fperi = fgear  fpreclk = fperi/32 */
   CG_SetPhiT0Level(CG_DIVIDE_64);
   /* set SCOUT source to fT0 */
   CG_SetSCOUTSrc(CG_SCOUT_SRC_PHIT0);
   /* enable high-speed oscillation */
   CG_SetFosc(ENABLE);
   /* enable low-speed oscillation */
   CG_SetFs(ENABLE);
   /* set low power consumption mode sleep */
   CG_SetSTBYMode(CG_STBY_MODE_SLEEP);
   /* set high-speed and low-speed oscillation to be enabled after releasing stop mode
*/
   CG_SetExitStopModeFosc(ENABLE);
   CG_SetExitStopModeFs(ENABLE);
   /* set pin status in stop mode to "active" */
   CG_SetPinStateInStopMode(ENABLE);
   /*set up pll and wait 200us for pll to warm up , set fc source to fpll */
   CG_EnableClkMulCircuit();
   /* set system clock to fgear */
   CG_SetFsysSrc(CG_FSYS_SRC_FGEAR);
```

SLOW モードへの遷移設定

システムクロックとして fs を選択し、高速クロックを停止します。

```
   CG_SetFsysSrc(CG_FSYS_SRC_FS);
   CG_SetFosc(DISABLE);
```

SLEEP モードへの遷移設定

SLEEP モード解除要因(RTC 割り込み)を設定します。

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

RTC 割り込みの割り込み保留ラッチレジスタをクリアし、RTC 割り込みを許可します。

```
   NVIC_ClearPendingIRQ(INTRTC_IRQn);
   NVIC_EnableIRQ(INTRTC_IRQn);
```

```
CG_ClearINTReq(CG_INT_SRC_RTC);
```

RTC のアラーム機能の設定を行い、2 分後に RTC 割り込みを発生するようにします。

WFI 命令を実行し、SLEEP モードに遷移します。

```
_WFI();
```

補足: SLEEP モードを解除するには割り込みが必要です。RTC 割り込み発生まで 2 分間待つ必要があります。

SLOW モードから NORMAL モードへの復帰

高速発振を許可します。

```
CG_SetFosc(ENABLE);
```

Fosc の安定待ちのためウォームアップを行い、1.953m 秒待ちます。

```
CG_SetWarmUpTime(CG_WARM_UP_SRCXT1, CG_WARM_UP_TIME_EXP_6);
CG_StartWarmUp();
```

ウォームアップが終了するまでポーリングします。

```
while(CG_GetWarmUpState() == BUSY);
```

システムクロックとして Fgear を選択します。

```
CG_SetFsysSrc(CG_FSYS_SRC_FGEAR);
```

マルチクロック回路を許可します。

```
Result CG_EnableClkMulCircuit(void)
{
    Result retval = ERROR;
    WorkState st = BUSY;
    retval = CG_SetPLL(ENABLE);
    if (retval == SUCCESS) {
        /*set warm up time to about 200us */
        retval = CG_SetWarmUpTime(CG_WARM_UP_SRC_X1,
CG_WARM_UP_TIME_EXP_11);
        if (retval == SUCCESS) {
            CG_StartWarmUp();
            /*wait warm up to end */
            do {
                st = CG_GetWarmUpState();
            } while (st != DONE);
            retval = CG_SetFcSrc(CG_FC_SRC_FPLL);
        } else {
            /*Do nothing */
        }
    } else {
        /*Do nothing */
    }
    return retval;
}
```

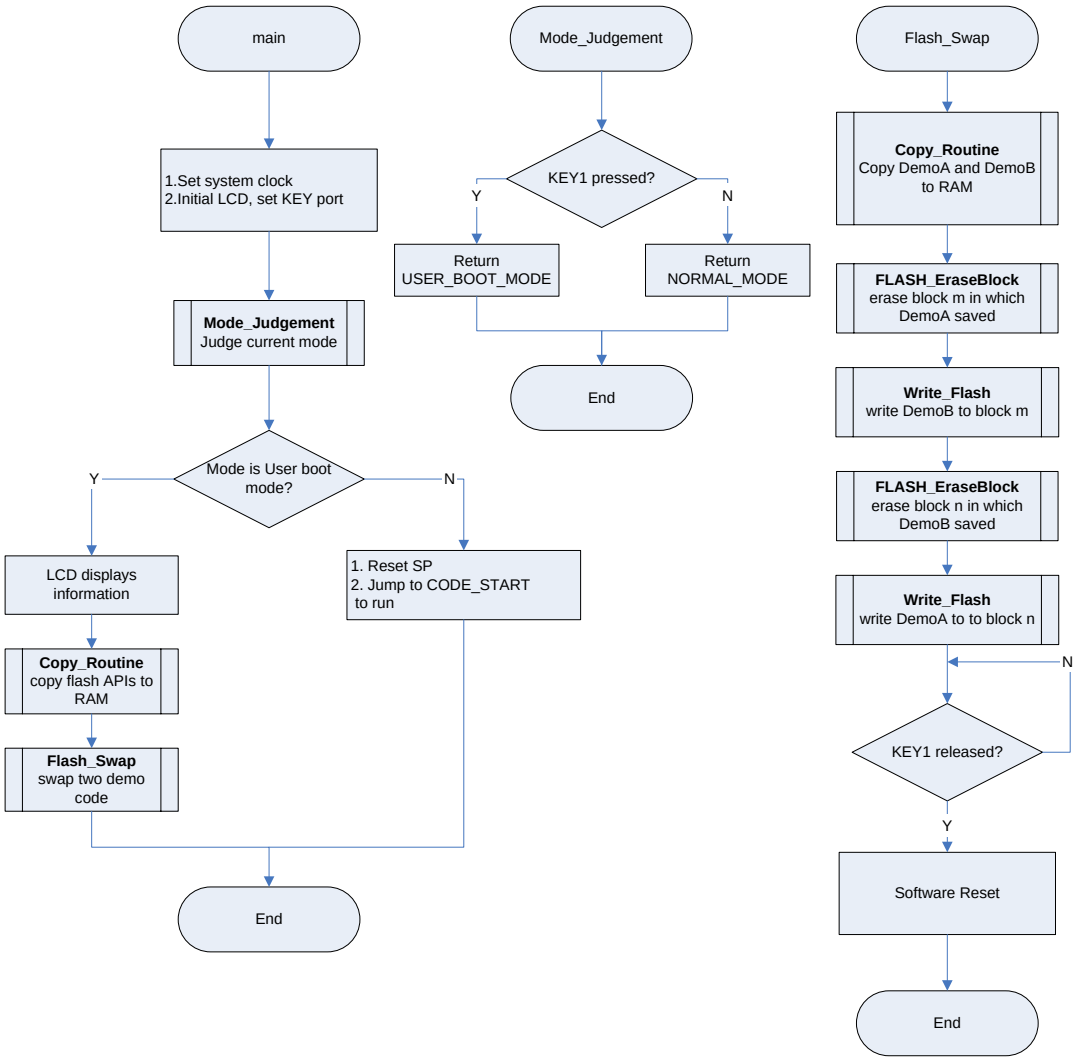
7-4 FLASH

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

このサンプルでは以下を行います:

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まずシステムクロックの設定を行い、LCD と KEY を初期化します。現在のモード判定を KEY で、現在の状態表示を LCD で行います。

```
CG_SetPLL(DISABLE);                 /* Disable PLL */
CG_SetFcSrc(CG_FC_SRC_FOSC);        /* Select fosc */
KEY_Configuration();                /* init KEY */
LCD_Configuration();                /* init LCD */
LCD_Light(ON);                      /* turn on the light of LCD */
```

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

```
uint8_t Mode_Judgement(void)
{
```

```
    return (KEY_Get(KEY1)) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

リセット時に KEY が離されている場合、ノーマルモードになります。SP をリセットし、“CODE_START” にジャンプします。プログラム A はブロックm内の“CODE_START” に保存されているため、プログラム A が動作します(LED 1~4 が点灯)。

```
    #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 が押されている場合、ユーザブートモードになります(LCD に"User Boot Mode"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、Flash 動作用 API を、Flash メモリ内の アドレス “FLASH_API_ROM” から RAM の “FLASH_API_RAM”にコピーします。RAM 転送中、LCD に"RAM transferring"を表示します。

```
    LCD_Display("User Boot Mode", "");
    LCD_Display("RAM transferring", ".....");
    /* copy flash API to RAM */
    Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
```

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

Flash_Swap() 関数では,まずサンプル A、サンプル B を Flash メモリから RAM にコピーします。次に、Flash メモリの消去/書き込みを行うため、関数 FC_EraseBlock () と Write_Flash() を呼び出します。Flash メモリ内のプログラム A とプログラム B がスワップします。

```
    /* copy A to RAM */
    Copy_Routine(DEMO_A_RAM, DEMO_A_FLASH, SIZE_DEMO_A);

    /* copy B to RAM */
    Copy_Routine(DEMO_B_RAM, DEMO_B_FLASH, SIZE_DEMO_B);

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

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

    /* erase B in block n */
    if (FC_SUCCESS == FLASH_EraseBlock(DEMO_B_FLASH)) {
         /* Do nothing */
    } else {
```

```
        return ERROR;
    }

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

スワップが完了するとLCDに"Finished", "Restart....."を表示します。KEYが離されると、SCB->AIRCR レジスタを用いてソフトリセットを行います。その後、ノーマルモードになります。サンプルAとサンプルBが差し替えられているため、アドレス “CODE_START” はサンプルBのスタートアドレスになり、サンプルBが動作します(LED5~8が が点灯)。

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

    while (KEY_Get(KEY1)) {     /* wait for KEY1 release */
    }
    reg_value = SCB->AIRCR;     /* software reset */
    reg_value = (uint32_t) 0x05FA0001;
    SCB->AIRCR = reg_value;
```

Flash メモリ動作関数 FC_EraseBlock() は指定されたブロックを消去します。このブロックは最初に引数 “Block_addr” で指定します。まず、この関数で引数“Block_addr” を確認します。次に、Flash ドライバ FC_GetBlockProtectState() を使用し、指定されたブロックがプロテクトされているか確認します。

```
    if (FC_GetBlockProtectState((FC_BlockNum) Addr_To_BN(blk_adr)) == ENABLE)
    {       /* check if this address is protected */
         return ERR_PRO;
    } else {
         /* do nothing */
    }
```

ブロックにプロテクトがかかっている場合、“ERR_PRO” を返します。プロテクトされていない場合は、ブロック消去コマンドにて、ブロックを消去します。

```
    addr1 = (uint32_t *) (FLASH_BEGIN + FLASH_CMD_BC1_ADDR);
    addr2 = (uint32_t *) (FLASH_BEGIN + FLASH_CMD_BC2_ADDR);
    *addr1 = FLASH_CMD_BC1_DATA;        /* bus cycle 1 */
    *addr2 = FLASH_CMD_BC2_DATA;        /* bus cycle 2 */
    *addr1 = 0x00000080;                 /* bus cycle 3 */
    *addr1 = FLASH_CMD_BC1_DATA;        /* bus cycle 4 */
    *addr2 = FLASH_CMD_BC2_DATA;        /* bus cycle 5 */
    *blk_adr = 0x00000030;               /* bus cycle 6 */
```

次に、消去が完了すると、Flash ドライバ FC_GetBusyState() を用いビジーチェックを行います。同時に、タイムアウトカウンタを使用し、動作が指定時間を越えていないか確認します。

```
    while (BUSY == FC_GetBusyState()) {      /* check if FLASH is busy */
        if (!(counter--)) {                   /* check overtime */
            *addr2 = 0xf0;                    /* Reset FLASH */
             return ERR_OT;
        } else {
            /* do nothing */
        }
    }
```

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

```
addr1 = (uint32_t *) (FLASH_BEGIN + FLASH_CMD_BC1_ADDR);
addr2 = (uint32_t *) (FLASH_BEGIN + FLASH_CMD_BC2_ADDR);
*addr1 = FLASH_CMD_BC1_DATA;        /* bus cycle 1 */
*addr2 = FLASH_CMD_BC2_DATA;        /* bus cycle 2 */
*addr1 = 0x000000a0;                /* bus cycle 3 */
source = data;
len = len & 0xfffc;
for (i = 0; i < (len / 4); i++) {   /* bus cycle 4~n */
   *page_adr = *source;             /* 32 bits write */
   source++;
}
```

7-5 GPIO

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。

このサンプル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

- **サンプルプログラムのコードと説明**

まず GPIO_Init()関数をコールして GPIO を LED に設定します。以下は GPIO_InitTypeDef 構造体を生成し、メンバに初期値を設定する例です。

```
GPIO_InitTypeDef led_io;
led_io.IOMode = GPIO_OUTPUT_MODE;
led_io.PullUp = GPIO_PULLUP_ENABLE;
led_io.PullDown = GPIO_PULLDOWN_NONE;
led_io.OpenDrain = GPIO_OPEN_DRAIN_NONE;
```

その後、GPIO_Init()関数をコールし、LED の初期化を行います。

```
GPIO_Init(GPIO_PG, GPIO_BIT_0, &led_io);
```

while()ループにて、LED on と LED off を行います。

GPIO_WriteData ()関数をコールし、指定ビットに”1”をライトすることで LED を on します。

```
GPIO_WriteDataBit(GPIO_PG, GPIO_BIT_0, GPIO_BIT_VALUE_1);
```

GPIO_WriteData ()関数をコールし、指定ビットに”0”をライトすることで LED を off します。

```
GPIO_WriteDataBit(GPIO_PG, GPIO_BIT_0, GPIO_BIT_VALUE_0);
```

7-6 RMC

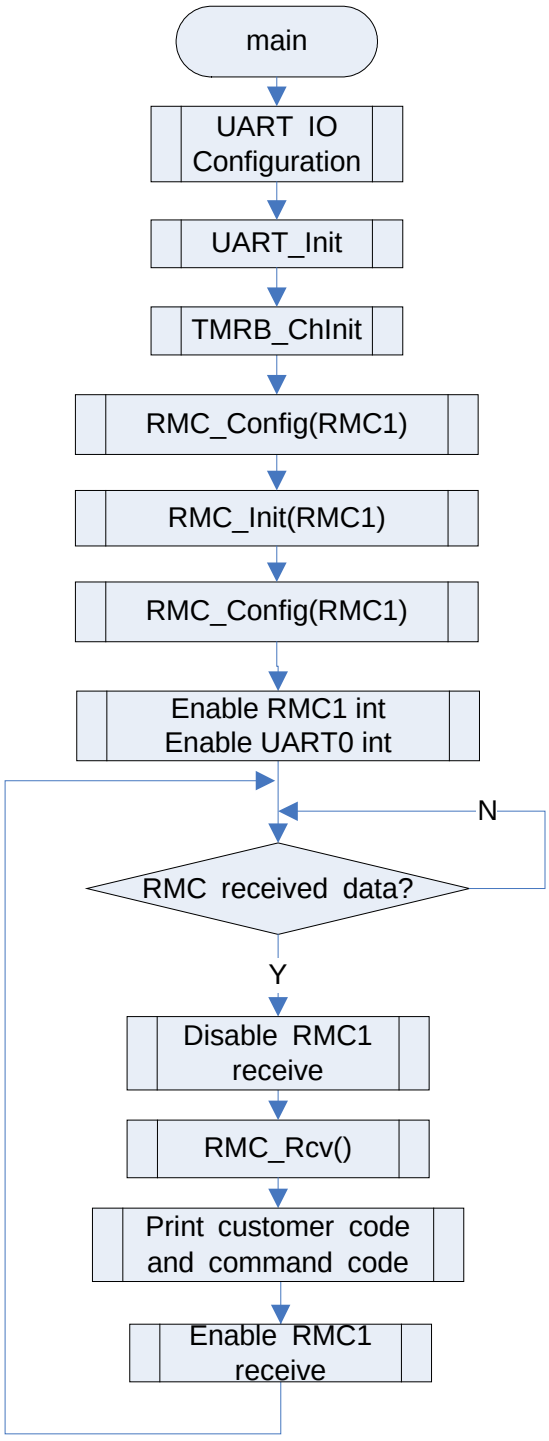
7-6-1 例: RMC 受信

ペリフェラルドライバ (RMC, GPIO, UART, TMRB)のサンプルプログラムです。

このサンプルでは以下を行います。

1. RMC の初期化
2. RMC データの受信

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、GPIO の設定、UART0、TMRB0、RMC1 の初期化を行います。ただし、UART0 と TMRB0 用 GPIO の設定は他の章を参照してください。ここでは RMC1 用 GPIO の設定内容を説明します。

GPIO の設定、RMC1 の初期化を行います。

```
GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_1, GPIO_BIT_3);
GPIO_SetInputEnableReg(GPIO_PF, GPIO_BIT_3, ENABLE);
```

RMC_InitTypeDef 構造体を作成し、全てのデータ項目を設定します。ここでは RMC レジスタには TOSHIBA フォーマットを設定します。

```
RMC_InitTypeDef myRMC;

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

RMC チャネル 1 の許可と初期化を行います。RMC1 の受信を許可します。

```
RMC_Enable(RMCx);
RMC_Init(RMCx, &myRMC);
RMC_SetRxCtrl(RMCx, ENABLE);
```

上記設定後、UART0 の送信割り込みと受信割り込みを許可します。

```
NVIC_EnableIRQ(INTTX0_IRQn);
NVIC_EnableIRQ(INTRMCRX1_IRQn);
```

RMC 割り込みハンドラ内で、割り込み要因の取得を行い、リーダー検出結果の取得やデータ受信を行います。

```
if (myRMC_INTFactor.Bit.MaxDataBitCycle) {  /*Max data bit cycle detection
interrupt factor */

    myRMC_LeaderDetection = RMC_GetLeader(TSB_RMC1);

    if (myRMC_LeaderDetection == RMC_LEADER_DETECTED) {     /*Leader has
been detected */

        myRMC_RxDataDef = RMC_GetRxData(TSB_RMC1);
        if (myRMC_RxDataDef.RxDataBits == 0x20U) {              /*32 bits data
received */

            uRMCBuf.reg = myRMC_RxDataDef.RxBuf1;            /* save the RMC data
for print in main() */

            gRMCTimeOut = 0U;       /* reset time-out count */
            if (fRMCPress == RELEASE) {
                fRMCPress = PRESS;
                } else {
                    /* do nothing */
```

```
                }
            } else {
                if (fRMCPress) {            /* repeat */
                    gRMCTimeOut = 0U;   /* reset time-out count */
                } else {
                    /* do nothing */
                }
            }
        }
    }
```

printf()にて受信データを UART 送信します。

```
    printf("%s: %02x %02x\r\n", ch, addr, cmd);
```

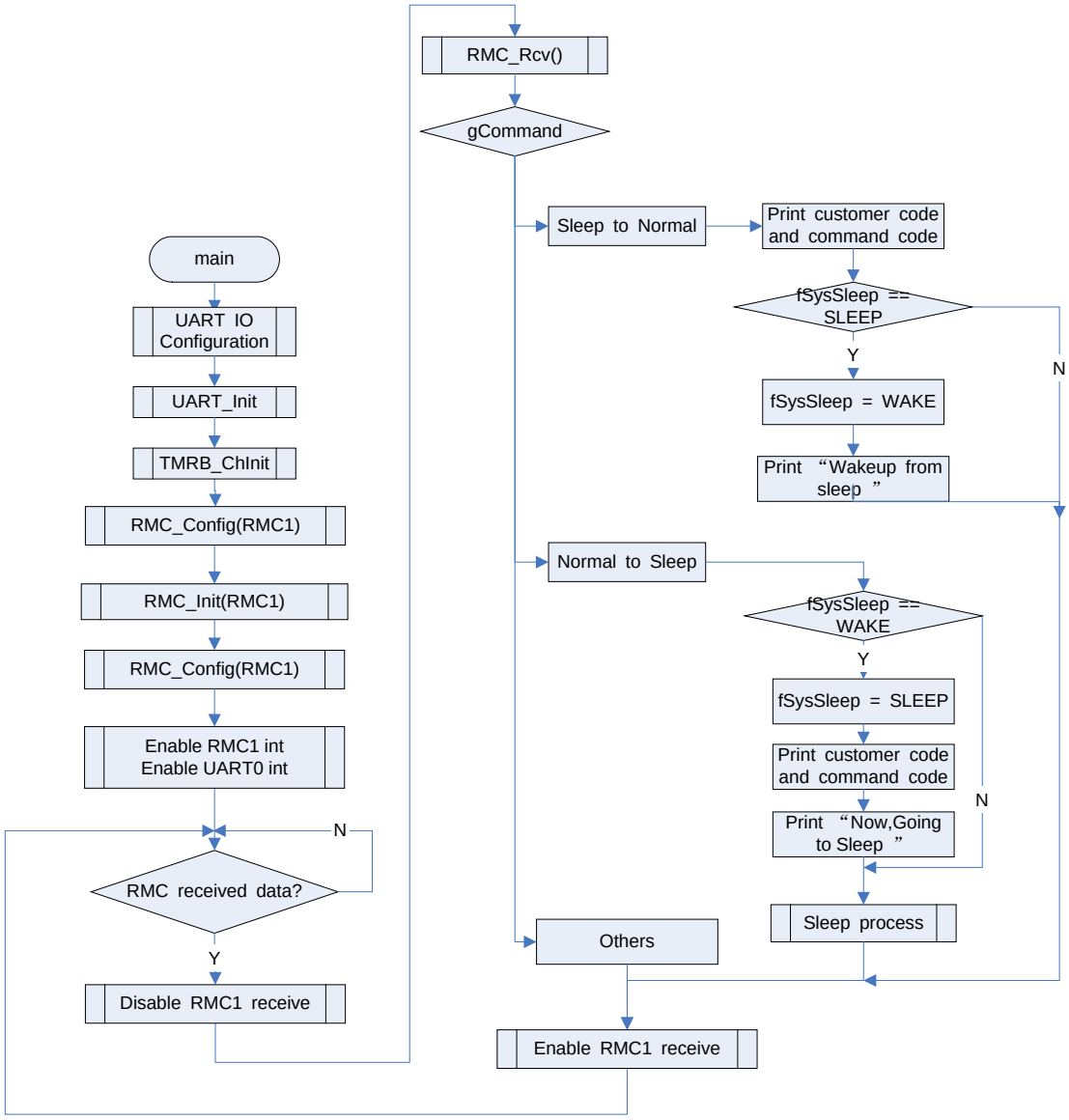
7-6-2 例: 動作モード変更

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

このサンプルでは以下を行います。

1. RMC データ受信 (上述と同じ処理です)
2. 動作モード変更(NORMAL モード→SLEEP モード)

- **フローチャート:**



- **サンプルプログラムのコードと説明**

RMC データ受信処理は上述と同じため、ここではおもに動作モードの変更(NORMAL モード→SLEEP モード)を行います。

SLEEP モードの解除要因として RMC1 割り込みを設定します。

```
CG_SetSTBYReleaseINTSrc(CG_INT_SRC_RMC_RX_1,
CG_INT_ACTIVE_STATE_RISING, ENABLE);
```

低速クロックを許可します。

```
CG_SetFs(ENABLE);
```

発振安定待ち時間 1.953ms を設定(ウォーミングアップ用ソースクロック:XT1)し、ウォーミングアップを開始します。

```
CG_SetWarmUpTime(CG_WARM_UP_SRC_XT1, CG_WARM_UP_TIME_EXP_6);
CG_StartWarmUp();          /* start warm up */
```

ウォーミングアップ完了を待ちます。

```
while (CG_GetWarmUpState() == BUSY)
```

発振安定待ち時間 3.2768 ms を設定します。(ウォーミングアップ用ソースクロック: X1)
低消費電力モードとして SLEEP モードを選択します。

```
CG_SetWarmUpTime(CG_WARM_UP_SRC_X1, CG_WARM_UP_TIME_EXP_15);
CG_SetSTBYMode(CG_STBY_MODE_SLEEP);
```

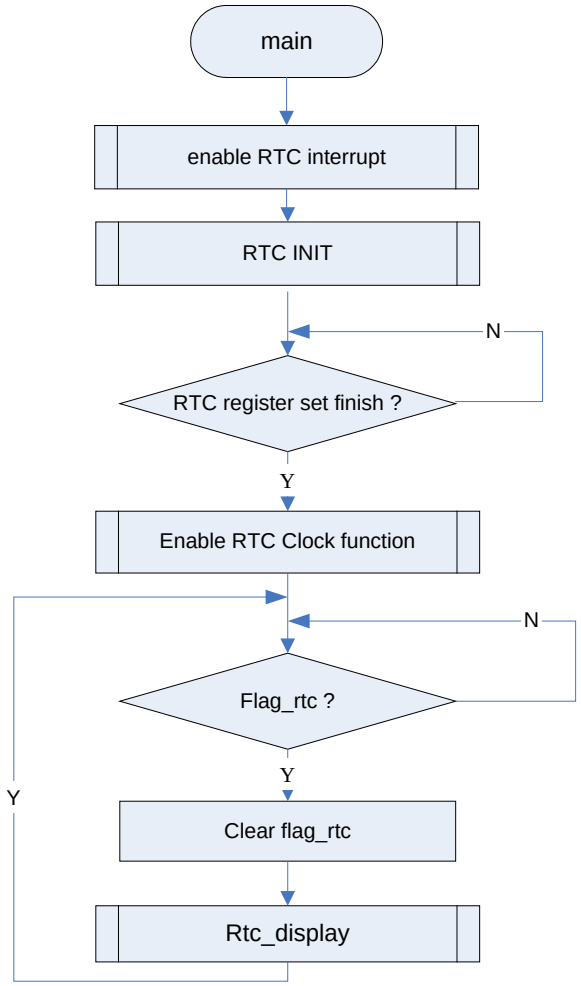
7-7 RTC

ペリフェラルドライバ (RTC, CG) のサンプルプログラムです。

このサンプルでは以下を行います。

1 RTC の時間設定
2 RTC の時間取得

- **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、SLEEP モード解除要因として RTC 割り込みを設定します。

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

RTC の時間と日付の値を設定します。ここでの初期値は、2010/10/22 12:50:55、24 時間フォーマットです。

```
RTC_DateTypeDef Date_Struct;

Date_Struct.LeapYear = RTC_LEAP_YEAR_2;
Date_Struct.Year = (uint8_t) 10U;
Date_Struct.Month = (uint8_t) 10U;
Date_Struct.Date = (uint8_t) 22U;
Date_Struct.Day = RTC_FRI;

RTC_TimeTypeDef Time_Struct;

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;
```

時間機能とアラーム機能を禁止します。

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

RTC の秒カウンタのクリアと 1Hz 割り込みの設定後、RTC 割り込みを許可します。

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

RTC の時間と日付を設定します。

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

上記設定後、NVIC の RTC 割り込みを許可します。RTC 設定完了のため、1 秒待ちます。その後、RTC 時間機能を許可します。

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

RTC 割り込みが 1 秒ごとに発生するするように設定します。その後、CG の RTC 割り込み要因をクリアします。

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

割り込み発生後、RTC の日時の値を取得し、LCD に表示します。

```
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

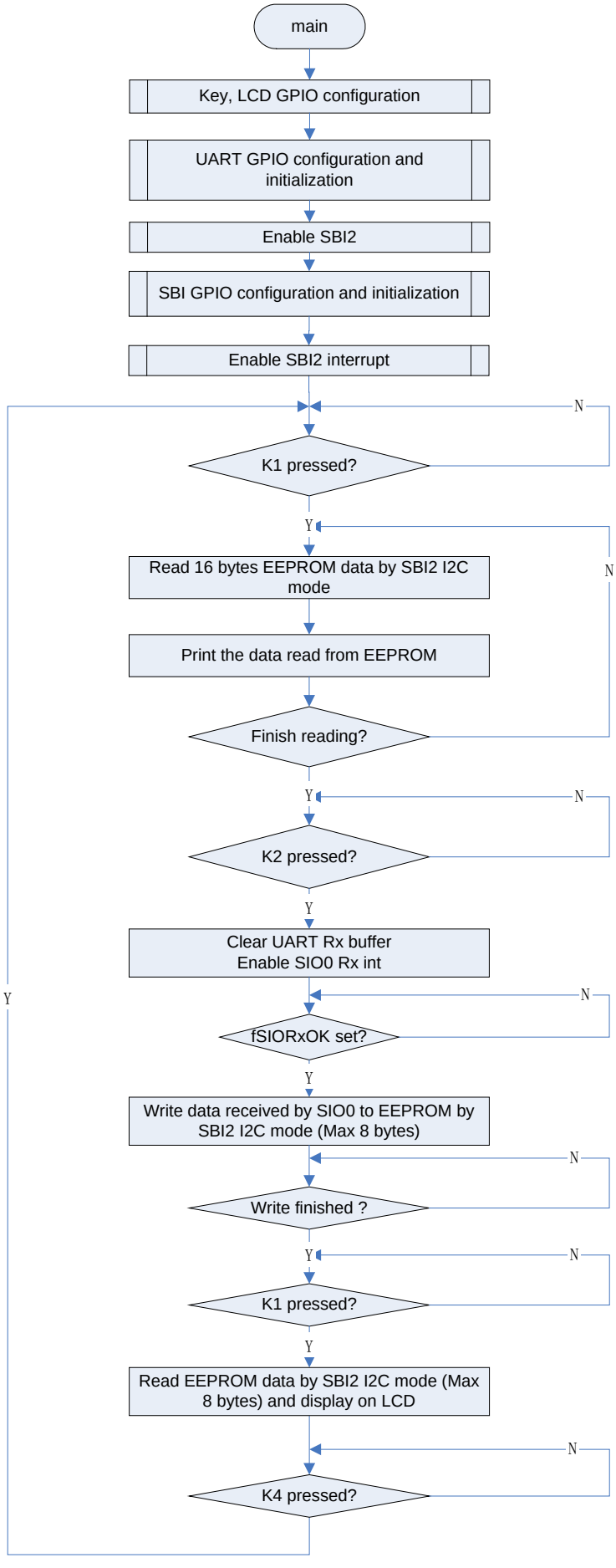
7-8-1 例: EEPROM リード/ライト

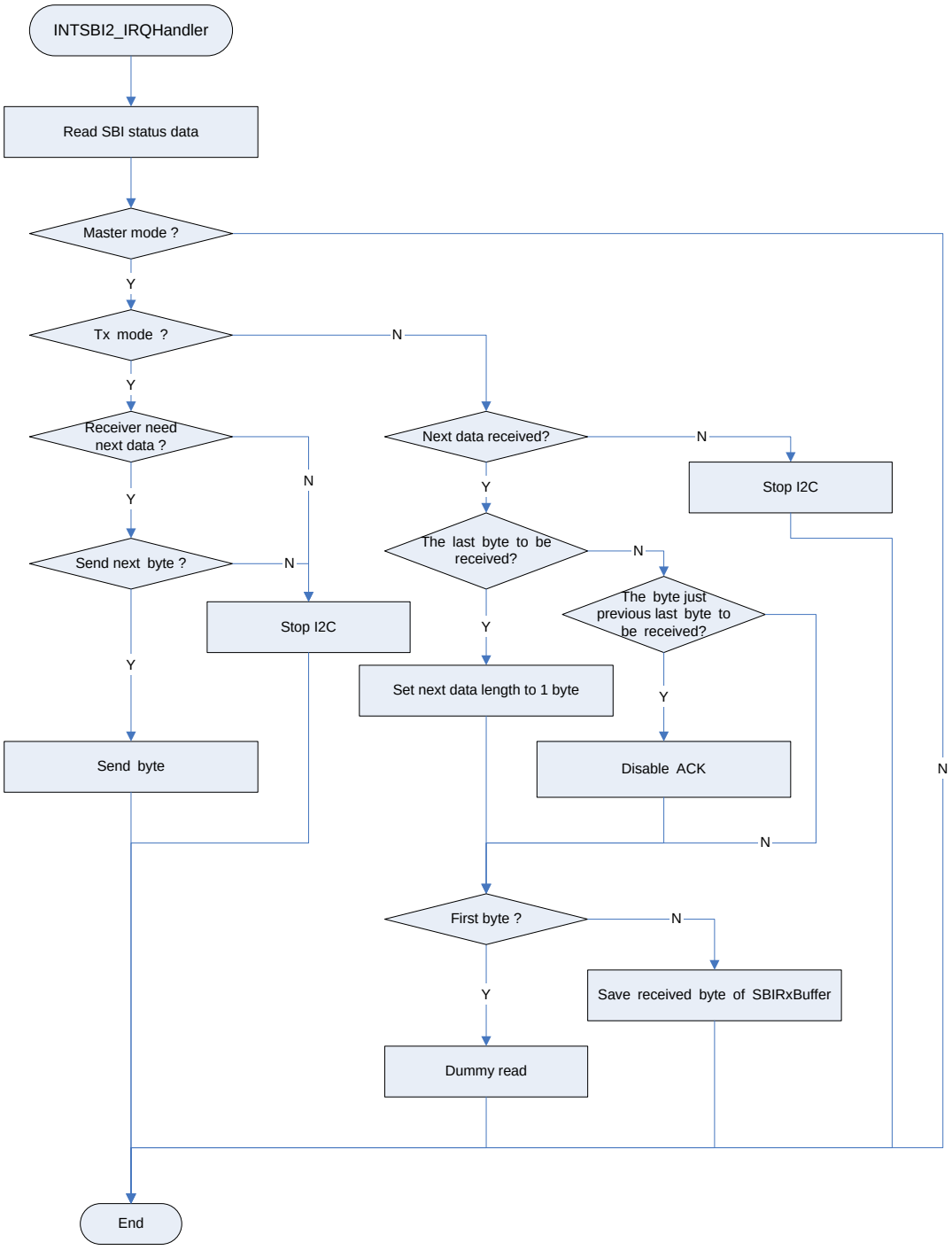
ペリフェラルドライバ(SBI, GPIO, UART)を使用したサンプルプログラムです。

このサンプルでは以下を行います。

1. SBI の設定と I2C の初期化
2. I2C のデータ送信

- **フローチャート:**


main
Key, LCD GPIO configuration
UART GPIO configuration and initialization
Enable SBI2
SBI GPIO configuration and initialization
Enable SBI2 interrupt
K1 pressed?
N
Y
Read 16 bytes EEPROM data by SBI2 I2C mode
Print the data read from EEPROM
Finish reading?
N
Y
K2 pressed?
N
Y
Clear UART Rx buffer
Enable SIO0 Rx int
fSIORxOK set?
N
Y
Write data received by SIO0 to EEPROM by SBI2 I2C mode (Max 8 bytes)
Write finished ?
N
Y
K1 pressed?
N
Y
Read EEPROM data by SBI2 I2C mode (Max 8 bytes) and display on LCD
K4 pressed?
N
Y



- **サンプルプログラムのコードと説明**

まず、SBI2 の I2C モード用に GPIO を設定します。

```
GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_1, GPIO_BIT_4);
GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_1, GPIO_BIT_5);
GPIO_SetOutputEnableReg(GPIO_PG, GPIO_BIT_4, ENABLE);
GPIO_SetOutputEnableReg(GPIO_PG, GPIO_BIT_5, ENABLE);
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_4, ENABLE);
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_5, ENABLE);
GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_4, ENABLE);
GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_5, ENABLE);
/* Pull up for SDA&SCL */
```

```
GPIO_SetPullUp(GPIO_PG, GPIO_BIT_5, ENABLE);
GPIO_SetPullUp(GPIO_PG, GPIO_BIT_4, ENABLE);
```

その後 SBI2 の設定を行い、INTSBI2 割り込みを許可します。

```
myI2C.I2CSelfAddr = SELF_ADDR;
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2C.I2CACKState = ENABLE;
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
SBI_Enable(TSB_SBI2);
SBI_SWReset(TSB_SBI2);
SBI_InitI2C(TSB_SBI2, &myI2C);
NVIC_EnableIRQ(INTSBI2_IRQn);
```

上記設定後、EEPROM リードを行います。
まず I2C の送信バッファを初期化します。I2C バスがフリーとなっていることを確認し、EEPROM デバイスアドレスと転送先を設定します。転送先は"SBI_I2C_SEND"です。その後スタートコンディションを送信します。

```
ClearBuffer(gI2C_RxData, SBI_BUFFER_SIZE);
do {
    i2c_state = SBI_GetI2CState(TSB_SBI2);
} while (i2c_state.Bit.BusState);
gI2C_WCnt = 0U;
gI2C_RCnt = 0U;
gI2C_TxData[0] = iCnt * SIZE;
gI2C_TxDataLen = 1U;
gI2C_RxDataLen = SIZE;
SBI_SetSendData(TSB_SBI2, EEPROM_ADDR | SBI_I2C_SEND);
SBI_GenerateI2CStart(TSB_SBI2);
```

EEPROM のリードアドレスを INTSBI2 割り込みハンドラ内で送信します。

I2C バスがフリーとなっていることを確認し、EEPROM デバイスアドレスと転送先を設定します。転送先は"SBI_I2C_RECEIVE"です。その後スタートコンディションを送信します。

```
do {
   i2c_state = SBI_GetI2CState(TSB_SBI2);
} while (i2c_state.Bit.BusState);

SBI_SetSendData(TSB_SBI2, EEPROM_ADDR | SBI_I2C_RECEIVE);
SBI_GenerateI2CStart(TSB_SBI2);
```

EEPROM リード処理を終了後、INTSBI2 割り込みハンドラ内で UART から EEPROM へ受信したデータをライトします。
SBI 送信バッファとバッファ長を初期化します。I2C バスがフリーとなっていることを確認し、EEPROM デバイスアドレスと転送先を設定します。転送先は"SBI_I2C_SEND"です。その後、スタートコンディションを送信します。

```
gI2C_TxData[0] = SUB_ADDR;
for (jCnt = 0U; jCnt < strlen(gSIORxBuffer); jCnt++) {
   gI2C_TxData[jCnt + 1U] = gSIORxBuffer[jCnt];
}
gI2C_TxDataLen = jCnt + 1U;
gI2C_RxDataLen = 0U;
gI2C_WCnt = 0U;
do {
   i2c_state = SBI_GetI2CState(TSB_SBI2);
} while (i2c_state.Bit.BusState);
SBI_SetSendData(TSB_SBI2, EEPROM_ADDR | SBI_I2C_SEND);
SBI_GenerateI2CStart(TSB_SBI2);
```

INTSBI2 割り込みハンドラ内にて、EEPROM ライト処理と gI2C_TxData[]の残りのデータをライトします。

INTSBI2 割り込みハンドラ内にて、I2C マスタ送信とスレーブ受信の各処理が I2C バス状態によって行われます。
I2C マスタ送信中、SBI_SetSendData()関数をコールして次のデータを送信します。I2C 転送処理を終了する場合は SBI_GenerateI2CStop()関数をコールします。
I2C スレーブ受信中、SBI_GetReceiveData()関数をコールして SBI データバッファから I2C データを受信します。次のデータを受信後、SBI_SetI2CACK()関数をコールして ACK 応答を停止します。I2C を停止するため、SBI_SetI2CBitNum()関数をコールして 1 クロックだけ設定します。I2C 受信処理を終了する場合は SBI_GenerateI2CStop()関数をコールします。同時に SBI_SetI2CACK()関数をコールして、I2C を再設定します。

```
void INTSBI2_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI_TypeDef *SBIx;
    SBI_I2CState sbi_sr;

    SBIx = TSB_SBI2;
    sbi_sr = SBI_GetI2CState(SBIx);

    if (sbi_sr.Bit.MasterSlave) {       /* Master mode     */
        if (sbi_sr.Bit.TRx) {   /* Tx mode        */
            if (sbi_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further data. */
                SBI_GenerateI2CStop(SBIx);
            } else {            /* LRB=0: the receiver requires further data. */
                if (gI2C_WCnt < gI2C_TxDataLen) {
                    SBI_SetSendData(SBIx, gI2C_TxData[gI2C_WCnt]);      /* Send next data
*/
                } else if (gI2C_WCnt == gI2C_TxDataLen) {       /* I2C data send finished. */
                    SBI_GenerateI2CStop(SBIx);
                } else {
                    /* Do nothing */
                }
                gI2C_WCnt++;
            }
        } else {                /* Rx Mode */
            if (gI2C_RCnt > gI2C_RxDataLen) {
                SBI_GenerateI2CStop(SBIx);
                SBI_SetI2CACK(SBIx, ENABLE);
            } else {
                if (gI2C_RCnt == gI2C_RxDataLen) {      /* Rx last data */
                    SBI_SetI2CBitNum(SBIx, SBI_I2C_DATA_LEN_1);
                } else if (gI2C_RCnt == (gI2C_RxDataLen - 1U)) {        /* Rx the data second
to last */
                    /* Not generate ACK for next data Rx end. */
                    SBI_SetI2CACK(SBIx, DISABLE);
                } else {
                    /* Do nothing */
                }
                tmp = SBI_GetReceiveData(SBIx);
                if (gI2C_RCnt > 0U) {
                    gI2C_RxData[gI2C_RCnt - 1U] = tmp;
                } else {
                    /* first read is dummy read */
                }
                gI2C_RCnt++;
            }
```

```
        }
    } else {                    /* Slave mode  */
        /* Do nothing */
    }
}
```

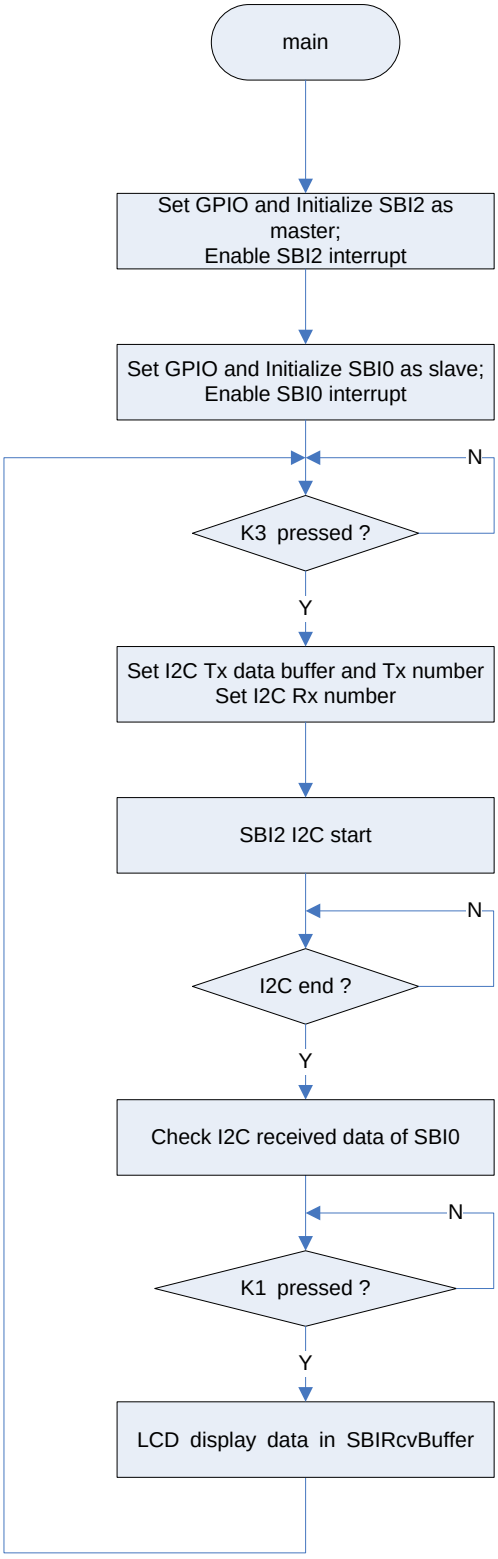
7-8-2 例: I2C スレーブ

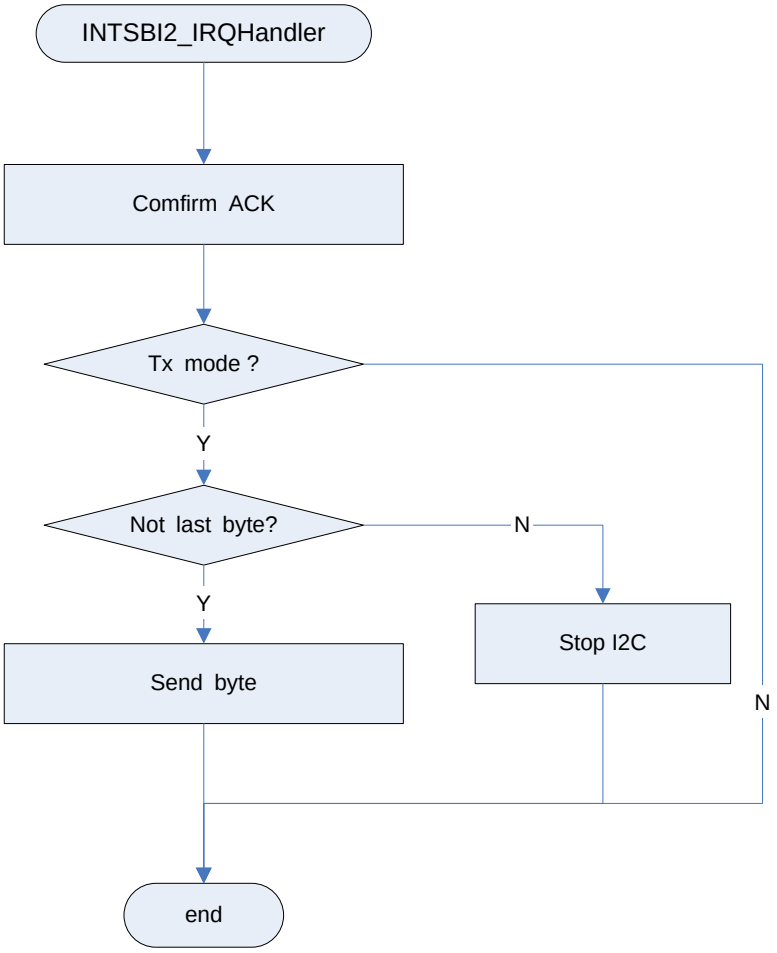
ペリフェラルドライバ(SBI、GPIO)を使用したサンプルプログラムです。

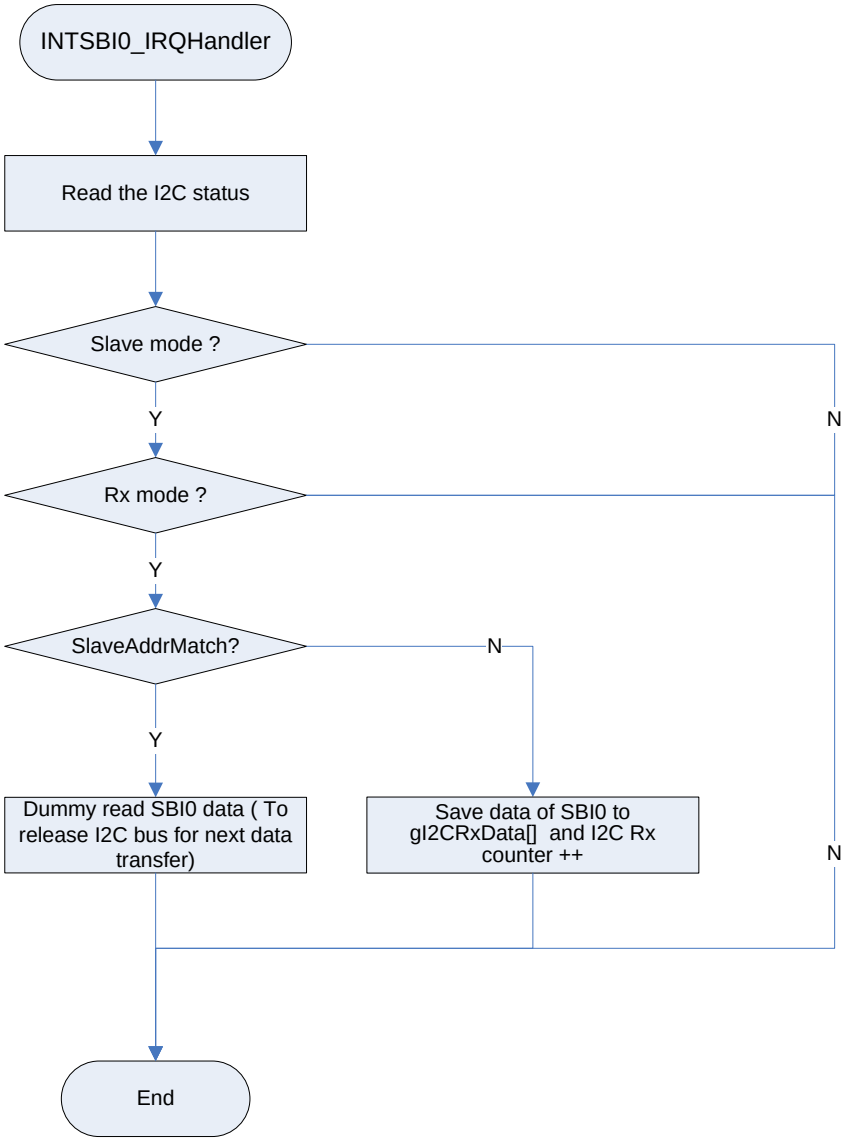
このサンプルでは以下を行います。

1. SBI の設定と I2C の初期化
2. I2C マスタ送信処理
3. I2C スレーブ受信処理

- **フローチャート:**


main
Set GPIO and Initialize SBI2 as master;
Enable SBI2 interrupt
Set GPIO and Initialize SBI0 as slave;
Enable SBI0 interrupt
N
K3 pressed ?
Y
Set I2C Tx data buffer and Tx number
Set I2C Rx number
SBI2 I2C start
N
I2C end ?
Y
Check I2C received data of SBI0
N
K1 pressed ?
Y
LCD display data in SBIRcvBuffer


INTSBI2_IRQHandler
Comfirm ACK
Tx mode ?
Y
Not last byte?
N
Y
Stop I2C
Send byte
N
end



- **サンプルプログラムのコードと説明**

まず、SBI2 の I2C モード用に GPIO を設定します。

```
GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_1, GPIO_BIT_4);
GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_1, GPIO_BIT_5);
GPIO_SetOutputEnableReg(GPIO_PG, GPIO_BIT_4, ENABLE);
GPIO_SetOutputEnableReg(GPIO_PG, GPIO_BIT_5, ENABLE);
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_4, ENABLE);
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_5, ENABLE);
GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_4, ENABLE);
GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_5, ENABLE);
/* Pull up for SDA&SCL */
GPIO_SetPullUp(GPIO_PG, GPIO_BIT_5, ENABLE);
GPIO_SetPullUp(GPIO_PG, GPIO_BIT_4, ENABLE);
```

SBI0 の I2C モード用に GPIO を設定します。

```
GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_1, GPIO_BIT_0);
GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_1, GPIO_BIT_1);
GPIO_SetOutputEnableReg(GPIO_PG, GPIO_BIT_0, ENABLE);
GPIO_SetOutputEnableReg(GPIO_PG, GPIO_BIT_1, ENABLE);
```

```
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_0, ENABLE);
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_1, ENABLE);
GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_0, ENABLE);
GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_1, ENABLE);
/* Pull up for SDA&SCL */
GPIO_SetPullUp(GPIO_PG, GPIO_BIT_1, ENABLE);
GPIO_SetPullUp(GPIO_PG, GPIO_BIT_0, ENABLE);
```

SBI2 を初期化し、NVIC の INTSBI2 割り込みを許可します。

```
myI2C.I2CSelfAddr = SELF_ADDR;
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2C.I2CACKState = ENABLE;
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
SBI_Enable(TSB_SBI2);
SBI_SWReset(TSB_SBI2);
SBI_InitI2C(TSB_SBI2, &myI2C);
NVIC_EnableIRQ(INTSBI2_IRQn);
```

SBI0 を初期化し、NVIC の INTSBI0 割り込みを許可します。

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

上記設定後、I2C 送信を開始します。
受信バッファ、送信バッファとバッファ長の初期化を行います。

```
    /* Initialize TRx buffer and Tx length */
case MODE_SBI_I2C_INITIAL:
    gI2CTxDataLen = 7U;
    gI2CTxData[0] = gI2CTxDataLen;
    gI2CTxData[1] = 'T';
    gI2CTxData[2] = 'O';
    gI2CTxData[3] = 'S';
    gI2CTxData[4] = 'H';
    gI2CTxData[5] = 'I';
    gI2CTxData[6] = 'B';
    gI2CTxData[7] = 'A';
    gI2CWCnt = 0U;
    for (gICnt = 0U; gICnt < 8U; gICnt++) {
        gI2CRxData[gICnt] = 0U;
    }
    gSBIMode = MODE_SBI_I2C_START;
    break;
```

I2C バスがフリーとなっていることを確認し SBI_SetSendData()関数をコールして、データ“SLAVE_ADDR”を設定します。転送先に “SBI_I2C_SEND” を設定します。その後、SBI_GenerateI2CStart()関数をコールして、I2C 処理を開始します。

```
    /* Check I2C bus state and start TRx */
case MODE_SBI_I2C_START:
    i2c_state = SBI_GetI2CState(TSB_SBI2);
    if (!i2c_state.Bit.BusState) {
        SBI_SetSendData(TSB_SBI2, SLAVE_ADDR | SBI_I2C_SEND);
        SBI_GenerateI2CStart(TSB_SBI2);
        gSBIMode = MODE_SBI_I2C_TRX;
```

```
    } else {
        /* Do nothing */
    }
    break;
```

データ転送は INTSBI2 割り込みハンドラ内で行います。

データ受信は INTSBI0 割り込みハンドラ内で行います。

INTSBI2 関数にて I2C バス状態に応じた I2C マスタ送信処理を行います。I2C マスタ送信中、SBI_SetSendData()関数をコールして次のデータ送信を行います。I2C 転送を終了する場合、SBI_GenerateI2CStop()関数をコールして、I2C 転送を終了します。

```
void INTSBI2_IRQHandler(void)
{
    TSB_SBI_TypeDef *SBIx;
    SBI_I2CState sbi_sr;

    SBIx = TSB_SBI2;
    sbi_sr = SBI_GetI2CState(SBIx);

    if (sbi_sr.Bit.MasterSlave) {       /* Master mode */
        if (sbi_sr.Bit.TRx) {   /* Tx mode */
            if (sbi_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further data. */
                SBI_GenerateI2CStop(SBIx);
            } else {            /* LRB=0: the receiver requires further data. */
                if (gI2CWCnt <= gI2CTxDataLen) {
                    SBI_SetSendData(SBIx, gI2CTxData[gI2CWCnt]);        /* Send next data */
                    gI2CWCnt++;
                } else {        /* I2C data send finished. */
                    SBI_GenerateI2CStop(SBIx);  /* Stop I2C */
                }
            }
        } else {                /* Rx Mode */
            /* Do nothing */
        }
    } else {                    /* Slave mode */
        /* Do nothing */
    }
}
```

INTSBI0 割り込み関数にて、I2C スレーブ受信処理を行います。SBI_GetReceiveData()関数をコールして SBI バッファの I2C データを取得します。I2C ストップコンディションはマスタ側が制御します。

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

    SBIy = TSB_SBI0;
    sbi0_sr = SBI_GetI2CState(SBIy);

    if (!sbi0_sr.Bit.MasterSlave) {     /* Slave mode */
        if (!sbi0_sr.Bit.TRx) { /* Rx Mode */
            if (sbi0_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRCnt = 0U;
```

```
        } else {
            /* Read I2C received data and save to I2C_RxData buffer */
            tmp = SBI_GetReceiveData(SBIy);
            gI2CRxData[gI2CRCnt] = tmp;
            gI2CRCnt++;
        }
    } else {            /* Tx Mode */
        /* Do nothing */
    }
  } else {              /* Master mode */
    /* Do nothing */
  }
}
```

7-9 SIO/UART

7-9-1 例: UART0 から UART2 へのデータ転送

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

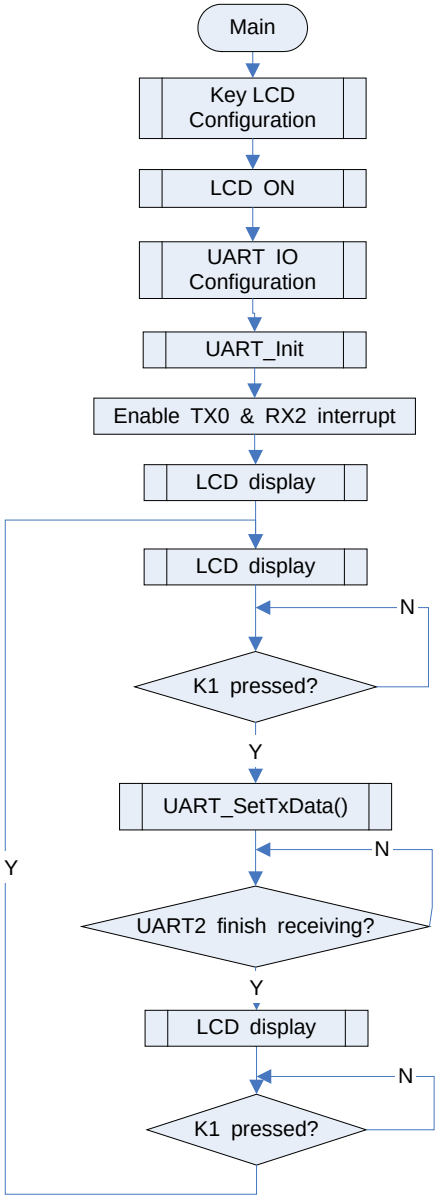
このサンプルでは以下を行います。

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

このサンプルでは以下を行います。

1. UART 設定と初期化
2. UART 送受信制御
3. UART0 送信割り込みを使用したデータ送信
4. UART2 受信割り込みを使用した UART0 からのデータ受信

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、GPIO を UART0 用に設定します。

```
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);
```

GPIO を UART2 用に設定します。

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

UART_InitTypeDef 構造体を準備し、構造体メンバに初期値を設定します。

```
UART_InitTypeDef myUART;

myUART.BaudRate = 115200U;  /* baut 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_RX | UART_ENABLE_TX;
  myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
```

UART を許可し、UART0 と UART2 の初期化を行います。

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

  UART_Enable (UART2);
  UART_Init(UART2, &myUART);
```

上記設定を行い、UART0 の送信割り込みと UART2 の受信割り込みを許可します。

```
  NVIC_EnableIRQ(INTTX0_IRQn);
  NVIC_EnableIRQ(INTRX2_IRQn);
```

次にデータ送信を行います。ここでは送信バッファに文字列を代入しています。

```
  UART_SetTxData(UART0, TxBuf[0]);
```

残りの処理は UART0 の送信割り込みルーチンと、UART0 の受信割り込みルーチンです。

UART0 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);
  }
}
```

UART2 受信割り込みルーチン:

```
void INTRX2_IRQHandler(void)
{
  volatile UART_Err err;

  err = UART_GetErrState(UART2);
  if (UART_NO_ERR == err) {
    RxBuffer[RxCounter++] = (uint8_t) UART_GetRxData(UART2);
  }
}
```

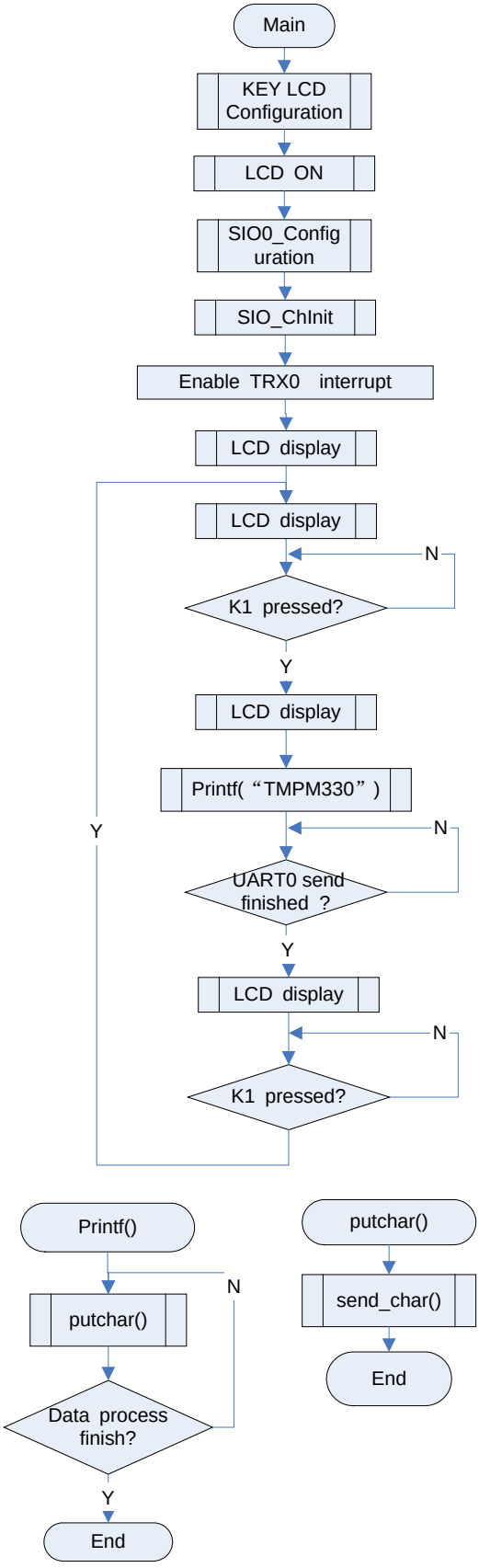
7-9-2 例: リターゲット

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

この例では以下を行います。

1. UART 設定と初期化
2. UART 送信制御
3. データ送信に UART0 の TX 割り込みを使用
4. UART0 に printf()関数をリターゲット

- **フローチャート:**


Main
KEY LCD Configuration
LCD ON
SIO0_Configuration
SIO_ChInit
Enable TRX0 interrupt
LCD display
LCD display
N
K1 pressed?
Y
LCD display
Printf(“TMPM330”)
N
UART0 send finished ?
Y
LCD display
N
K1 pressed?
Y
Printf()
putchar()
N
Data process finish?
Y
End
putchar()
send_char()
End

- **サンプルプログラムのコードと説明**

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

```
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 構造体を準備し、データを設定します。以下は設定例です。

```
UART_InitTypeDef myUART;

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

UART ペリフェラルドライバを使用して UART0 の許可と初期設定を行います。

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

上記設定後、UART0 の送信割り込みを許可します。

```
 NVIC_EnableIRQ(INTTX0_IRQn);
```

printf()を使用し、UART0 へ出力します。ここでは送信バッファに文字列を設定します。

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

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 (fSIO0_INT == CLEAR) {   /* if SIO0 INT disable, enable it */
      fSIO0_INT = SET;        /* set SIO0 INT flag */
      UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);
      NVIC_EnableIRQ(INTTX0_IRQn);
   }
```

```
    return ch;
}
```

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(UART0, gSIOTxBuffer[gSIORdIndex++]);    /* send data */
        fSIO0_INT = SET;        /* SIO0 INT is enable */
    } else {
        /* disable SIO0 INT *
        fSIO0_INT = CLEAR;
        NVIC_DisableIRQ(INTTX0_IRQn);
        fSIOTxOK = YES;
    }
    if (gSIORdIndex >= gSIOWrIndex) {   /* reset buffer index */
        gSIOWrIndex = CLEAR;
        gSIORdIndex = CLEAR;
    } else {
        /* do nothing */
    }
}
```

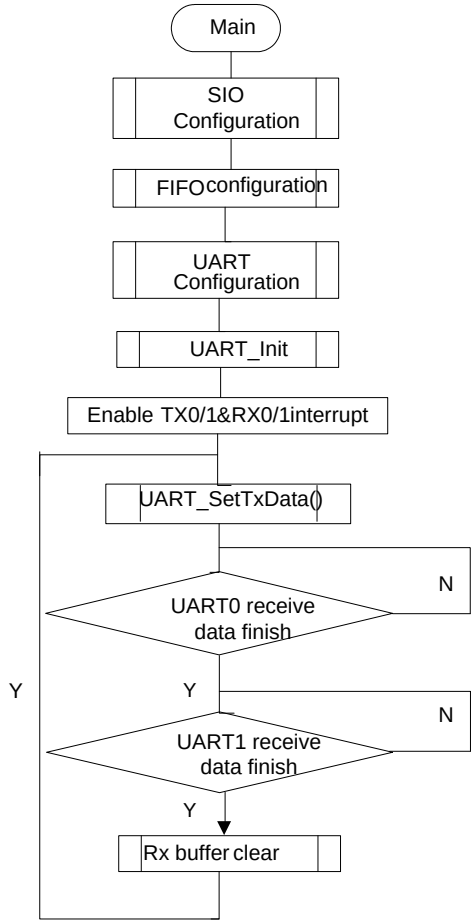
7-9-3 例: UART FIFO

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

この例では以下を行います。

1.UART と FIFO の初期設定
2.FIFO を使用した UART の送受信

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に GPIO の設定と UART の初期化を行います。
GPIO ドライバを使用して、GPIO を UART0 と UART1 に設定します。

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
   if (SCx == TSB_SC0) {
      TSB_PE->CR |= GPIO_BIT_1;
      TSB_PE->FR1 |= GPIO_BIT_1;
      TSB_PE->FR1 |= GPIO_BIT_0;
      TSB_PE->IE |= GPIO_BIT_0;
   } else if (SCx == TSB_SC1) {
      TSB_PE->CR |= GPIO_BIT_4;
      TSB_PE->FR1 |= GPIO_BIT_4;
      TSB_PE->FR1 |= GPIO_BIT_5;
      TSB_PE->IE |= GPIO_BIT_5;
   }
}
```

UART_InitTypeDef 構造体を用意し、すべてのメンバを設定します。

```
   UART_InitTypeDef myUART;

   /* configure SIO0 for reception */
   UART_Enable(UART_RETARGET);
```

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

UART のドライバを使用して、UART0/1 の各チャネルの許可と初期設定を行います。

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

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

FIFO の設定を行います。

```
   UART_RxFIFOByteSel(UART0,UART_RXFIFO_RXFLEVEL);
   UART_RxFIFOByteSel(UART1,UART_RXFIFO_RXFLEVEL);

   UART_TxFIFOINTCtrl(UART0,ENABLE);
   UART_TxFIFOINTCtrl(UART1,ENABLE);

   UART_RxFIFOINTCtrl(UART0,ENABLE);
   UART_RxFIFOINTCtrl(UART1,ENABLE);

   UART_TRxAutoDisable(UART0,UART_RXTXCNT_AUTODISABLE);
   UART_TRxAutoDisable(UART1,UART_RXTXCNT_AUTODISABLE);

   UART_FIFOConfig(UART0,ENABLE);
   UART_FIFOConfig(UART1,ENABLE);

   UART_RxFIFOFillLevel(UART0, UART_RXFIFO4B_FLEVLE_4_2B);
   UART_RxFIFOFillLevel(UART1, UART_RXFIFO4B_FLEVLE_4_2B);

   UART_RxFIFOINTSel(UART0,UART_RFIS_REACH_EXCEED_FLEVEL);
   UART_RxFIFOINTSel(UART1,UART_RFIS_REACH_EXCEED_FLEVEL);

   UART_RxFIFOClear(UART0);
   UART_RxFIFOClear(UART1);

   UART_TxFIFOFillLevel(UART0, UART_TXFIFO4B_FLEVLE_0_0B);
   UART_TxFIFOFillLevel(UART1, UART_TXFIFO4B_FLEVLE_0_0B);

   UART_TxFIFOINTSel(UART0,UART_TFIS_REACH_NOREACH_FLEVEL);
   UART_TxFIFOINTSel(UART1,UART_TFIS_REACH_NOREACH_FLEVEL);

   UART_TxFIFOClear(UART0);
   UART_TxFIFOClear(UART1);
```

上記設定を行い、その後 UART0/1 の各割り込みを許可します。

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

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

最後に UART0/1 の送信割り込みと受信割り込みの割り込み処理ルーチンを準備します。
以下は UART0 の送信割り込み処理ルーチンです。

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

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

以下は UART1 の送信割り込み処理ルーチンです。

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

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

以下は UART0 の受信割り込み処理ルーチンです。

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

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

以下は UART1 の受信割り込み処理ルーチンです。

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

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

7-9-4 例: SIO

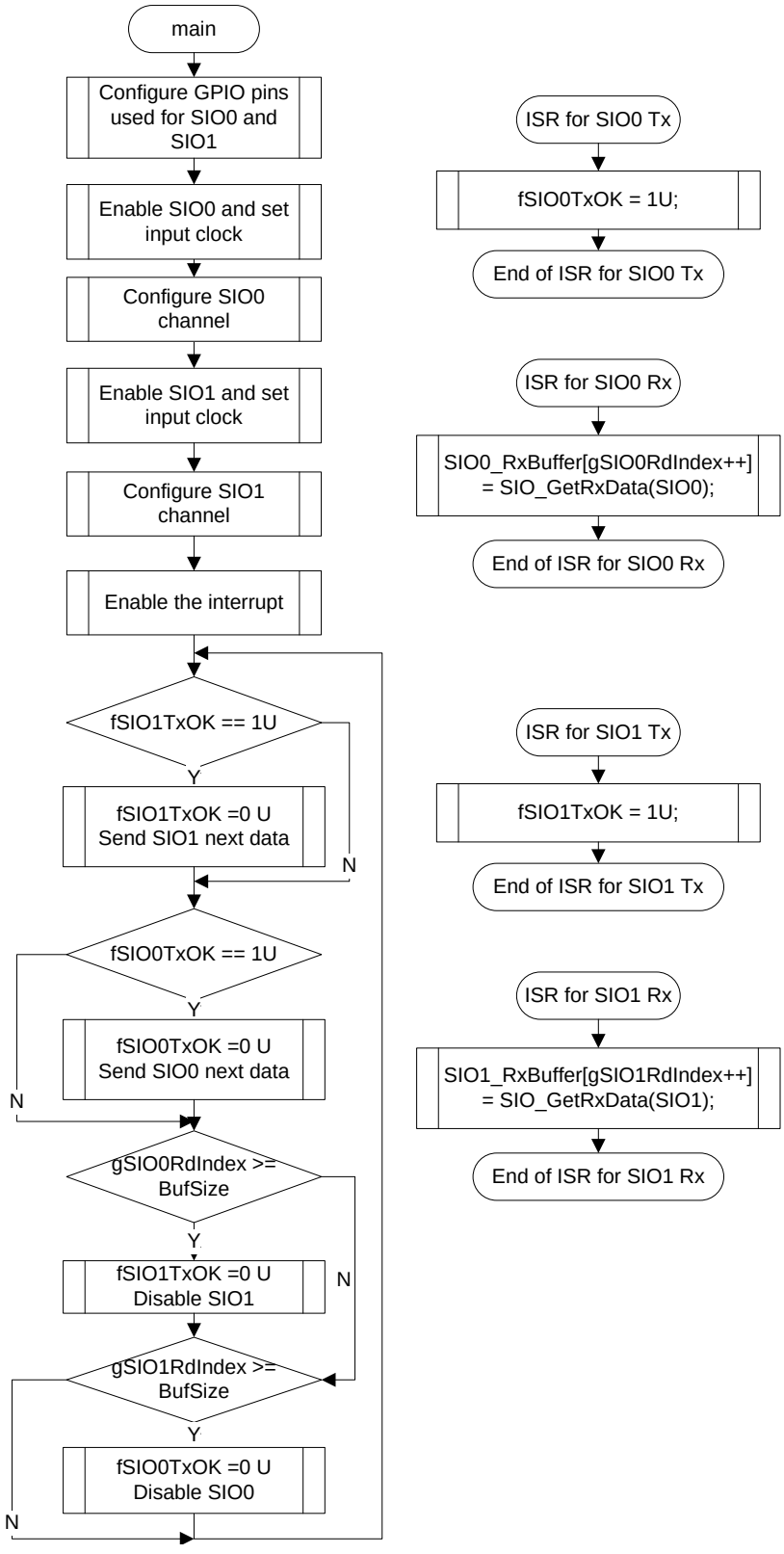
ペリフェラルドライバ(SIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. SIO 動作の基本設定

2.SIO0~SIO1 間のデータ転送
3.SIO の送受信割り込み

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、GPIO のペリフェラルドライバを使用し 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_1;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_T4;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;

SIO_Init(SIO0, SIO_CLK_BAUDRATE, &SIO0_Init);
```

SIO1 を有効にし、入力クロックの設定と SIO1 の初期化を行います。

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

/*initialize the SIO1 struct */
SIO1_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO1_Init.IntervalTime = SIO_SINT_TIME_SCLK_1;
SIO1_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO1_Init.TransferDir = SIO_LSB_FRIST;
SIO1_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO1_Init.DoubleBuffer = SIO_WBUF_ENABLE;

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

SIO の送信割り込みと受信割り込みを許可します。

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

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

すべての基本的な設定を行い、その後、データ転送処理に入ります。

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

```
            fSIO0TxOK = 0U;
            SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
        } else {
            /*Do Nothing */
        }

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

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

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

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

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

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

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

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

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

7-10 TMRB

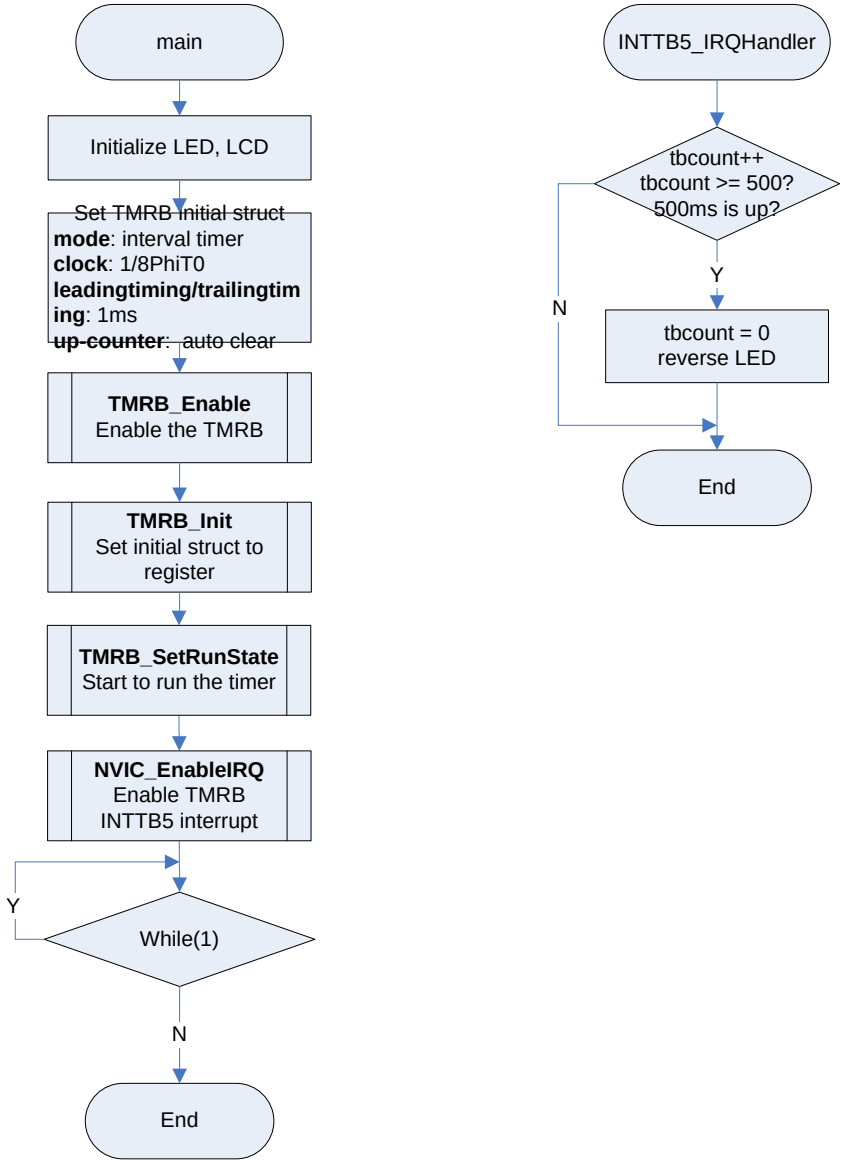
7-10-1 例: 汎用タイマ

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

この例では以下を行います。

1. TMRB5 の初期化
2. 1ms の汎用タイマ
3. 1Hz 周期の LED ON/OFF

- **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、STK ボードの LED を点灯します。

```
LED_Configuration();            /* LED initialize */
LCD_Configuration();            /* LCD initialize */
LCD_Light(ON);                  /* Turn on LCD backlight */
```

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、0x1388 です。(φT0=fsys=10MHz * PLL* = 40MHz, ftmrb = 1/8φT0 = 5MHz, Ttmrb = 0.2us, 1ms/0.2us = 5000 = 0x1388 (クロック設定についての詳細は CG 章を参照してください))

```
    TMRB_InitTypeDef myTMRB;

    myTMRB.Mode = TMRB_INTERVAL_TIMER;    /* internal timer */
    myTMRB.ClkDiv = TMRB_CLK_DIV_8;       /* 1/8PhiT0 */
    myTMRB.TrailingTiming = TMRB_1ms;                 /* trailingtiming is 1ms */
    myTMRB.UpCntCtrl = TMRB_AUTO_CLEAR;   /* up-counter auto clear */
    myTMRB.LeadingTiming = TMRB_1ms;                  /* leadingtiming 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。INTTB5 割り込み（1ms ごとにトリガ）を有効にします。最後に TMRB5 を動作させます。

```
    TMRB_Enable(TSB_TB5);                     /* enable the TMRB5 */
    TMRB_Init(TSB_TB5, &myTMRB);              /* initial the TMRB5 */
    TMRB_SetRunState(TSB_TB5, TMRB_RUN);  /* run TMRB5*/
    NVIC_EnableIRQ(INTTB5_IRQn);              /* enable INTTB5 interrupt */
```

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。

```
    if (tb5count >= 500U) {                   /* 500ms is up */
       tb5count = 0U;
       /* Reverse LED output */
       ledon = ~ledon;
       LED_Display(ledon);
    } else {
       /* do nothing */
    }
```

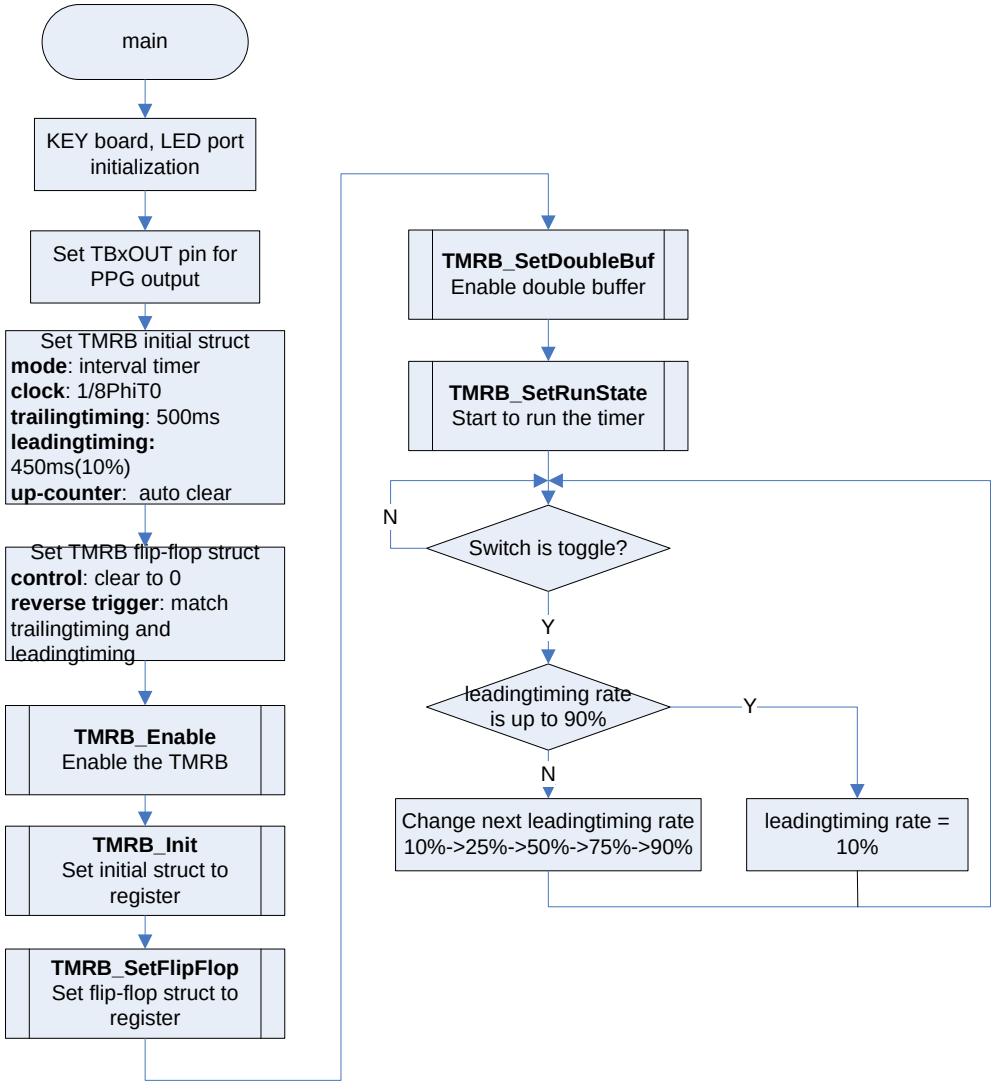
7-10-2 例: PPG 出力

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**

まず、GPIO を LED 用、LCD 用に設定し、PI0 を PPG 出力用に TB0OUT に設定します。

```
    LED_Configuration();
    LCD_Configuration();
    LCD_Light(ON);

    GPIO_WriteData(GPIO_PG, tmp);

    /* Set PI0 as TB0OUT for PPG output */
    GPIO_SetOutput(GPIO_PI, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PI, GPIO_FUNC_REG_1, GPIO_BIT_0);
    GPIO_SetPullUp(GPIO_PI, GPIO_BIT_0, ENABLE);
```

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、TMRB7TIME マクロを定義してあります。このマクロは 0x09C4 です。(φT0 = fsys = fc = 10MHz * PLL * 1/4 = 20MHz, ftmrb = 1/8 φT0 = 2.5MHz, Ttmrb = 0.4us, 1000us/0.4us = 2500 = 0x09C4 (クロック設定の詳細は CG 章を参照してください))

```
    TMRB_InitTypeDef myTMRB;
```

```
    myTMRB.Mode = TMRB_INTERVAL_TIMER;    /* internal timer */
    myTMRB.ClkDiv = TMRB_CLK_DIV_8;       /* 1/8PhiT0 */
    myTMRB.TrailingTiming = TMRB0TIME;            /* trailingtiming is 500us
*/
    myTMRB.UpCntCtrl = TMRB_AUTO_CLEAR;   /* up-counter auto clear */
    myTMRB.LeadingTiming = LeadingTiming[0];                    /*
leadingtiming, initial value 10% */
```

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

```
    PPGFFInital.FlipflopCtrl = TMRB_FLIPFLOP_CLEAR;
    PPGFFInital.FlipflopReverseTrg=TMRB_FLIPFLOP_MATCH_TRAILINGTIMING|
    TMRB_FLIPFLOP_MATCH_LEADINGTIMING;
```

TMRB モジュールを許可し、指定レジスタに初期化構造体、フリップフロップ構造体を設定します。ダブルバッファを許可し、キャプチャ機能を禁止にします。最後に、TMRB を動作させます。

```
    TMRB_Enable(TSB_TB0);
    TMRB_Init(TSB_TB0, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB0, &PPGFFInital);
    /* enable double buffer */
    TMRB_SetDoubleBuf(TSB_TB0,ENABLE);
    TMRB_SetRunState(TSB_TB0, TMRB_RUN);
```

スイッチ状態の変化を待ちます。

```
    do {            /* Wait for key released */
       key_info = GPIO_ReadData(KEY_PORT) & KEYRELEASE;
      } while (key_info != KEYRELEASE);
```

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
leadingtiming rate */
```

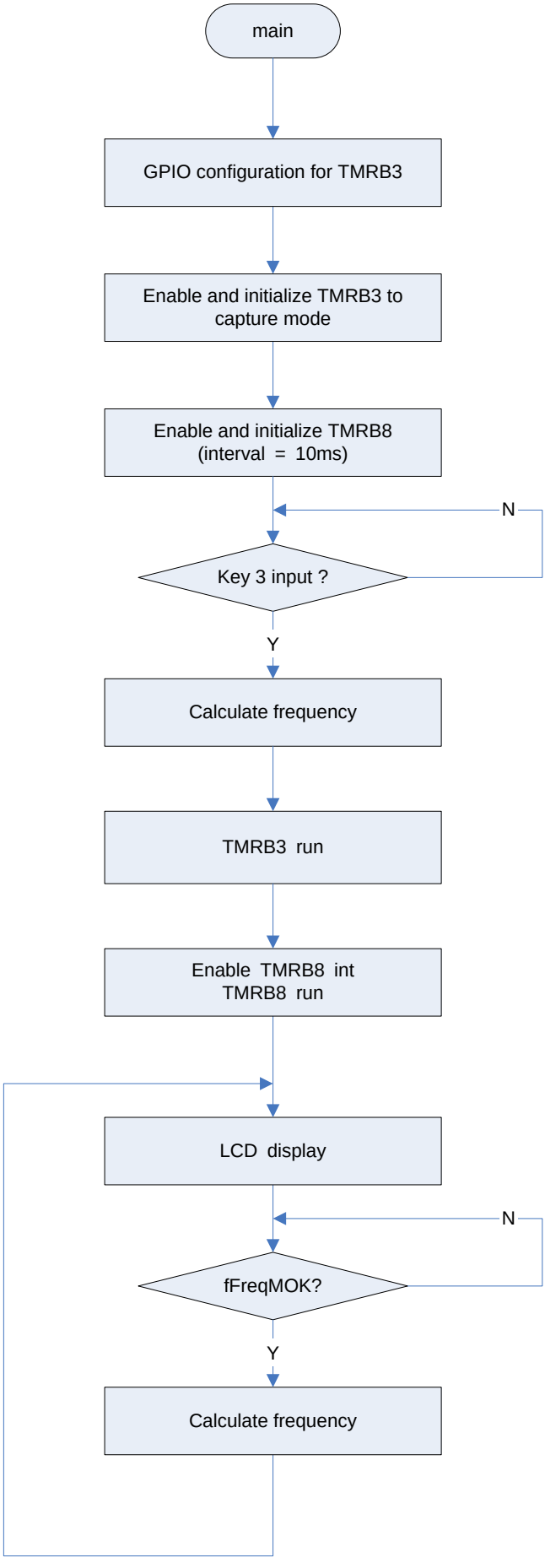
7-10-3 例: 周波数測定

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. TMRB の初期化
2. 周波数測定

- **フローチャート:**


main
GPIO configuration for TMRB3
Enable and initialize TMRB3 to capture mode
Enable and initialize TMRB8 (interval = 10ms)
N
Key 3 input ?
Y
Calculate frequency
TMRB3 run
Enable TMRB8 int
TMRB8 run
LCD display
N
fFreqMOK?
Y
Calculate frequency

- **サンプルプログラムのコードと説明**

まず GPIO を TB3IN0 に設定します。

```
GPIO_SetInput(GPIO_PH, GPIO_BIT_6);
GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_1, GPIO_BIT_6);
GPIO_SetPullUp(GPIO_PH, GPIO_BIT_6, DISABLE);
GPIO_SetInputEnableReg(GPIO_PH, GPIO_BIT_6, ENABLE);
```

TMRB を許可し、TMRB3 と TMRB8 の初期設定を行います。

```
myTMRB.Mode = TMRB_EVENT_CNT;
myTMRB.TrailingTiming = 0U;          /* Specific value depends on system clock */
myTMRB.UpCntCtrl = TMRB_FREE_RUN;
myTMRB.LeadingTiming = 0U;           /* Specific value depends on system clock */

TMRB_Enable(TSB_TB3);
TMRB_Init(TSB_TB3, &myTMRB);

/* enable capture, Capture timing is both edges of TBnOUT */
/* UC clear/disable, source clock is TBnIN0 pin input */
TMRB_SetCaptureTiming(TSB_TB3, TMRB_CAPTURE_OUTPUT_EDGE);
TMRB_ExecuteSWCapture(TSB_TB3);

/* enable double buffer, individual, IDLE stop */
TMRB_SetIdleMode(TSB_TB3, DISABLE);
TMRB_SetSyncMode(TSB_TB3, DISABLE);
TMRB_SetDoubleBuf(TSB_TB3, ENABLE);

/* no flip-flop output */
myTMRBFFInit.FlipflopCtrl = TMRB_FLIPFLOP_INVERT;
myTMRBFFInit.FlipflopReverseTrg = TMRB_DISABLE_FLIPFLOP;
TMRB_SetFlipFlop(TSB_TB3, &myTMRBFFInit);

/* no capture, UC clear/enable, source clock is T4 */
myTMRB.Mode = TMRB_INTERVAL_TIMER;
myTMRB.ClkDiv = TMRB_CLK_DIV_8;
myTMRB.TrailingTiming = TMRB8TIME;   /* TMRB8 for frequency measure: 10ms */
myTMRB.UpCntCtrl = TMRB_AUTO_CLEAR;
myTMRB.LeadingTiming = 0U;

TMRB_Enable(TSB_TB8);
TMRB_Init(TSB_TB8, &myTMRB);

/* set TBnRG1 as flip-flop output trigger */
myTMRBFFInit.FlipflopReverseTrg = TMRB_FLIPFLOP_MATCH_TRAILINGTIMING;
myTMRBFFInit.FlipflopCtrl = TMRB_FLIPFLOP_INVERT;
TMRB_SetFlipFlop(TSB_TB8, &myTMRBFFInit);
```

TMRB3 と TMRB8 を開始します。

```
    case K3:              /* Frequency Measurement */
      do {              /* wait for key released */
        key_info = GPIO_ReadData(KEY_PORT) & KEYRELEASE;
      } while (key_info != KEYRELEASE);
      NVIC_EnableIRQ(INTTB8_IRQn);
      TMRB_SetRunState(TSB_TB3, TMRB_RUN);       /* start TMRB3 */
      TMRB_SetRunState(TSB_TB8, TMRB_RUN);       /* start TMRB8 */
```

周波数測定を何度か行い、周波数を求めます。

```
void INTTB8_IRQHandler(void)
```

```
{
    static uint8_t tb8count = 0U;
    TMRB_INTFactor TMRB_Status;
    uint16_t eTMRB_CP0;
    uint16_t eTMRB_CP1;

    TMRB_Status = TMRB_GetINTFactor(TSB_TB3);
    if (!TMRB_Status.Bit.OverFlow) {    /* no timer overflow */
        /*
          frequency = number of pulses / interval
          number of pulses = |CP0_TMRB3 - CP1_TMRB3|
          interval = 10ms
         */
        eTMRB_CP0 = TMRB_GetCaptureValue(TSB_TB3, TMRB_CAPTURE_0);
        eTMRB_CP1 = TMRB_GetCaptureValue(TSB_TB3, TMRB_CAPTURE_1);
        if (eTMRB_CP0 >= eTMRB_CP1) {
            gFreq[tb8count] = eTMRB_CP0 - eTMRB_CP1;
        } else {
            gFreq[tb8count] = eTMRB_CP1 - eTMRB_CP0;
        }
        gFreq[tb8count] *= 1000U;
        gFreq[tb8count] /= 10U;

        tb8count++;
        if (tb8count >= TIMES) {
            fFreqMOK = true;
            tb8count = 0U;
            NVIC_DisableIRQ(INTTB8_IRQn);
        } else {
            /* do nothing */
        }
    } else {
        /* do nothing */
    }
}
```

7-11 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();
```