TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM341)

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M341UE-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。

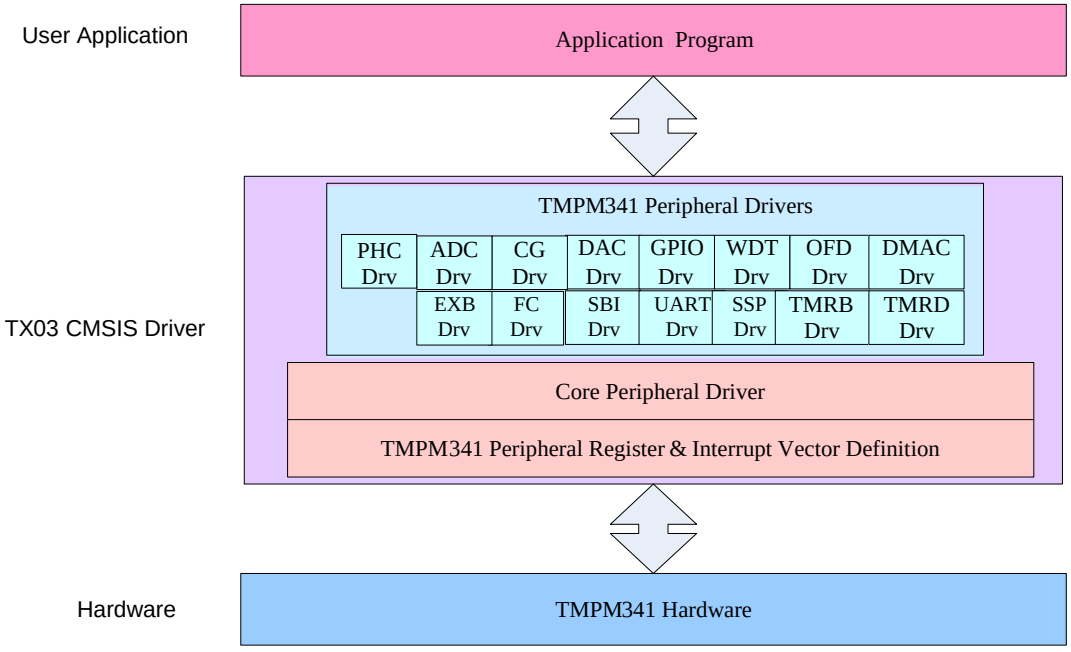
目次

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	8 逓倍
低消費電力モード	-	STOP1 モード
SysTick	-	未使用
ウォッチドッグタイマ (WDT)	-	使用(WDT サンプル、OFD サンプル)

機能	ブロック/ユニット/チャネル	使用／未使用
外部割込み (INT)	INT1	使用
	上記以外のチャネル	未使用
DMAC	-	使用(DMAC サンプル)
シリアルチャネル (SIO/UART)	SIO0	使用(UART & DMAC Tx サンプル、UART りターゲットサンプル、SIO サンプル、UART_FIFO サンプル)
	SIO1	使用(SIO サンプル、UART_FIFO サンプル)
	SIO2	使用(UART & DMAC Rx サンプル)
	上記以外のチャネル	未使用
同期式シリアルインタフェース (SSP)	SSP	使用(SSP セルフループバックサンプル)
シリアルバスインタフェース(SBI)	SBI0	使用(SBI サンプル: マスタ Tx)
	SBI1	使用(SBI サンプル: スレーブ Rx)
16 ビットタイマ/イベントカウンタ(TMRB)	TMRB0	使用(TMRB: 汎用タイマサンプル)
	TMRB7	使用(TMRB: PPG 波形出力サンプル)
	上記以外のチャネル	未使用
周波数検知回路(OFD)	-	使用(Startup サンプル)
D/A コンバータ(DAC)	DA0	使用(DAC: DAC0 波形出力サンプル)
	DA1	未使用
高分解能 16 ビットタイマ(TMRD)	TMRD0	使用(TMRD: インターバルタイマ、連動 PPG 波形出力サンプル)
	TMRD1	使用(TMRD: 連動 PPG 波形出力サンプル)
10ビットA/Dコンバータ (ADC)	AIN14	使用(ADC データリードサンプル)
	上記以外のチャネル	未使用
外部バスインタフェース (EXB)	CS0	未使用
	CS1	使用(SRAM リード/ライトサンプル)
2 相パルス入力カウンタ(PHCNT)	PHCNT0	使用(PHC カウンタサンプル)

4 端子用途

本サンプルプログラムは、SBIとSSPを除き、IAR TMPM341-SKボードを用いてテストされています。以下に、端子用途を説明します。

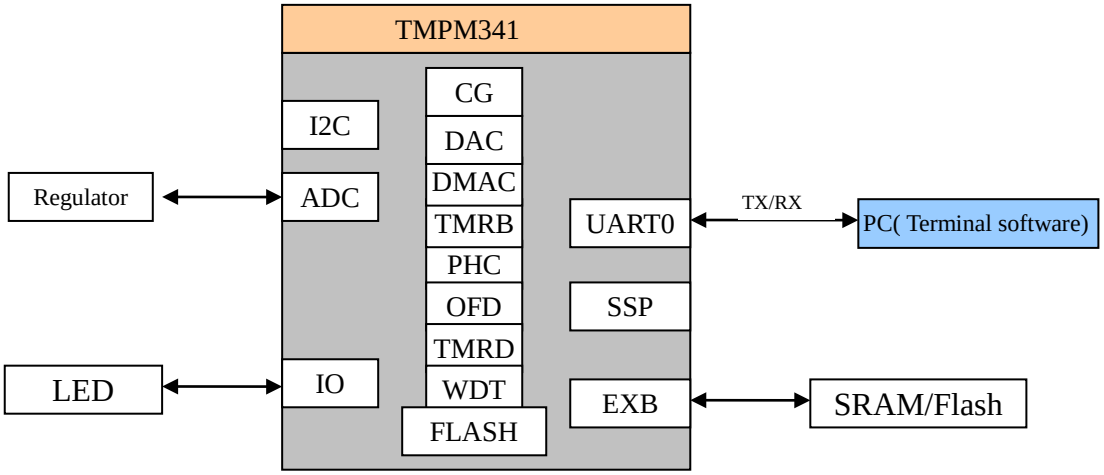
No	Name	Usage
A1	PK7,AIN15	未使用
A2	PK4,AIN12	未使用
A3	PJ7,AIN07	未使用
A4	PJ3,AIN03,PHC1IN1	未使用
A5	DVDD3A	+3V
A6	NMI	未使用
A7	X1/ECLKIN	未使用
A8	DVSSC	GND
A9	X2	未使用
A10	P15,TCK/SWCLK	未使用
A11	PH5,TRACEDATA3	未使用
B1	PK6,AIN14	AIN14
B2	PK5,AIN13	未使用
B3	PK0,AIN08,TB1IN0	未使用
B4	PJ4,AIN04,PHC2IN0	未使用
B5	PJ0,AIN00,PHC0IN0	PHCNT0入力
B6	RESET	RESET
B7	PH3,PHC3IN0	未使用
B8	RVSS	GND
B9	RVDD3	+3V
B10	PI4,TDI	未使用
B11	PH6,TACEDATA2	未使用
C1	AVDD3	+3V
C2	AVREFH	未使用
C3	PK1,AIN09,INTA,TB1IN1	未使用
C4	PJ5,AIN05,PHC2IN1	未使用
C5	PJ1,AIN05,PHC2IN1	未使用
C6	MODE	GND
C7	PH4,PHC3IN1,TB5OUT	未使用
C8	PH0,TXD4	未使用
C9	PI7,TDO/SWDIO	未使用
C10	PI6,TMS/SWDIO	未使用
C11	DVDD3A	+3V
D1	AVSS	GND
D2	AVREFL	未使用

No	Name	Usage
D3	PK2,AIN10,TB6IN0	未使用
D4	PJ6,AIN06,TB0IN0	未使用
D5	PJ2,AIN02,PHC1IN0	未使用
D6	PG7,INT1,TB9IN1	未使用
D7	PG6,SCLK3,TB9IN0	未使用
D8	PH1,RXD4	未使用
D9	PI2,TRACECLK	未使用
D10	PI3,TRST	未使用
D11	DVSSA	GND
E1	DVSSB	GND
E2	DVDD3B	+3V
E3	PK3,AIN11,INTB, TB6IN1	未使用
E4	DA0	アナログ出力端子
E5	DA1	未使用
E8	PH2,SCLK4,CTS4	未使用
E9	PG5,RXD3,TB8IN1	未使用
E10	PI1,TRACEDATA0	未使用
E11	DVSSB	GND
F1	PA0,D0/AD0	AD0
F2	PF0,BOOT,TB6OUT	未使用
F3	PF1,RD	未使用
F4	INTLV	未使用
F8	PG4,TXD3,TB8IN0	未使用
F9	PG3,INT0	未使用
F10	PI0,TRACEDATA1	未使用
G1	PA1,D1/AD1	AD1
G2	PA2,D2/AD2	AD2
G3	PF3,BELL	BELL
G4	PF2,WR	WR
G8	PPG2,SCK0	未使用
G9	PG1,SIO/SCL0,TB7IN1	未使用
G10	PD7,A15,SPFSS,SCOUT	未使用
G11	PD6,A14,SPCLK	未使用
H1	PA3,D3/AD3	AD3
H2	PA4,D4/AD4	AD4
H3	PF4,BELH,INT6,TB5IN0	BELH
H4	PF5,CS1,INT7,TB5IN1	CS1
H5	PC1,A1,RXD1,TB2IN1	SIO1 RXD1
H6	PC3,A3,INT2,TB1OUT	未使用
H7	PC6,A6,SCLK2,TB4IN0	未使用
H8	PC7,A7,INT3,TB4IN1	未使用

No	Name	Usage
H9	PG0,SO0/SDA0	未使用
H10	PD5,A13,SPDI	未使用
H11	PD4,A12,,SPD0	未使用
J1	PA5, D5/AD5	AD5
J2	PA6, D6/AD6	AD6
J3	PF6, CS0	CS0
J4	PF7, ALE	ALE
J5	PC0, A0, TXD1, TB2IN0	SIO1 TXD1
J6	PC2, A2, SCLK1, TB0OUT, CTS1	SIO1 SCLK1
J7	PC5, A5, RXD2, TB3IN1	未使用
J8	PE0, TXD0, A16	A16 / SIO0 TXD0
J9	PE2, SCLK0, A18, TB2OUT, CTS0	LED出力/ A18/ SIO0 SCLK0
J10	PD3, A11, INT4, ADTRG	未使用
J11	PD2, A10, SCK1, TB9OUT	未使用
K1	PA7, D7/AD7	AD7
K2	PB1, D9/AD9, A1	AD9
K3	PB3, D11/AD11, A3	AD11
K4	PB5, D13/AD13, A5	AD13
K5	PB7, D15/AD15, A7	AD15
K6	DVDD3B	未使用
K7	PC4, A4, TXD2, TB3IN0	未使用
K8	PE1, RXD0, A17	A17 / SIO0 RXD0
K9	PE3, INT5, A19, TB3OUT	A19
K10	PD1, A9, SI1/SCL1, TB8OUT	未使用
K11	PD0, A8, SO1/SDA1, TB7OUT	PPG波形出力
L1	BSC	未使用
L2	PB0, D8/AD8, A0	AD8
L3	PB2, D10/AD10, A2	AD10
L4	PB4, D12/AD12, A4	AD12
L5	PB6, D14/AD14, A6	AD14
L6	DVSSB	GND
L7	PE4, A20, TD0OUT0	TMRD PPGパルスA0出力/ A20
L8	PE5, A21, TD0OUT1	A21
L9	PE6, A22, TD1OUT0	TMRD PPGパルスB0出力/ A22
L10	PE7, A23, TD1OUT1	A23
L11	FTEST3	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 TMPM341-SK ボード

2. 開発ツール:
    - IAR:
        2) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
        3) IDE: IAR Embedded workbench 6.30.1 version
    - KEIL:
        1) U-Link: Realview ULINK2
        2) IDE: KEIL uVision 4.21

6 機能説明

6-1 動作モード

本デバイスには 4 つの動作モードがあります: NORMAL, IDLE, STOP1, STOP2 モード
IDLE と STOP1、STOP2 モードは低消費電力モードです。
低消費電力モードへ移行するには、CGSTBYCR<STBY2:0>にて IDLE、STOP1、STOP2 のいずれかのモードを選択し、WFI (Wait For Interrupt) 命令を実行します。

- **NORMAL モード:**

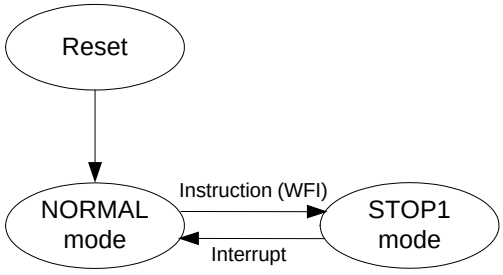
このモードは、CPU コアおよび周辺ハードウェアを高速クロックで動作させるモードです。リセット解除後は NORMAL モードになります。

補足: STOP1 モードのサンプルが含まれています。

- **STOP1 モード:**

STOP1 モードは、内蔵発振器も含めてすべての内蔵回路が停止するモードです。STOP1 モードが解除されると内蔵発振器が発振を開始し、NORMAL モードへ復帰します。CGSTBYCR<DRVE>を設定することにより、STOP1 モード中は端子のドライブ状態を保持することができます。
STOP1 モードの解除要因により、STOP1 モードから NORMAL モードへ復帰する場合、ウォーミングアップカウンタは自動的に起動します。この場合のウォーミングアップ時間は内蔵 Flash の安定時間として、450μs 以上を設定してください。
リセットで NORMAL モードへ復帰する場合、ウォーミングアップは行われませんので、発振動作が安定するまでのリセット信号を有効に保ってください。

補足: STOP1 モードのサンプルが含まれています。



6-2 ADC

6-2-1 ADC データリード

PK6/AIN14 に接続されたポテンションメータの値を変更します。測定された電圧値は標準出カラ

イブラリ(stdout)を経由で出力します。stdout は UART にリターゲットされますので、PC と UART0 を接続してください。AD 変換結果は、ターミナル出力ソフトに表示されます。

6-3 CG

6-3-1 Power モード変更

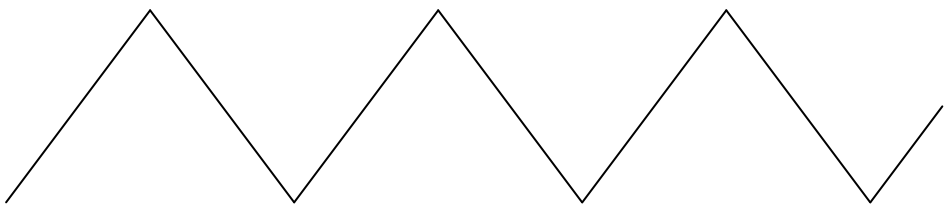
CPU 動作モードを変更します。NORMAL モードと STOP1 モードの 2 つのモードを使用します。

現在のモード	アクション (Key)	動作	ターミナル表示	LED 表示
NORMAL	SW1を押す (Lowアクティブ)	NORMAL→ STOP1	Now, Going to Stop1	LED0 オフ
STOP1	PG3(INT0)-VCC(3.3V)ショート	STOP1 →NORMAL	Wakeup from Stop1	LED0 オン

6-4 DAC

6-4-1 のこぎり波形出力

DAx の出力コードを 0 から 1023 へ間隔固定で変更します。出力コードが 1023 に達すると、DAxの出力コードを 1023 から 0 へ、間隔固定にて変更します。波形は DAx端子より、下記のように出力されます。



6-5 DMAC

6-5-1 メモリから周辺回路

UART0 から UART2 へのデータ転送を行う処理が実装されています。“TOSHIBA”の文字列データ

を UART0 から UART2 へ送信されます。
DMAC により、RAM から UART0 データレジスタにデータが送信され、文字列の各文字データは UART0 経由で送信され、UART2 で受信します。UART2 のデータ受信後、データは文字配列に保存されます。

デバッガの変数ウィンドウに文字配列変数を登録して、受信されたデータを確認します。

補足: サンプルソフトウェアを使用するには、TMPM341-SK ボードの設定変更を行います。
UART0(TX)と UART2(RX)を接続します。

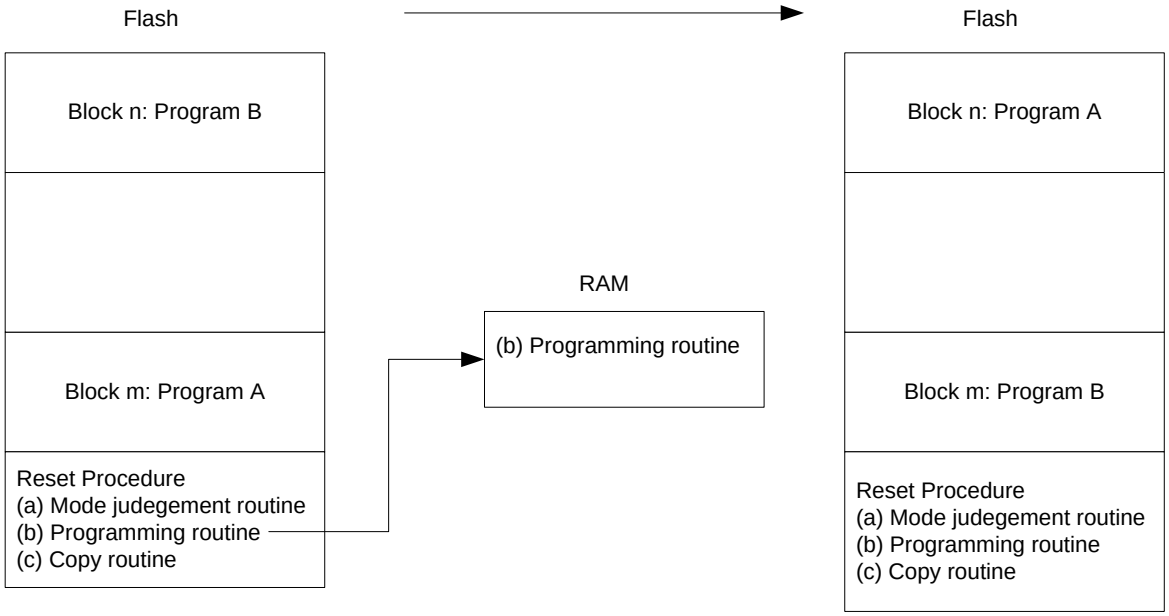
6-6 EXB

6-6-1 SRAM リード/ライト

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

6-7 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 0 点灯, B: LED 1 点灯)
初期状態は、プログラム A が最初に動作します。
ーSW1 キーはリセット動作のモード判定に使います。
SW1 キーが押されていない場合 → Normal モードです。
SW1 キーが押された場合 → ユーザブートモードです。



動作シーケンス

(1) 電源投入

SW1 を OFF した状態で電源投入またはリセット端子を ON してください。まずプログラム A が実行され、LED2 が点灯します。

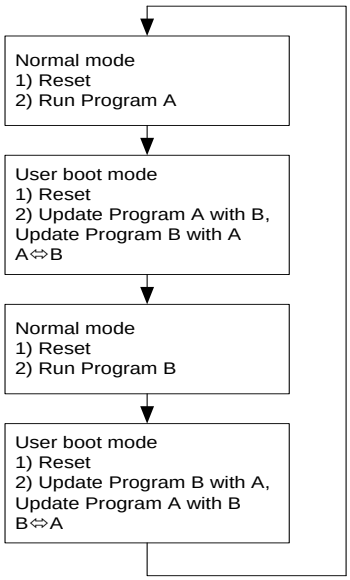
(2) SW1 を ON している間にリセット端子を ON してください。

スワップ処理:

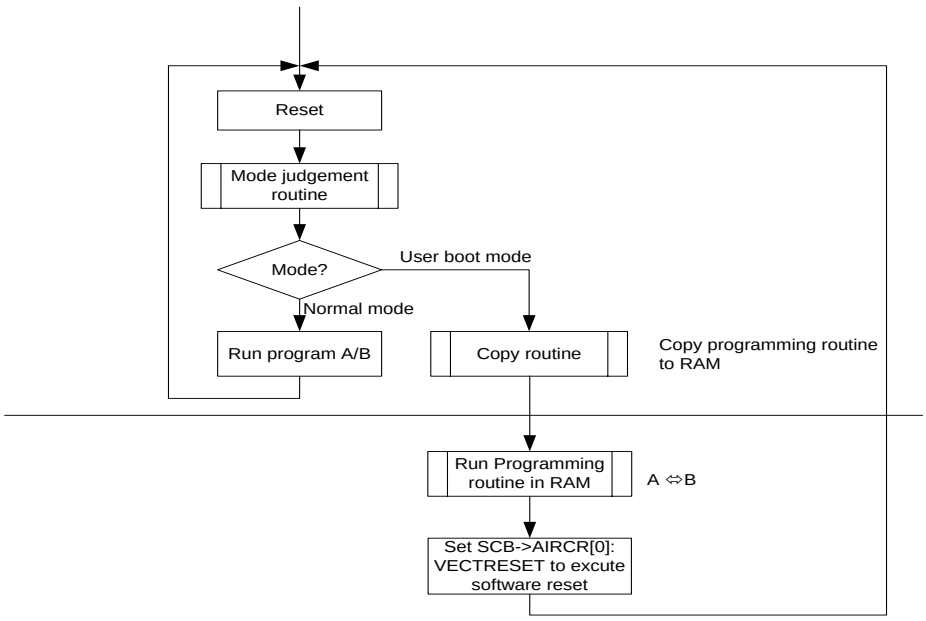
LED2 が 8 回点滅したらユーザブートモードに入り、LED2 を 4 回点滅したら Flash API の RAM 転送が完了します。その後 LED2 が 2 回点滅したらプログラム A と B のスワップが完了します。

スワップ成功処理: LED2 が 12 回点滅します。

(3) スワップ完了後、SW1 を OFF すると、システムリセットを行い、プログラム B が動作します。



・サンプルプログラムのフローチャート



サンプルプログラムがユーザブートモードになると、LED は次のような動作をします。

LED2 が 8 回点滅　　(ユーザブートモード)

RAM 転送中、または Flash 書込み中、LED は現在の状態を示します。

LED 表示例:
LED2 が 4 回点滅:　　(RAM 転送中)
LED2 が 2 回点滅:　　(プログラム A⇔B スワップ中)
LED2 が 12 回点滅:　　(スワップ完了、リセット待ち)

6-8　GPIO

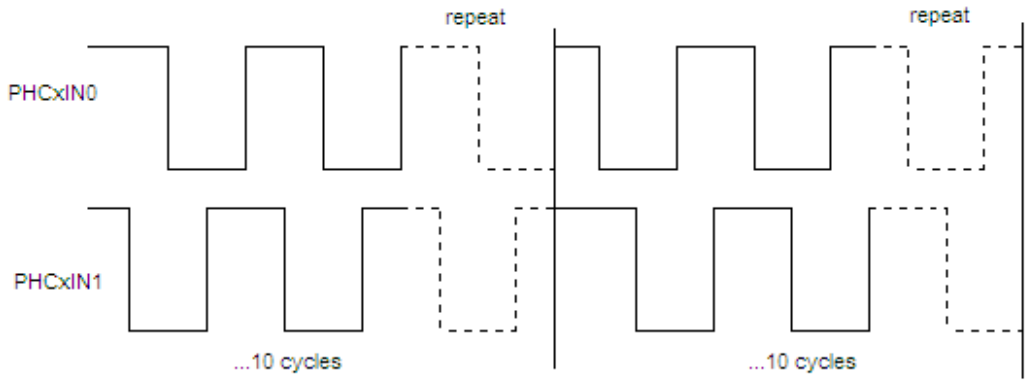
ペリフェラルドライバ(GPIO)を使用した簡単なサンプルプログラムです。ポートEの下位4ビット(PE0～PE3をLEDポートとして使用します。

6-9　OFD

ペリフェラルドライバ(OFD)を使用した簡単なプログラムです。
クロックが OFD 検出周波数帯を超えると、OFD は I/O 用のリセットを生成します。その後ソフトウェアがリセットされ、OFD リセットフラグがセットされます。このリセットフラグが検出されると、OFD はディセーブルとなり LED2 を点滅します。

6-10 PHC

PHC のドライバを使用して、4 逓倍モードを実現するサンプルプログラムです。
GPIO を使用して、2 相入力 PHC0IN0 および PHC0IN1 を同時に入力します。
入力波形は以下のような図になります。



アップ/ダウンカウンターは、2 相パルスの状態が変化するごとに増加/減少します。
カウンタは増加に 40 回、減少に 40 回かかります。
カウンタの結果は、デバッガの RAM 変数 count_result[] により確認できます。
count_result[0] = 0x7FFE,
count_result[1] = 0x7FFD,
count_result[2] = 0x7FFC
…
count_result[38] = 0x7FD8,
count_result[39] = 0x7FD7,
count_result[40] = 0x7FD8,
count_result[41] = 0x7FD9,
count_result[42] = 0x7FDA
…
count_result[79] = 0x7FFF.

補足:本サンプルソフトを実行するためには、以下の接続を行った TMPM341 評価ボードが必要です。

PJ0 (PHC0IN0) 端子と PJ2 (出力) 端子を接続します。
PJ1 (PHC0IN0) 端子と PJ3 (出力) 端子を接続します。

6-11 SBI

ペリフェラルドライバの I2C を使用し、I2C バスのスレーブモード処理を行うサンプルプログラムです。

評価ボードの 2 本の I2C バス (SBI0 & SBI1) を接続します。
片方は I2C バスのマスタモードで動作し、片方は I2C バスのスレーブモードで動作します。
I2C 割り込みを使用して I2C バスのリード/ライトを行います。
I2C のスレーブアドレス: 0xB0.
I2C スレーブ (SBI1) は I2C マスタ (SBI0)から “TOSHIBA”を受信します。
受信結果はデバッガにて RAM 変数 gI2CrxData[]にて確認できます。

補足:本サンプルソフトを実行するためには、以下の接続を行った TMPM341 評価ボードが必要です。

PG1 (SCL0) 端子と PD1 (SCL1) 端子を接続します。
PG0 (SDA0) 端子と PD0 (SDA1) 端子を接続します。

6-12 SIO/UART

6-12-1 リターゲット

この例では、C 言語の標準入出力ライブラリの stdin、stdout のリターゲットを行います。
stdin、stdout を共に UART0 へ設定します。アプリケーションから printf()と getchar() 関数を用いて、シリアルポートからデータを入出力します。

6-12-2 UART FIFO

FIFO を UART0 から"TMPM3411"というデータを使用して UART1 へ送信し、同時に FIFO を UART1 から"TMPM3412"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=”TMPM3412” と RxBuffer1="TMPM3411"となった場合にブレークポイントで停止します。

6-13 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)

6-14 SSP

6-14-1 SSP セルフループバック

データを送信し、その後、受信データを確認します。受信データが送信したものと同一かどうかの確認を行います。

補足: SD カードソケットは何も入っていない状態にしてください。

6-15 TMRB

6-15-1 汎用タイマ

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

6-15-2 PPG 出力

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

6-16 TMRD

6-16-1 インターバルタイマ

TMRD0 の 16 ビットインターバル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。
時間周期は 1s です。
このタイマを使い 1s(500ms でオン、500ms でオフ)間隔で LED 点滅を行います。

6-16-2 連動 PPG 出力

TMRD0 を使用して連動 PPG 出力を行うサンプルプログラムです。
PPG 周期は 500us で、デューティは 50% (High 幅と Low 幅がそれぞれ 250us です)
位相 A は位相 B より早く設定し、位相遅延量(θ)は 120 度です。
位相 A0 と B0 は PE4/TD0OUT0 と PE6/TD1OUT0 をオシロスコープで確認することができます。

6-17 WDT

リセットが行われるとウォッチドッグタイマは有効となるので、ウォッチドッグタイマを使用しない場

合は無効にしてください。ウォッチドッグタイマは、高周波クロックが停止している場合、使用できません。下記の動作モードへ移行する前に、ウォッチドッグタイマを無効にしてください。IDLEモードでは、ウォッチドッグタイマの動作はWDMOD<I2WDT> 設定に依存します。

ドライバ使用方法ステップ:

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

   DEMO1:

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

   DEMO2:

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

7 ソフトウェア

本ソフトウェアは、TMPM341 MCU の主要機能を IAR TMPM341-SK ボード上で動作確認するためのサンプルプログラムです。

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

ワークスペース構造とプロジェクト名は、以下の通りです:

IAR EWARM:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm341_wdt_int.c
        ├─TX03_CMSIS
        │   │   system_TMPM341.c
        │   │   system_TMPM341.h
        │   │   TMPM341.h
        │   │
        │   └─startup
        │         startup_TMPM341.s
        │
        ├─TX03_Periph_Driver
        │   ├─inc
        │   │   tmpm341_wdt.h
        │   │   tmpm341_gpio.h
        │   │   tx03_common.h
        │   │
        │   └─src
        │         tmpm341_wdt.c
        │         tmpm341_gpio.c
        └─TMPM341-SK
                led.c
                led.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm341_wdt_int.c
        │
```

```
├─TX03_CMSIS
│      system_TMPM341.c
│      startup_TMPM341.s
│
├─TX03_Periph_Driver
│      tmpm341_wdt.c
│      tmpm341_gpio.c
│
└─TMPM341-SK
        led.c
```

7-1 ADC

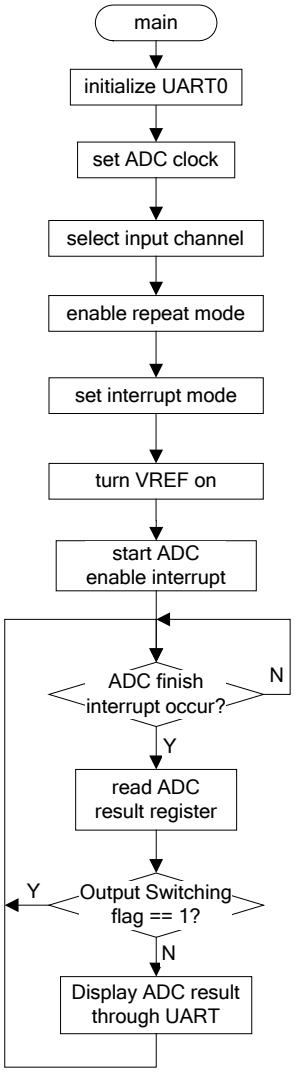
7-1-1 例: ADC データリード

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

以下の例が含まれます:

1. ADC 設定と初期化
2. チャネル固定リピートモードによる AD 変換を開始し、AD 変換結果を読み出します。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず ADC のクロック設定を行い、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 ON します。

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

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

/* select ADC input channel */
ADC_SetInputChannel(ADC_AN_14);

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

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

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

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

AD 変換を開始し、INTAD を許可します。

```
    ADC_Start();

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

AD 変換を行い、その後、INTAD を待ちます。INTAD の発生後、ADC_Display()をコールします。

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

ADC_Display()では、AD 変換結果を確認するため、ADREG をリードします。

```
    ADC_Result result = ADC_GetConvertResult(ADC_REG_00);

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

7-2 CG

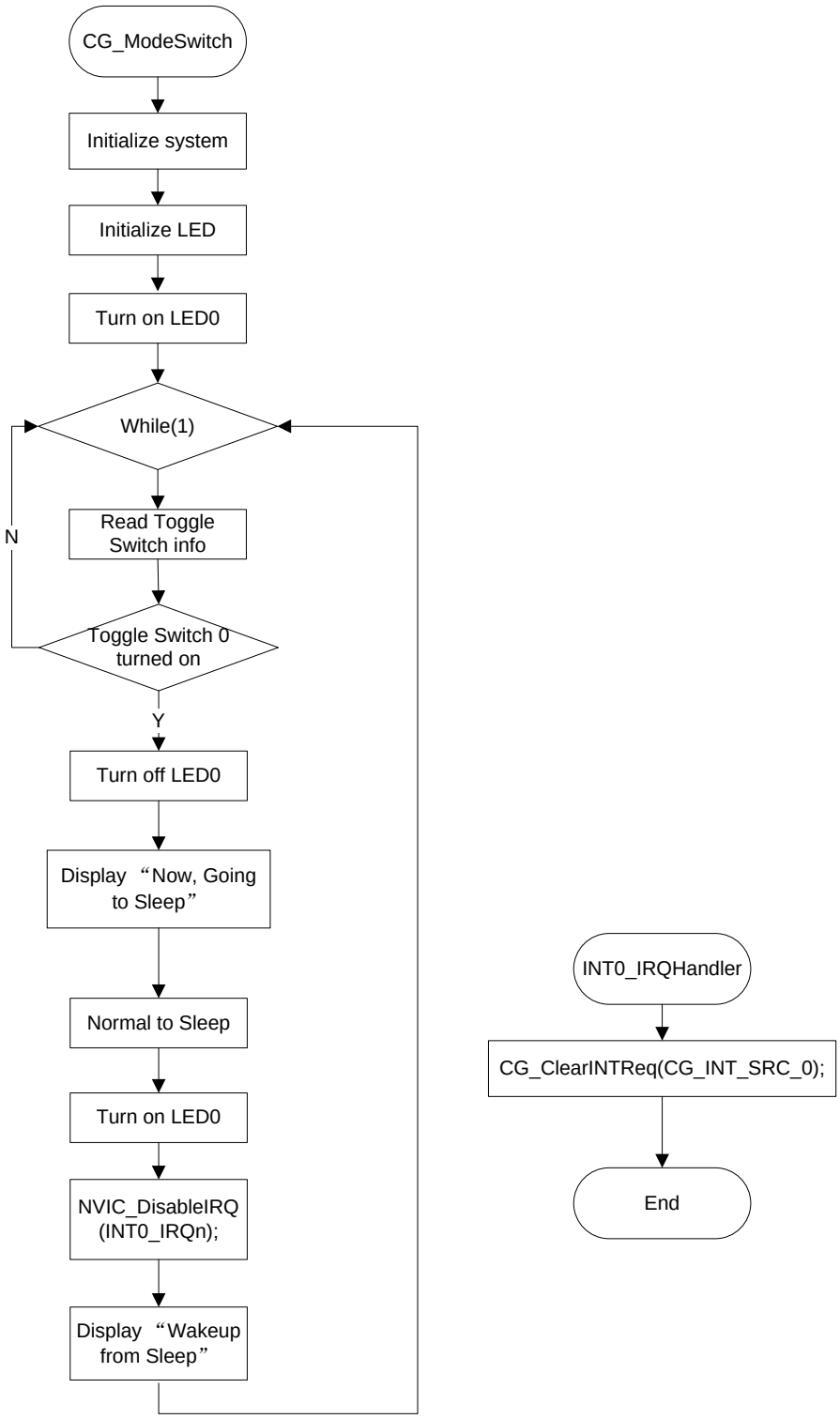
7-2-1 例: Power モード変更

ペリフェラルドライバ(CG, GPIO)を用いたサンプルプログラムです。

以下の例が含まれます:

1. 基本的な CG 動作の設定
2. NORMAL モードと STOP1 モードの切り替え方法
3. マルチクロック回路の許可方法

- **フローチャート:**



- **サンプルプログラムのコードと説明**

CG の初期化

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

```
/* Set SCOUT source */
CG_SetSCOUTSrc(CG_SCOUT_SRC_PHIT0);
```

```
    if (CG_GetFoscSrc()==CG_FOSC_OSC_INT){
        /* Switch over from IHOSC to EHOSC*/
        switchFromIHOSCtoEHOSC();
    }
    /* Set up pll and wait 200us for pll to warm up , set fc source to fpll */
    CG_EnableClkMulCircuit();
    /* Set fgear = fc/2 */
    CG_SetFgearLevel(CG_DIVIDE_2);
    /* Set fperiph to fgear */
    CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
    CG_SetPhiT0Level(CG_DIVIDE_64);
    /* Set low power consumption mode stop1 */
    CG_SetSTBYMode(CG_STBY_MODE_STOP1);
    /* Set pin status in stop mode to "active" */
    CG_SetPinStateInStop1Mode(ENABLE);
```

STOP1 モードへの遷移設定

STOP1 モードへの遷移設定を行います。ウォームアップ時間を設定します。
解除要因として INT0 の設定を行います。INT0 割り込みを許可し、割り込み要求をクリアします。最後に__WFI()命令を実行して STOP1 モードへ遷移します。

```
    /* Set CG module: Normal ->STOP1 mode */
    CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT, CG_WUODR_INT);
    /* Set wake up system */
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_0,        CG_INT_ACTIVE_STATE_H,
ENABLE);

    /* Disable interrupts */
    __disable_irq();
    CG_ClearINTReq(CG_INT_SRC_0);
    NVIC_ClearPendingIRQ(INT0_IRQn);
    NVIC_EnableIRQ(INT0_IRQn);
    CG_ClearINTReq(CG_INT_SRC_0);
    __enable_irq();

    /* Enter stop1 mode */
    __WFI();
```

マルチクロック回路の許可

PLL を設定し、fc ソースを設定します。まず、PLL 値をセットし、PLL を許可します。そしてウォームアップ時間を設定し、ウォームアップ時間が完了するまで待ちます。その後、fPLL を fc ソースとして設定します。

```
    Result retval = ERROR;
    WorkState st = BUSY;
    CG_SetPLL(DISABLE);
    CG_SetFPLLValue(CG_FPLL_MULTIPLY_8);
    retval = CG_SetPLL(ENABLE);
    if (retval == SUCCESS) {
        /* Set warm up time */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT, CG_WUODR_PLL);
        CG_StartWarmUp();

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

        retval = CG_SetFcSrc(CG_FC_SRC_QUARTER_FPLL);
    } else {
```

```
        /*Do nothing */
    }

    return retval;
```

7-3 DAC

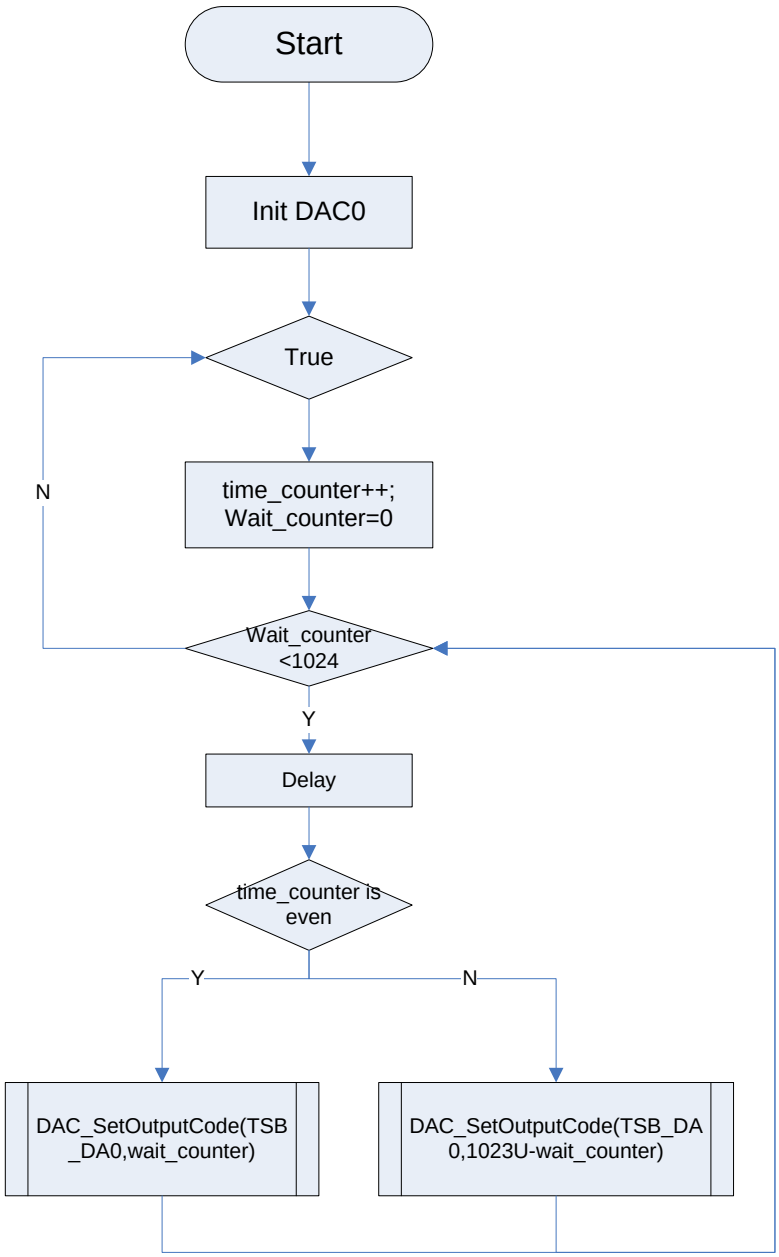
7-3-1 例: のこぎり波形出力

ペリフェラルドライバ(DAC)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。

DAC0 からのこぎり波形を出力します。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

以下は DAC のドライバを使用して、DA0 端子からのこぎり波形を出力するサンプルプログラムです。

まず、DA0 出力のための初期設定を行い、動作を開始します。

```
    DAC_SetOutputCode(TSB_DA0,0U);
    DAC_Start(TSB_DA0);
```

のこぎり波形を出力するため、固定間隔で出力を変化させます。

```
  while(1){
        time_counter++;

        for(wait_counter=0U;wait_counter<1024U;wait_counter++){
            for(i=0U;i<3000U;++i){
```

```
                /* Do nothing */
            }
            if( time_counter%2U == 1U){
                DAC_SetOutputCode(TSB_DA0,wait_counter);
            }
            else{
                DAC_SetOutputCode(TSB_DA0,1023U-wait_counter);
            }
        }
    }
```

7-4 DMAC

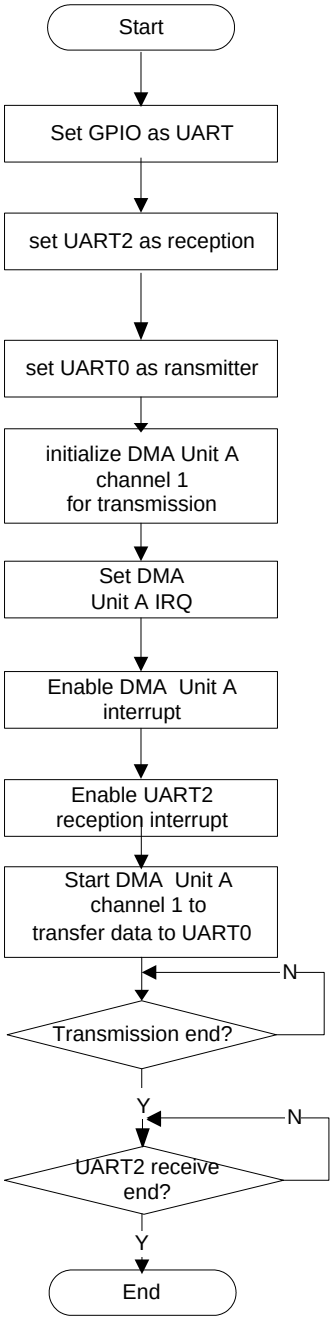
7-4-1 例: メモリから周辺回路

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

以下の例が含まれます。

1. UART0/2 の初期化と DMAC の初期化を行います。
2. メモリから DMAC 経由で UART0 へデータを転送します。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

以下は DMAC のドライバを使用して、メモリから UART0 へデータ転送を行うサンプルプログラムです。

まず、データ転送のために UART0 の設定と許可を行います。

```
UART_InitStruct.Mode = UART_ENABLE_TX;
UART_Enable(UART0);
UART_Init(UART0, &UART_InitStruct);_
UART_SetTxDMAReq(UART0, ENABLE);
```

データ転送のために DMAC ユニット A のチャネル 1 の初期化と設定を行います。

```
DMAC_InitStruct.SrcAddr = (uint32_t) SRC_Buffer;
DMAC_InitStruct.SrcIncrementState = ENABLE;
```

```
    DMAC_InitStruct.SrcBitWidth = DMAC_BYTE;
    DMAC_InitStruct.SrcBurstSize = DMAC_1_BEAT;
    DMAC_InitStruct.DstAddr = (uint32_t) (UART0_BASE_ADDR + UART0_BUF_OFFSET);
    DMAC_InitStruct.DstIncrementState = DISABLE;
    DMAC_InitStruct.DstBitWidth = DMAC_BYTE;
    DMAC_InitStruct.DstBurstSize = DMAC_1_BEAT;
    DMAC_InitStruct.TxSize = size;
    DMAC_InitStruct.TxDirection = DMAC_MEMORY_TO_PERIPH;
    DMAC_InitStruct.A_TxDstPeriph = DMACA_SIO0_UART0_TX;
    DMAC_InitStruct.TxINT = ENABLE;

    DMAC_Init(DMAC_UNIT_A, myChannel, &DMAC_InitStruct);
```

DMAC ユニット A の割り込みを許可します。

```
    NVIC_ClearPendingIRQ(INTDMAC0TC_IRQn);
    NVIC_EnableIRQ(INTDMAC0TC_IRQn);
```

DMAC ユニット A の許可、送信終了割り込みの設定、UART0 へのバースト転送設定を行い、DMAC ユニット A のチャネル 1 から転送を開始します。

```
    DMAC_Enable(DMAC_UNIT_A);
    DMAC_SetTxINTConfig(DMAC_UNIT_A, myChannel, DMAC_INT_TX_END, ENABLE);
    DMACA_SetSWBurstReq(DMACA_SIO0_UART0_TX);
    DMAC_SetDMAChannel(DMAC_UNIT_A, myChannel, ENABLE);
```

DMAC 転送が終わるまで待ちます。

```
    while (TxEndFlag != DONE) {
        /* Do nothing */
    }
```

DMAC 転送終了時に ISR の DMAC ユニット A にフラグがセットされます。

```
    state = DMAC_GetINTReq(DMAC_UNIT_A);
    if (state.Bit.CH1_INTReq == 1U) {
        tmp = DMAC_GetTxINTReq(DMAC_UNIT_A, DMAC_CHANNEL_1);
        if (tmp == DMAC_TX_END_REQ) {
            TxEndFlag = DMAC_GetChannelTxState(DMAC_UNIT_A, DMAC_CHANNEL
  _1);
            DMAC_ClearTxINTReq(DMAC_UNIT_A, DMAC_CHANNEL_1, DMAC_INT_T
  X_END);
        } else {
            /* Do nothing */
        }
    } else {
        /* Do nothing */
    }
```

7-5 EXB

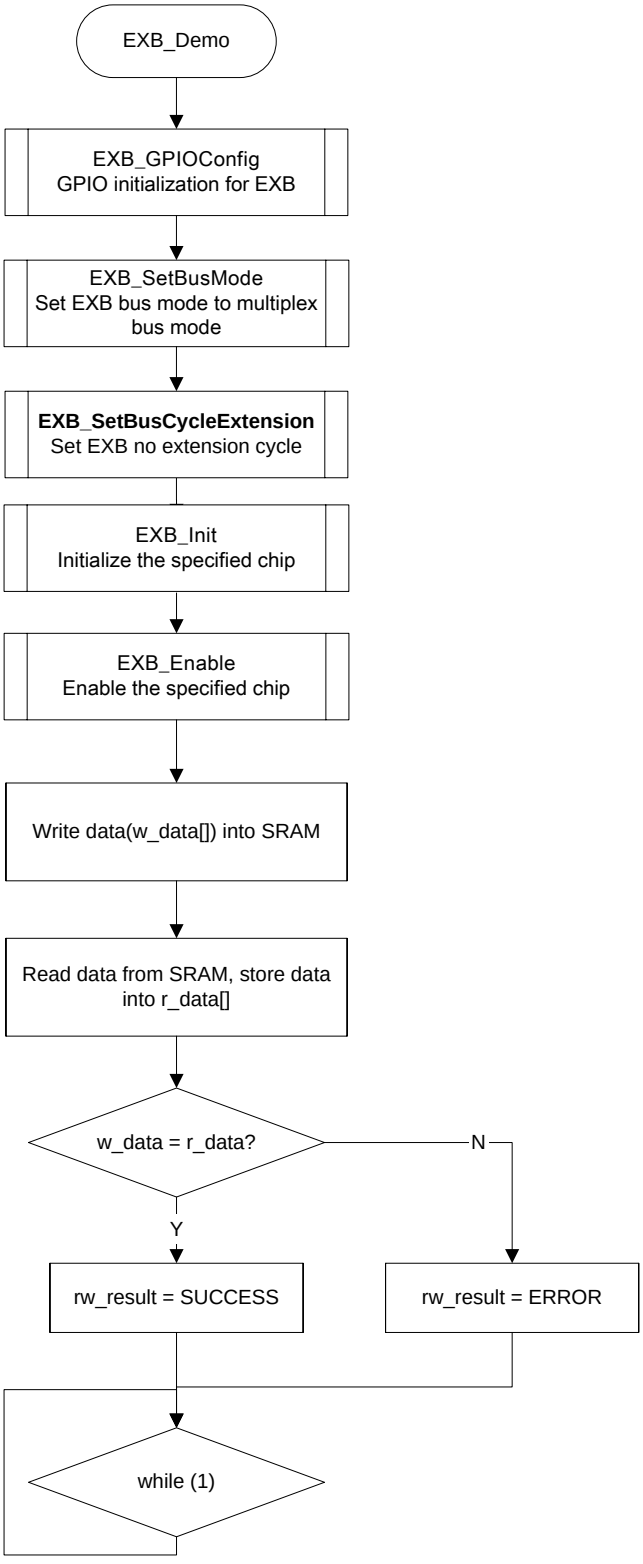
7-5-1 例: SRAM のリード/ライト

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

以下の例が含まれます。

1. EXB の初期設定
2. 外部 SRAM のリード/ライト

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、EXB の初期化を行います。

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

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

	EXB_InitTypeDef InitStruct = { 0U };

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

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

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

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

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

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

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

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

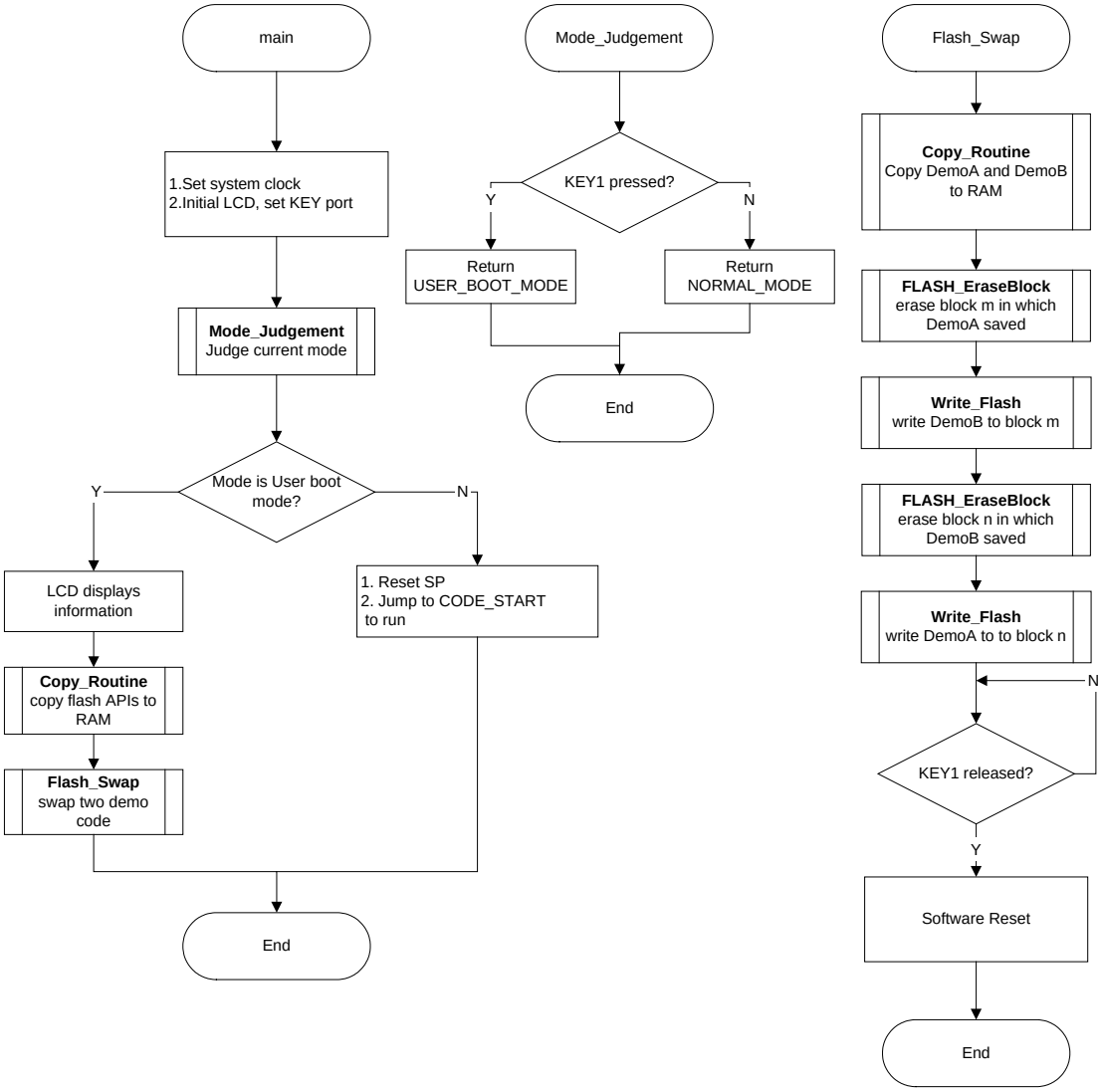
7-6 FLASH

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

以下の例が含まれています。

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず LED と SW を初期化します。リセット時に現在のモード判定を SW で、現在の状態表示を LED で行います。

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

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

```
uint8_t Mode_Judgement(void)
{
    return (GPIO_ReadDataBit(GPIO_PD, GPIO_BIT_0) ==
            GPIO_BIT_VALUE_0) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

リセット時に SW 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 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 SW0 が押されている場合、ユーザブートモードになります(LED3 が点灯)。Flash メモリは自分自身で消去/書き込みを実行できないため、Flash 動作用 API を、Flash メモリ内の アドレス “FLASH_API_ROM” から RAM の “FLASH_API_RAM”にコピーします。RAM 転送中、LED2 が点灯します。

```
Status_Display(0x08);       /* status 1: enter user boot mode */
Status_Display(0x04);       /* status 2: RAM transferring */
Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
                                        /* copy flash API to RAM */
```

Flash 動作用 API を RAM にコピー後、ルーチンプログラムは Flash_Swap()関数にジャンプします。この関数は、上記の Copy_Routine() を使用し、Flash メモリから 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。スワップ中、LED1 が点灯します。

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

Status_Display(0x02);               /* status 2: swap the demo */

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

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

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

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

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

スワップ完了するとLED3とLED2が点灯します。SWが離されると、SCB->AIRCR レジスタを用いてソフトリセットを行います。その後、ノーマルモードになります。サンプル A とサンプル B が差し替えられているため、アドレス “CODE_START” はサンプル B のスタートアドレスになり、サンプル B が動作します(LED1 が点灯)。

```
Status_Display(0x0C);                    /* status 3: complete and restart */

while (GPIO_ReadDataBit(GPIO_PD, GPIO_BIT_0) == GPIO_BIT_VALUE_0) {
            /* 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 (ENABLE == FC_GetBlockProtectState(BlockNum)) {
      retval = FC_ERROR_PROTECTED;
}
```

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

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

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

```
while (BUSY == FC_GetBusyState()) {
                                    /* check if FLASH is busy with overtime counter */
      if (!(counter--)) {                    /* check overtime */
            retval = FC_ERROR_OVER_TIME;
            *addr1 = FC_RESET_CMD;          /* Reset FLASH */
             break;
      } else {
             /* Do nothing */
      }
}
```

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

```
*addr1 = (uint32_t) 0x000000AA;          /* bus cycle 1 */
*addr2 = (uint32_t) 0x00000055;          /* bus cycle 2 */
*addr1 = (uint32_t) 0x000000A0;          /* bus cycle 3 */
for (i = 0U; i < FC_PAGE_SIZE; i++) {                    /* bus cycle 4~7 */
```

```
        *addr3 = *source;
        source++;
    }
```

7-7 GPIO

ペリフェラルドライバ(GPIO)を用いたサンプルプログラムです。

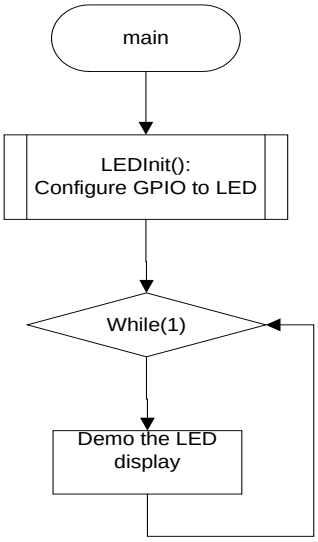
この例は以下を行います。

1. GPIO の初期化
2. GPIO へのデータ書き込み
3. GPIO からのデータ読み出し

補足:

1. IAR TMPM341-SK board では、LED2 の On/Off に PE2 を使用するため、ジャンパ E JP11 をショートしてください。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず GPIO_SetOutput()関数を用いて GPIO を LED に設定します。以下は GPIO_WriteData()関数を用いて LED を点灯する例です。

```
    GPIO_SetOutput(LED_DATA_PORT,0X04U);
    GPIO_WriteData(LED_DATA_PORT,0X04U);
```

サンプルでは、while ループにて LED On と LED Off を繰り返します。

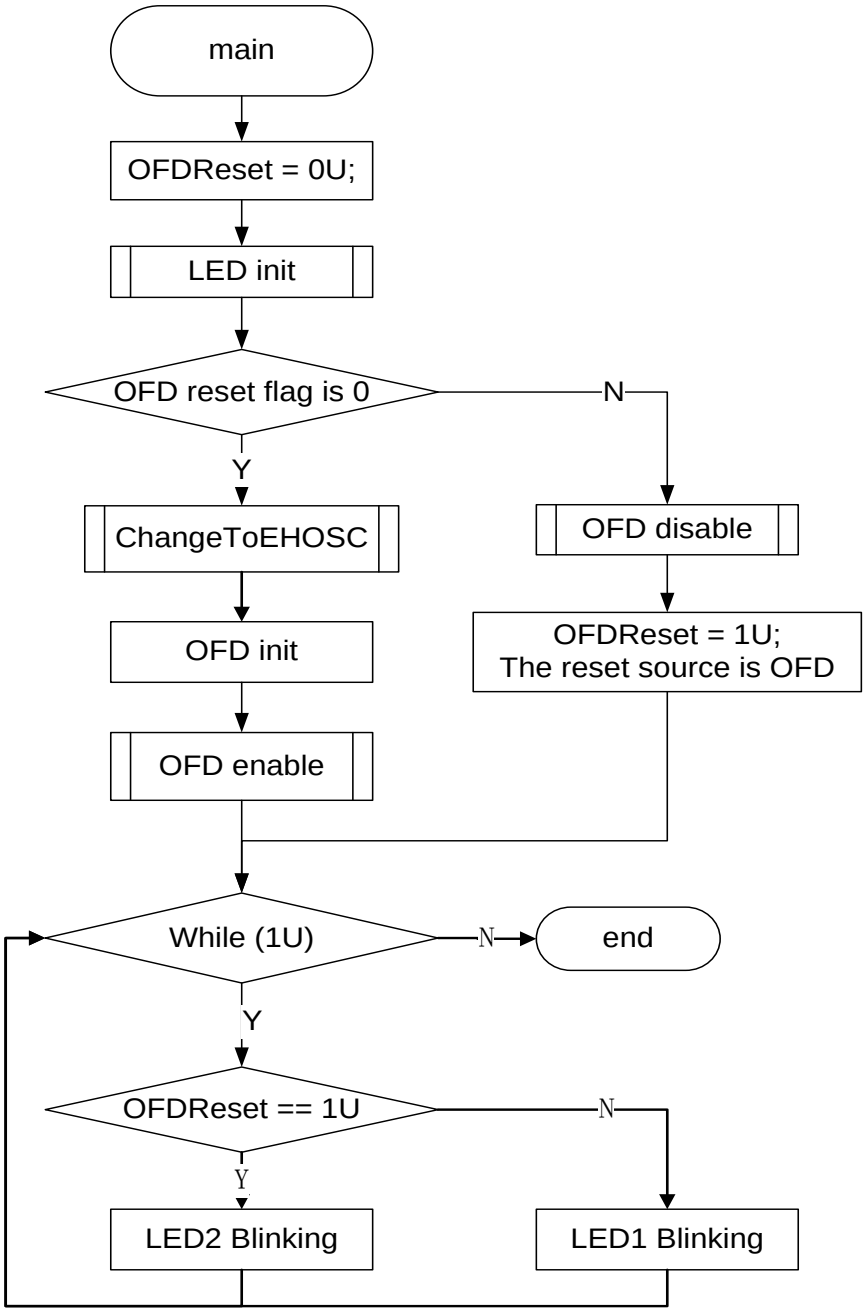
7-8 OFD

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

以下の例が含まれます。

1. OFD の初期化(検知周波数範囲の設定)
2. OFD リセットフラグの確認
3. OFD の有効／無効

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、LED を初期化します。

```
    LEDInit();
```

OFD のリセットフラグを確認します。

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

例えば、通常のリセットでは、MCU は電源投入によるリセットで OFD フラグは"0"となり、プログラムは EHOSC を許可し、検知周波数範囲を OFD に設定し、OFD を初期化します。

以下は EHOSC を許可する CG の設定です。

```
void ChangeToEHOSC(void)
{
    /* Use the external oscillation */
    TSB_CG->OSCCR &= 0x000FFFFFUL;
    TSB_CG->OSCCR |= 0xFFF00000UL;                /*Set the warm up time WUODR[13:2]*/
    TSB_CG_OSCCR_EHOSCSEL = 1U;/*Select external oscillator */
    TSB_CG_OSCCR_XEN1 = 1U;/*Enable external oscillator */
    TSB_CG_OSCCR_HWUPSEL = 1U;/*Select warm-up clock */
    TSB_CG_OSCCR_WUEON = 1U;      /*Start warm up */
    while (TSB_CG_OSCCR_WUEF) {}/* Warm-up */
    TSB_CG_OSCCR_OSCSEL = 1U;     /*Use the external oscillation */
    TSB_CG_OSCCR_XEN2 = 1U;       /*Enable internal oscillator for ofd */
}
```

OFD の設定のため、まずこのレジスタ設定にて許可します。

```
    OFD_SetRegWriteMode(ENABLE);
```

検知周波数を設定します。

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

DEMO2 を定義すると、異常な周波数範囲が設定されます。

```
    OFD_SetDetectionFrequency(OFD_HIGHER_COUNT_ABNORMAL,
                                                OFD_LOWER_COUNT_ABNORMAL);
```

初期化し、その後 OFD を有効にします。

```
    OFD_Enable();
```

上記の設定を行い、その後 OFD は外部クロックを検知する準備を行います。このクロックが検知周波数範囲を超える (例えば、EHOSC が外れる、発振が乱れる、DEMO2 が定義されている) と OFD はリセット信号を生成します。この結果 MCU は自動的にリセットを行います。この OFD リセットフラグを検知すると、MCU はデフォルトの IHOSC で動作します。この結果、OFD 機能が無効となり、OFDReset 変数に"1"をセットします。

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

システムクロック状態を示す LED1 と LED2 の意味は次の通りです。

LED 状態	意味
LED1 点滅	EHOSC が正常であり、MCU は通常動作します。 OFD 機能は有効であり、異常周波数を検知できる状態になります。
LED2 点滅	EHOSC が異常であり、OFD リセットが行われるため、MCU は IHOSC で動作します。この場合、OFD 機能は無効です。

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

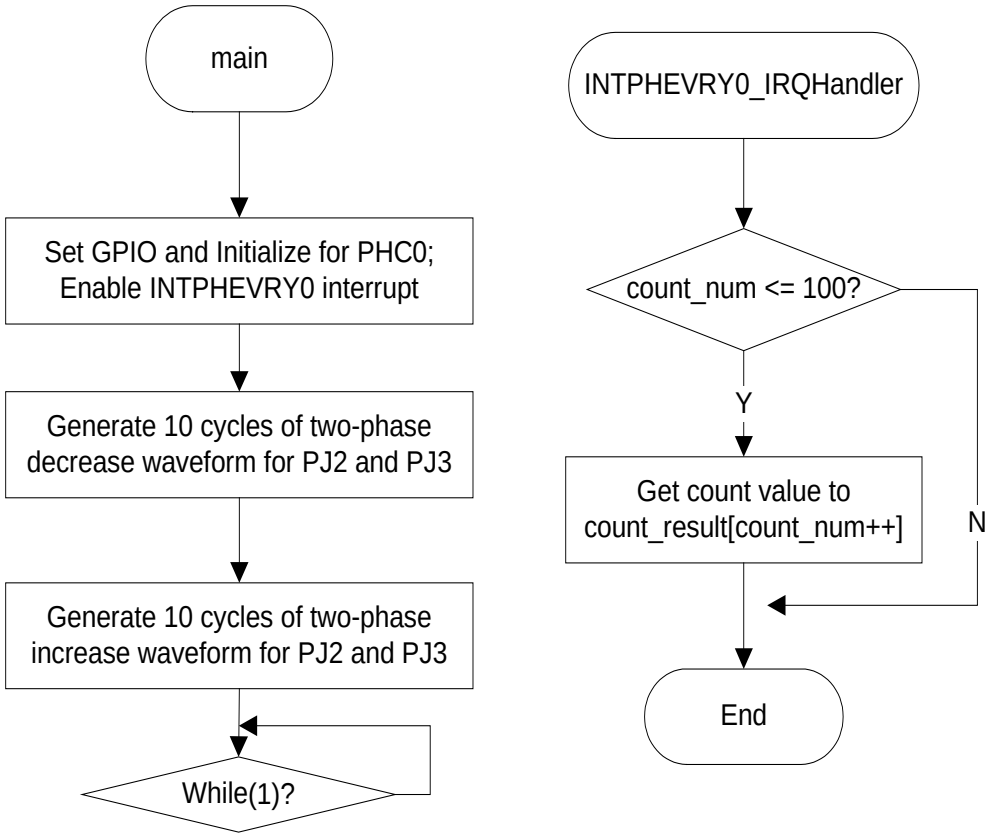
7-9 PHC

ペリフェラルドライバ (PHC, 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. 4 逓倍モードでの PHC アップ/ダウンカウンター動作

- **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、GIPO 機能を初期化します。

PJ0 - PHC0IN0 機能、入力

PJ1 - PHC0IN1 機能、入力

PJ2 - GPIO 出力

PJ3 - GPIO 出力

```
    GPIO_SetInput(GPIO_PJ, (GPIO_BIT_1 | GPIO_BIT_0));
    GPIO_SetOutput(GPIO_PJ, (GPIO_BIT_2 | GPIO_BIT_3));
    GPIO_EnableFuncReg(GPIO_PJ, GPIO_FUNC_REG_3, (GPIO_BIT_1 |
GPIO_BIT_0));
    GPIO_DisableFuncReg(GPIO_PJ, GPIO_FUNC_REG_3, (GPIO_BIT_2 |
GPIO_BIT_3));
    GPIO_SetInputEnableReg(GPIO_PJ, (GPIO_BIT_1 | GPIO_BIT_0), ENABLE);
    GPIO_SetOutputEnableReg(GPIO_PJ, (GPIO_BIT_2 | GPIO_BIT_3), ENABLE);
```

PHC 初期化構成、PHC モード入力、ノイズフィルタ設定とカウンタークリア設定を行います。こ

```
    PHC_InitTypeDef InitStruct;

    InitStruct.Mode = PHC_CR_MODE_4TIMES; /* Quadruple mode */
    InitStruct.NoiseFilterCtrl = PHC_CR_NOISEFILTER_ON; /* Noise Filter on */
    InitStruct.CountClearCtrl = PHC_COUNT_CLR; /* Clear counter */
```

PHC モジュールをイネーブルにし、初期化構成を指定済のレジスタに設定します。INTPHEVRY0 割り込みをイネーブルにすると、本割り込みはカウンタが変化するごとに発生します。最後に、PHC が実行されます。

```
    PHC_Enable(TSB_PHC0); /* Enable PHC Operation */
    PHC_Init(TSB_PHC0, &InitStruct); /* Set InitStruct to PHC */
    PHC_DisableInterrupt(TSB_PHC0, PHC_CR_INT_ALL); /* Disable all interrupt */
    PHC_EnableInterrupt(TSB_PHC0,PHC_CR_INT_EVERY); /* Enable
INTPHEVRY0 interrupt */
    NVIC_EnableIRQ(INTPHEVRY0_IRQn);
    PHC_SetRunState(TSB_PHC0, PHC_RUN); /* Run PHC0 */
```

その後メインルーチンは、割り込み発生の待機をするために“While(1)”へ移行します。割り込みサービスルーチンで取得したカウンター値を使用してください。

```
    void INTPHEVRY0_IRQHandler(void)
    {
        if(count_num <= 100) {          /*Judge if overflow count_result[] or not*/
            count_result[count_num++] = PHC_GetPulseCntValue(TSB_PHC0); /* get
count value*/
        } else {
            //Do nothing
        }
    }
```

7-10 SBI

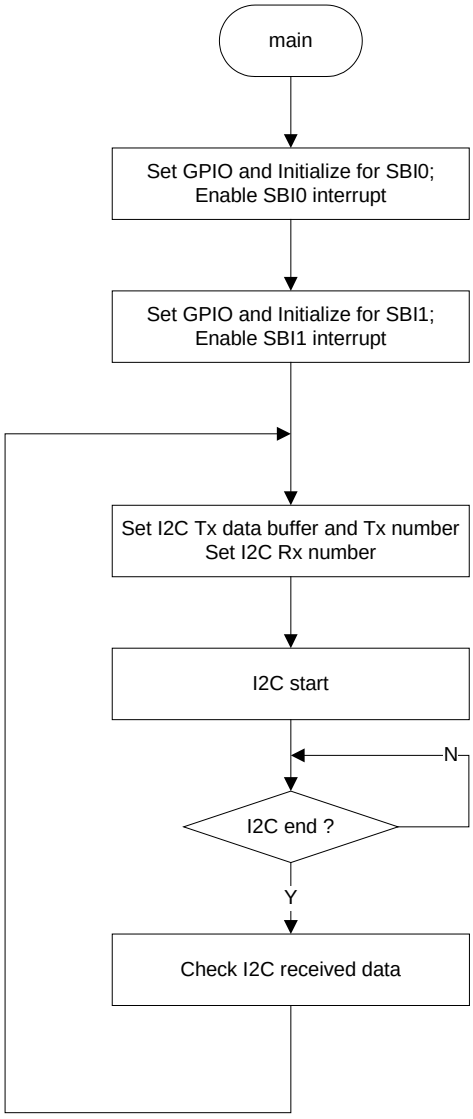
ペリフェラルドライバ(SBI, GPIO)を使用したサンプルプログラムです。

以下の例が含まれます。

1. SBI 設定、I2C 初期化

2. I2C マスタによるデータプロセスの送信
3. I2C スレーブによるデータプロセスの受信

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、GPIO を SBI0 と SBI1 に設定します。

```
SBI0_IO_Configuration();
SBI1_IO_Configuration();
```

次に、SBI0 をイネーブルし、初期化します。その後 INTSBI0 をイネーブルにします。

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

そして SBI1 をイネーブルし、初期化します。その後 INTSBI1 をイネーブルにします。

```
    myI2C.I2CSelfAddr = SLAVE_ADDR;
    myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
    myI2C.I2CACKState = ENABLE;
    myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
    SBI_Enable(TSB_SBI1);
    SBI_SWReset(TSB_SBI1);
    SBI_InitI2C(TSB_SBI1, &myI2C);
    NVIC_EnableIRQ(INTSBI1_IRQn);
```

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

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

I2C バスが空いているかどうか、“SLAVE_ADDR” データを SBI_SetSendData()に設定します。そして、送信方向を“SBI_I2C_SEND”から SBI データバッファへ設定します。その後、SBI_GenerateI2CStart(TSB_SBI0) を使用して、I2C 動作を開始します。

```
        /* Check I2C bus state and start TRx */
    case MODE_SBI_I2C_START:
        i2c_state = SBI_GetI2CState(TSB_SBI0);
        if (!i2c_state.Bit.BusState) {
            SBI_SetSendData(TSB_SBI0, SLAVE_ADDR | SBI_I2C_SEND);
            SBI_GenerateI2CStart(TSB_SBI0);
            gSBIMode = MODE_SBI_I2C_TRX;
        } else {
            /* Do nothing */
        }
        break;
```

INTSBI0 でデータ転送を行います。
INTSBI1 でデータ受信を行います。

INTSBI0 ハンドラ内で、I2C バス状態を取得し、その値で I2C マスタ送信プロセスを決定します。I2C マスタ送信中は、I2C_SetSendData() で次のデータを送信し、I2C プロセス終了時には、I2C_GenerateStop() で I2C を停止します。

```
void INTSBI0_IRQHandler(void)
{
    TSB_SBI_TypeDef *SBIx;
    SBI_I2CState sbi_sr;
```

```
    SBIx = TSB_SBI0;
    sbi_sr = SBI_GetI2CState(SBIx);

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

INTSBI1 ハンドラで、I2C バス状態を取得し、その値で I2C スレーブ受信プロセスを決定します。SBI バッファの受信データ読み出しは I2C_GetReceiveData() 関数を用いて行います。I2C 停止状態はマスタによって制御します。

```
void INTSBI1_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI_TypeDef *SBIy;
    SBI_I2CState sbi1_sr;

    SBIy = TSB_SBI1;
    sbi1_sr = SBI_GetI2CState(SBIy);

    if (!sbi1_sr.Bit.MasterSlave) {         /* Slave mode */
        if (!sbi1_sr.Bit.TRx) { /* Rx Mode */
            if (sbi1_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRCnt = 0U;
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRxData[gI2CRCnt] = tmp;
                gI2CRCnt++;
            }
        } else {                        /* Tx Mode */
            /* Do nothing */
        }
    } else {                            /* Master mode */
        /* Do nothing */
    }
}
```

7-11 SIO/UART

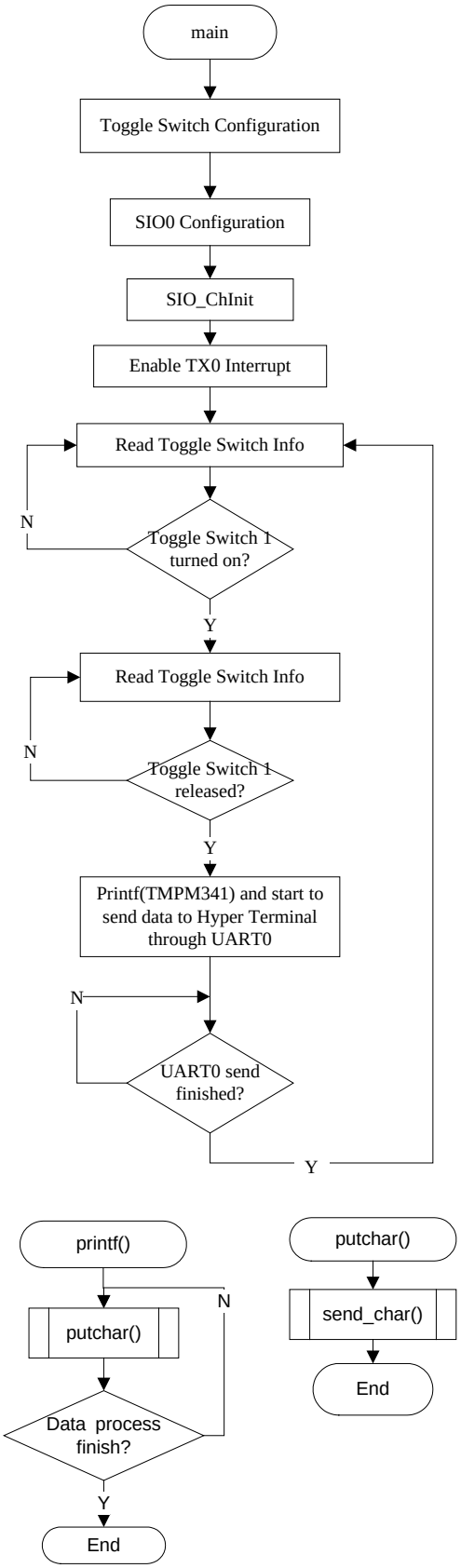
7-11-1 例: リターゲット

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

この例では以下を行います。

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

GPIO を SW に設定します。

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

GPIO を UART0 に設定します。

```
GPIO_SetOutputEnableReg(GPIO_PC, GPIO_BIT_4, ENABLE);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_2, GPIO_BIT_4);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_2, GPIO_BIT_1);
GPIO_SetInputEnableReg(GPIO_PC, GPIO_BIT_1, ENABLE);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_0, ENABLE);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_1, ENABLE);
```

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

```
UART_InitTypeDef myUART;

myUART.BaudRate = 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;
```

UART0 を初期化します。

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

上記設定を行い、その後、UART0 の送信割り込みを有効にします。

```
NVIC_EnableIRQ(INTTX0_IRQn);
```

SW 状態を取得します。

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

SW1 の状態を取得し、On の場合 Off になるまで待ちます。

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

SW1 が Off になると、UART0 経由で printf()にてデータを送信します。

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

      return ch;
}
```

最後に UART0 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 */
            fSIO_INT = SET;               /* SIO0 INT is enable */
      } else {
            /* disable SIO0 INT */
            fSIO_INT = CLEAR;
            NVIC_DisableIRQ(INTTX0_IRQn);
            fSIOTxOK = YES;
      }
      if (gSIORdIndex >= gSIOWrIndex) {     /* reset buffer index */
            gSIOWrIndex = CLEAR;
            gSIORdIndex = CLEAR;
      } else {
            /* do nothing */
      }
}
```

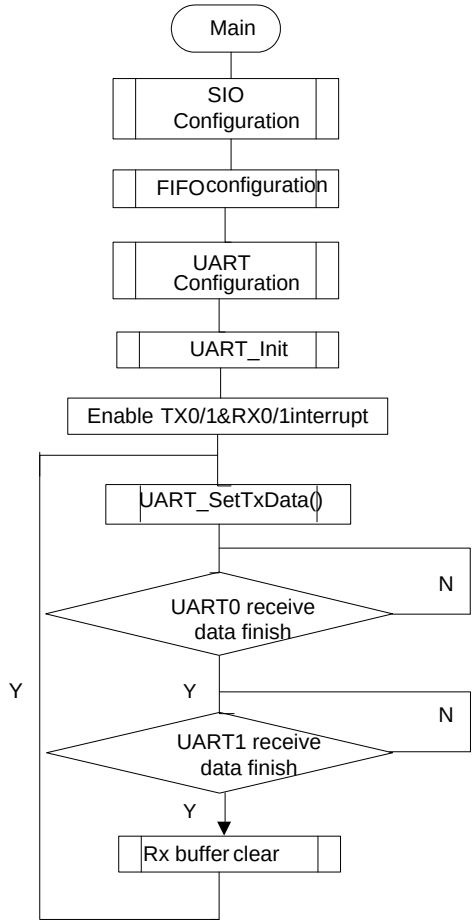
7-11-2 例: UART FIFO

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

この例では以下を行います。

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

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

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
      if (SCx == TSB_SC0) {
            TSB_PE->CR |= GPIO_BIT_0;
            TSB_PE->FR1 |= GPIO_BIT_0;
            TSB_PE->FR1 |= GPIO_BIT_1;
            TSB_PE->IE |= GPIO_BIT_1;
      } else if (SCx == TSB_SC1) {
            TSB_PC->CR |= GPIO_BIT_0;
            TSB_PC->FR2 |= GPIO_BIT_0;
            TSB_PC->FR2 |= GPIO_BIT_1;
            TSB_PC->IE |= GPIO_BIT_1;
      }
}
```

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

```
      UART_InitTypeDef myUART;

    /* configure SIO0 for reception */
    UART_Enable(UART_RETARGET);
    myUART.BaudRate = 115200U;    /* baud rate = 115200 */
```

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

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

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

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

FIFO の設定を行います。

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

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

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

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

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

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

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

    UART_RxFIFOClear(UART0);
    UART_RxFIFOClear(UART1);

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

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

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

上記設定を行い、その後 UART0/1 の各割り込みを許可します。

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

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

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

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

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

以下は UART1 の送信割り込み処理ルーチンです。

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

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

以下は UART0 の受信割り込み処理ルーチンです。

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

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

以下は UART1 の受信割り込み処理ルーチンです。

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

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

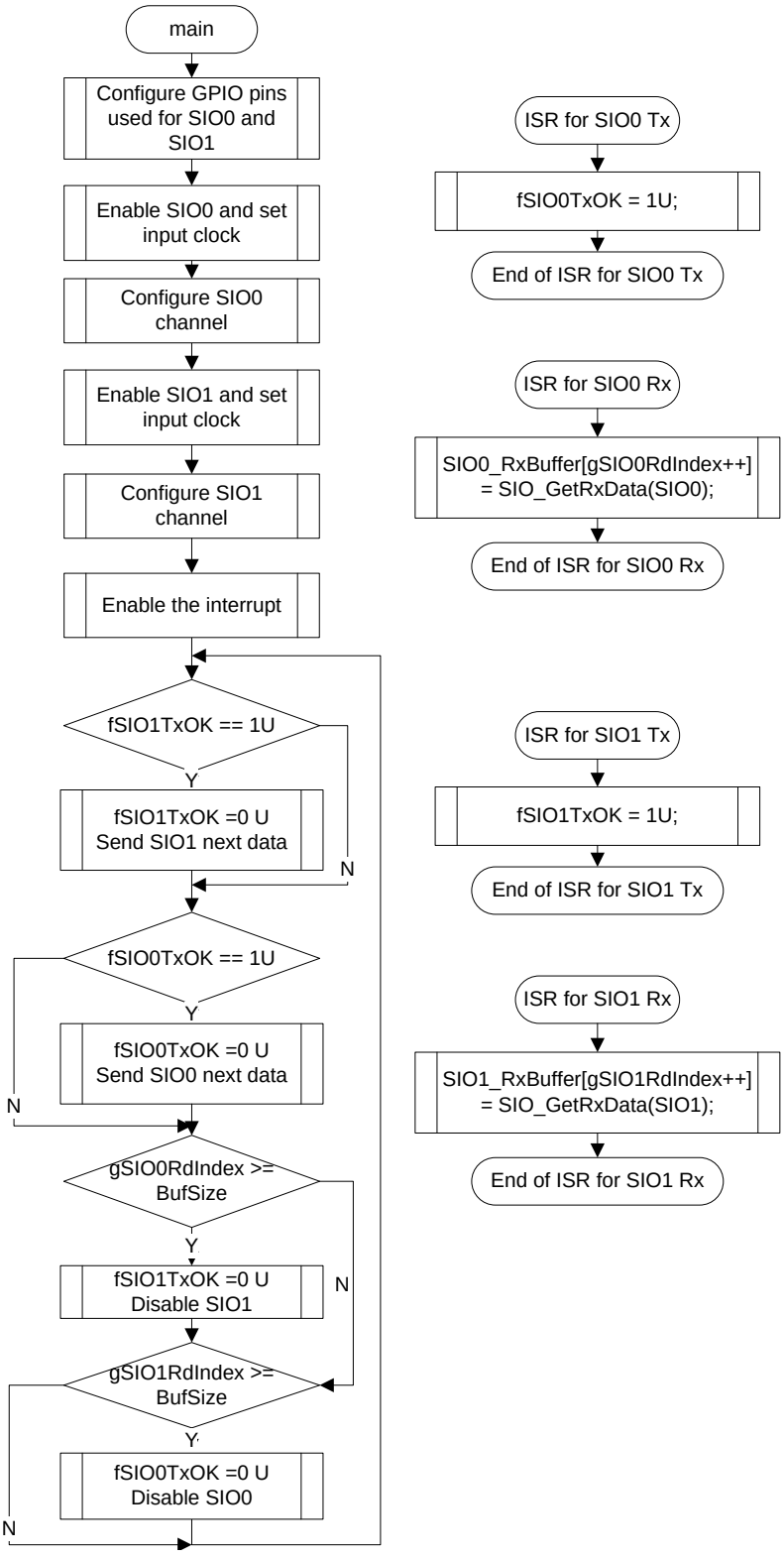
7-11-3 例: SIO

ペリフェラルドライバ(SIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. SIO 動作の基本設定
2. SIO0~SIO1 間のデータ転送
3. SIO の送受信割り込み

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に GPIO 端子を SIO に設定します。

その後、SIO0 を有効にし、入力クロックの設定と SIO0 の初期化を行います。

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

```
    /*initialize the SIO0 struct */
    SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
    SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_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);
}
```

7-12 SSP

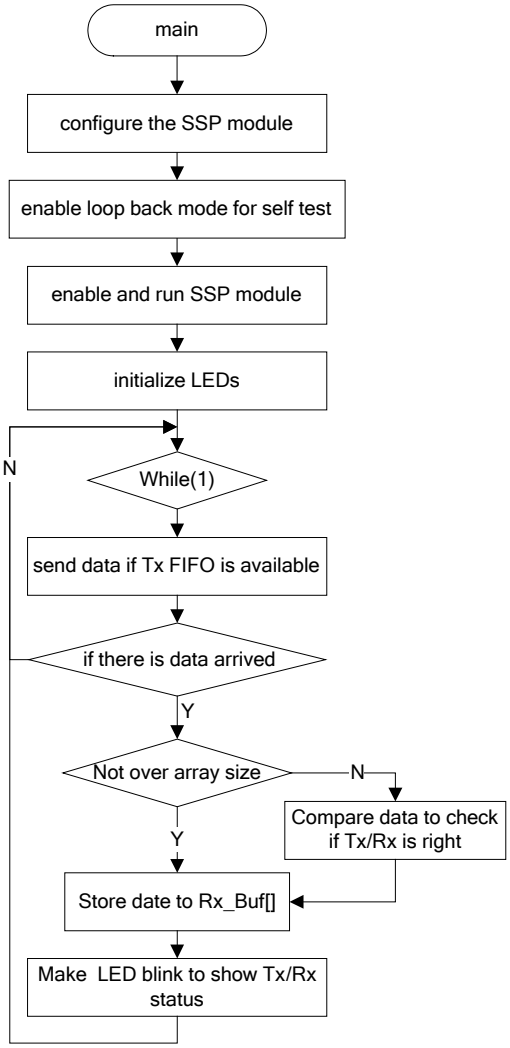
7-12-1 例: SSP セルフループバック

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

以下の例が含まれます。

1. SSP の初期化
2. ループバック設定

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず SSP_InitTypeDef 構造体と他の変数を準備し、初期値を設定します。

```
SSP_InitTypeDef initSSP;_
SSP_FIFOState fifoState;_

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

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

/* configure the SSP module */
initSSP.FrameFormat = SSP_FORMAT_SPI;_

/* default is to run at maximum bit rate */
initSSP.PreScale = 2U;_
initSSP.ClkRate = 1U;_
/* define BITRATE_MIN to run at minimum bit rate */
/*     BitRate = fSYS / (PreScale x (1 + ClkRate)) */
```

```
#ifdef BITRATE_MIN
initSSP.PreScale = 254U;_
initSSP.ClkRate = 255U;_
#endif
initSSP.ClkPolarity = SSP_POLARITY_LOW;_
initSSP.ClkPhase = SSP_PHASE_FIRST_EDGE;_
initSSP.DataSize = 16U;_
initSSP.Mode = SSP_MASTER;_
```

上記設定後、API SSP_Init()関数をコールして、SSP を初期化します。

```
SSP_Init(&initSSP);_
```

ループバックモードを許可します。

```
/* enable loop back mode for self test */
SSP_SetLoopBackMode(ENABLE);_
```

SSP 動作を許可します。

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

最後に送信 FIFO を有効化しデータ送信を行います。受信 FIFO がエンプティでなければデータ受信を行います。

```
while (1) {

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

    /* check if there is data arrived */
    fifoState = SSP_GetFIFOState(SSP_RX);_
    if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
        receive = SSP_GetRxData(TSB_SSP);_
        if (cntRx < MAX_BUFSIZE) {
            Rx_Buf[cntRx] = receive;_
            cntRx++;
        } else {
            /* Place a break point here to check if receive data is right. */
            /* Success Criteria:       */
            /* Every data transmited from Tx_Buf is received in Rx_Buf. */
            /* When the line "#define BITRATE_MIN" is commented, the SSP
               is run in maxium */
            /* bit rate, so we can find there is enough time to transmit date
               from 1 to    */
            /* MAX_BUFSIZE one by one. but if we uncomment that line, SSP
               is run in          */
            /*   minimum bit rate, we will find that receive data can't catch
                 "datTx++", */
```

```
                    /*    in this so slow bit rate, when the Tx FIFO is available, the
                          cntTx has */
                    /*    been increased so much.    */
                     __NOP();
                     result = Buffercompare( Tx_Buf, Rx_Buf, MAX_BUFSIZE);_
                }

          } else {
                /* do nothing */
          }

          DisplayLED(receive);_
    }
```

7-13 TMRB

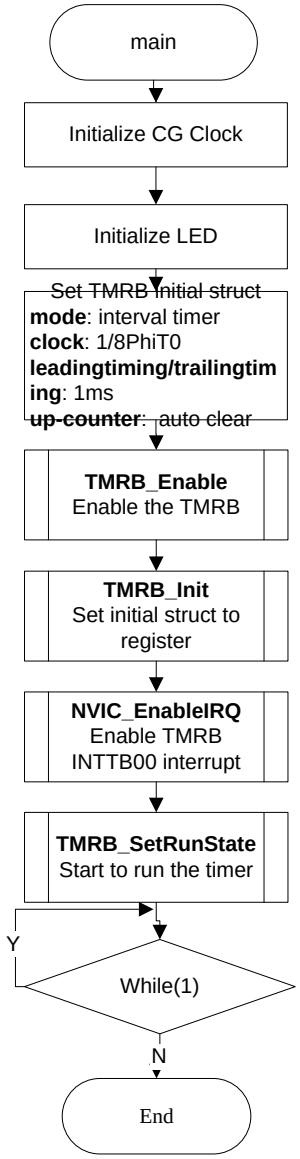
7-13-1 例: 汎用タイマ

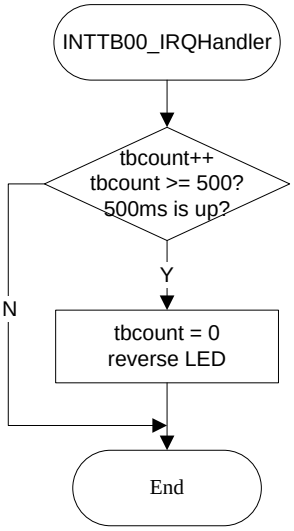
ペリフェラルドライバ(TMRB, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. TMRB0 の初期化
2. 1ms の汎用タイマ

- **フローチャート:**





- **サンプルプログラムのコードと説明**

最初に LED を初期化し、LED を On します。

```
CG_InitSystem();                    /* CG_SetSystem */
LEDInit();                          /* LED initialize */
LedDisable(LED0 | LED1 | LED2);
LedOn(LED3);                                    /* Turn on LED1 */
```

TMRB 設定用の構造体を用意し TMRB モード、クロック、アップカウンタクリア方法、周期とデューティを設定します。このデモでは、1ms の周期とデューティを設定します。TMRB_1MS マクロは、0x1388 です。(φT0=fsys=10MHz * PLL* = 40MHz, ftmrb = 1/8φT0 = 5MHz, Ttmrb = 0.2us, 1ms/0.2us = 5000 = 0x1388 (クロック設定についての詳細は CG 章を参照してください))

```
TMRB_InitTypeDef m_tmrb;

m_tmrb.Mode = TMRB_INTERVAL_TIMER;      /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;         /* 1/8PhiT0 */
m_tmrb.TrailingTiming = TMRB_1MS;                   /* periodic time is 1ms */
```

```
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
m_tmrb.LeadingTiming = TMRB_1MS;                    /* periodic time is 1ms */
```

TMRB モジュールを有効にした後、初期化構造体を指定のレジスタに設定します。INTTB0 割り込み（1ms ごとにトリガ）を有効にします。最後に TMRB を動作させます。

```
TMRB_Enable(TSB_TB0);                          /* enable the TMRB0 */
TMRB_Init(TSB_TB0, &m_tmrb);                  /* initial the TMRB0 */
NVIC_EnableIRQ(INTTB0_IRQn);                 /* enable INTTB0 interrupt */
TMRB_SetRunState(TSB_TB0, TMRB_RUN);     /* run TMRB0*/
```

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

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

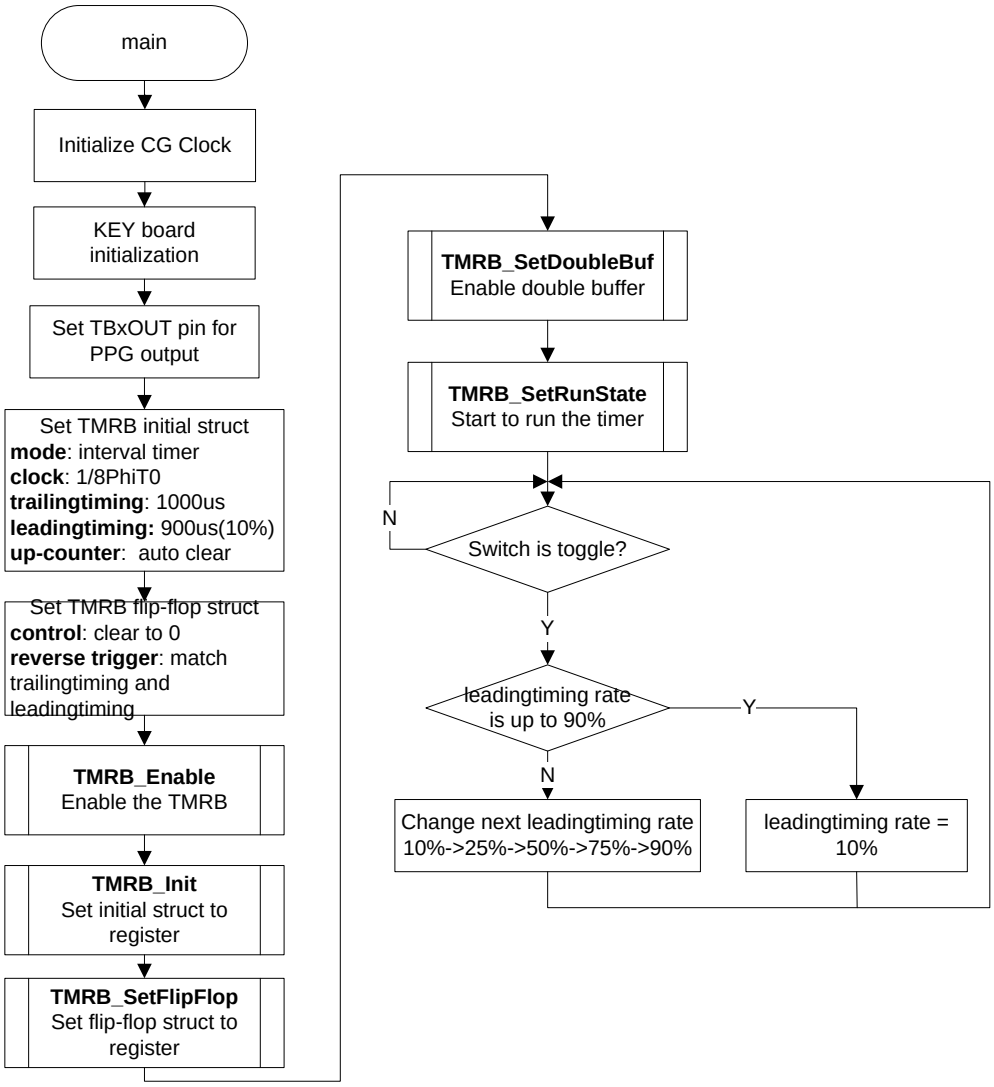
7-13-2 例: PPG 出力

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. TMRB7 の初期化
2. PPG 機能の設定と開始
3. PPG デューティの調整

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に GPIO を SW に設定し、PPG 出力用に PD0 を TB7OUT に設定します。

```
    GPIO_SetInput(KEYPORT, GPIO_BIT_3);                    /* set KEY port to input */
    /* Set PD0 as TB7OUT for PPG output */
    GPIO_SetOutput(GPIO_PD, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_3, GPIO_BIT_0);
```

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 m_tmrb;

    m_tmrb.Mode = TMRB_INTERVAL_TIMER;      /* internal timer */
    m_tmrb.ClkDiv = TMRB_CLK_DIV_8;         /* 1/8PhiT0 */
    m_tmrb.TrailingTiming = TMRB7TIME;              /* trailingtiming is 500us
*/
```

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

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

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

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

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

スイッチ状態の変化を待ちます。

```
    do {                            /* wait if switch is Low */
        keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_3);
       } while (GPIO_BIT_VALUE_0 == keyvalue);
    delay(0xFFFU);                  /* noise cancel */
```

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

```
    do {
          keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_3);
       } while (GPIO_BIT_VALUE_1 == keyvalue);
    delay(0xFFFU);                  /* noise cancel */

    Rate++;
    if (Rate >= LEADINGTIMINGMAX) {
        Rate = LEADINGTIMINGINIT;
       } else {
            /* Do nothing */
       }
    TMRB_ChangeLeadingTiming(TSB_TB7,  LeadingTiming[Rate]);      /*  switch  is
High again */
```

デューティの算出方法:
Trailing Timing = 1ms, ftmrb = 1/8 fphiT0 = 2.5MHz, Ttmrb = 0.4us (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 1000*10% = 100us, Low 幅は 1000-100 = 900us, カウンタ値 = 900us/Ttmrb = 0x8CA

LeadingTiming = 25%: High 幅は 1000*25% = 250us, Low 幅は 1000-250 = 750us, カ

ウンタ値= 750us/Ttmrb = 0x753U

LeadingTiming = 50%: High 幅は 1000*50% = 500us, Low 幅は 1000-500 = 500us, カウンタ値= 500us/Ttmrb = 0x4E2U

Duty = 75%: High 幅は 1000*75% = 750us, Low 幅は 1000-750 = 250us, カウンタ値 = 250us/Ttmrb = 0x271U

Duty = 90%: High 幅は 1000*90% = 900us, Low 幅は 1000-900 = 100us, カウンタ値 = 100us/Ttmrb = 0xFAU

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] = { 0x8CAU, 0x753U, 0x4E2U, 0x271U, 0xFAU };
/* leadingtiming: 10%, 25%, 50%, 75%, 90% */
```

7-14 TMRD

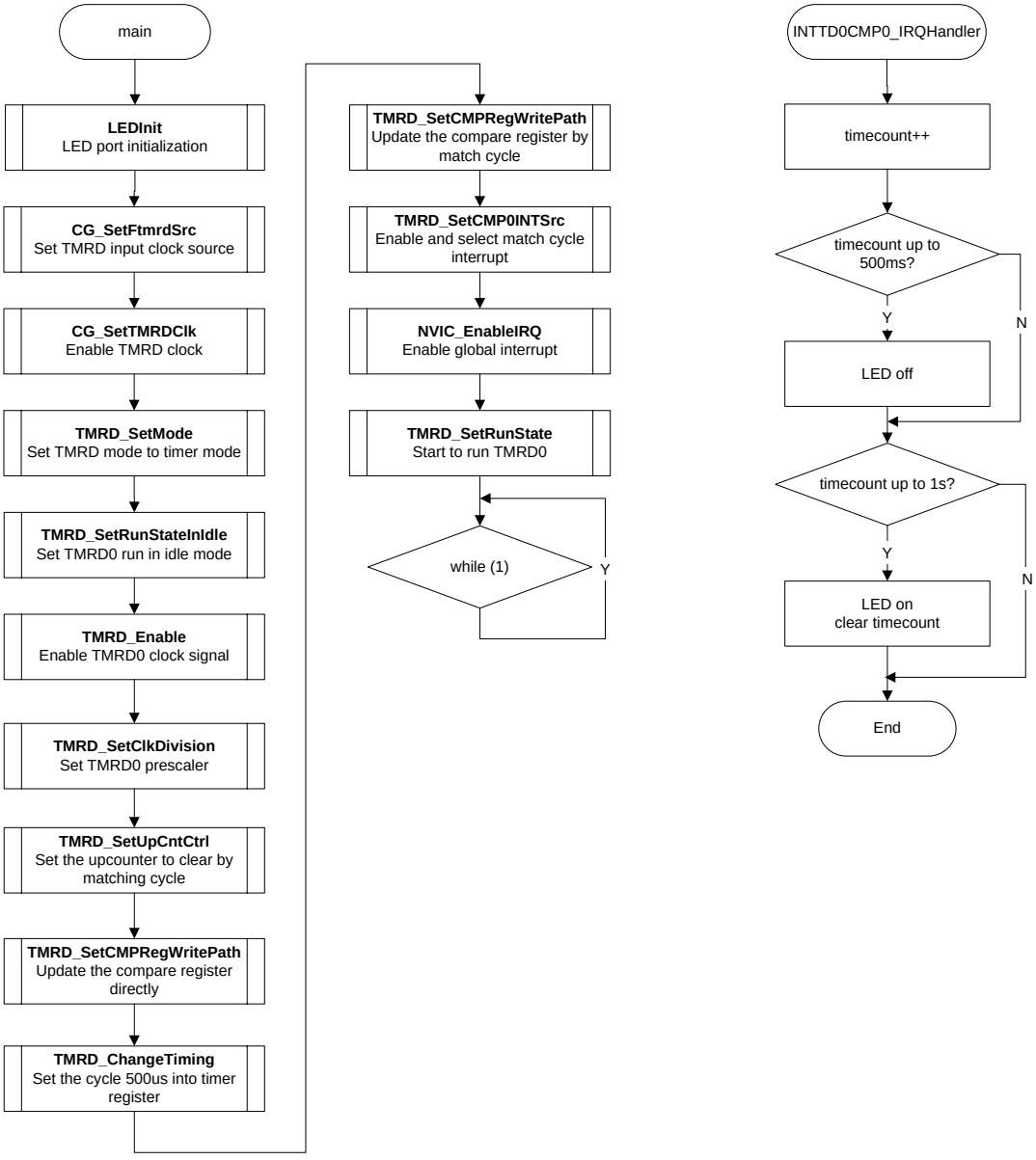
7-14-1 インターバルタイマ

ペリフェラルドライバ(TMRD, CG) のプログラムサンプルです。

この例では以下を行います。

1. TMRD の汎用タイマモード
2. TMRD の初期化
3. TMRD 割り込み処理

- **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 の初期化を行います。

```
    LEDInit();
```

CG_SetFtmrdSrc()をコールして TMRD のクロックを設定し、次に CG_SetTMRDClk()をコールして TMRD クロックを有効にします。

```
    /* TMRD CG setting */
    CG_SetFtmrdSrc(CG_FTMRD_SRC_FPLL);        /* clock is set to fpll */
    CG_SetTMRDClk(ENABLE);                    /* enable the TMRDCLK */
```

TMRD_SetMode()をコールして、TMRD モードを設定し、TMRD0 と TMRD1 を設定します。

```
    /* TMRD mode setting */
    TMRD_SetMode(TMRD_MODE_BOTH_TMR);
```

TMRD_SetRunStateInIdle()をコールして、TMRD0 を IDLE モードで動作させます。

```
    /* TMRD IDLE mode */
```

```
    TMRD_SetRunStateInIdle(TSB_TD0, TMRD_RUN);
```

次に、TMRD_Enable()をコールして TMRD0 クロックを許可し、TMRD_SetClkDivision()をコールすることでプリスケーラを設定します。

```
    /* TMRD enable clock signal */
    TMRD_Enable(TSB_TD0);

    /* Set the TMRD prescaler */
    TMRD_SetClkDivision(TSB_TD0, TMRD_CLK_DIV_1);
```

サイクルと一致すると、TMRD0 のアップカウンタをクリアするように設定します(自動クリア)。

```
    /* Set the upcounter clear mode */
    TMRD_SetUpCntCtrl(TSB_TD0, TMRD_AUTO_CLEAR);
```

直接コンペアレジスタを更新するため、TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_DIRECT)をコールします。TMRD_ChangeTiming()をコールしてデータをタイマレジスタ書き込む時に、対応するコンペアレジスタに同値を書き込まれます。

以下は、マクロ TIME_500US の演算方法です。
fosc = 10MHz(外部高速クロック)
fc = 8 * fosc = 80MHz(サンプルプログラムでは、PLL 多重化の値は、8 逓倍です。)
ftmrd = fpll = 80MHz(サンプルプログラムでは、ftmrd = fpll です。詳細は CG_SetFtmrdSrc()を参照してください)
上記のため、TIME_500US の値は、500us*ftmrd = 40000 = 0x9C40

```
    /* Update the compare register directly */
    TMRD_SetCMPRegWritePath(TSB_TD0, TMRD_CMP_WRITE_DIRECT);

    /* Set the value to timer register */
    TMRD_ChangeTiming(TMRD_TIMING_TD0_CYCLE, TIME_500US); /* 500us */
```

次に、API TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_INDIRECT)使用し、アップカウンタがサイクル時間と一致した際のタイマーレジスタからコンペアレジスタへのデータ書き込みをイネーブルにします。

```
    /* Indirct update the compare register */
    TMRD_SetCMPRegWritePath(TSB_TD0, TMRD_CMP_WRITE_INDIRECT);
```

その後、周期一致とのコンペア 0 割り込みソースを API TMRD_SetCMP0INTSrc()を使用し、選択します。NVIC_EnableIRQ()でコンペア 0 割り込みをイネーブルにします。

```
    /* Enable TMRD intterrupt */
    TMRD_SetCMP0INTSrc(TSB_TD0, TMRD_INT_MATCH_CYCLE);

    NVIC_EnableIRQ(INTTD0CMP0_IRQn);
```

最後に、TMRD_SetRunState() を使用して、TMRD0 を実行、カウンタを開始し、デッドループに入ります。

```
    /* Start to run TMRD */
    TMRD_SetRunState(TSB_TD0, TMRD_RUN);
```

500us を過ぎると、コンペア 0 割り込みが発生します。IRQ INTTD0CMP0_IRQHandler()にて、累積用の *timecount* 変数を設定します。500ms を過ぎると、LED を消灯します。1s が過ぎると、再度 LED を点灯、*timecount* をクリアし、始めから累積します。

```
void INTTD0CMP0_IRQHandler(void)
{
    static uint16_t timecount = 0U;

    timecount++;
```

```
    if (timecount == 1000U) {     /* 500ms */
        LedOff(0x00U);
    } else if (timecount == 2000U) {       /* 1s */
        timecount = 0U;
        LedOn(0x02U);
    } else {
        /* Do nothing */
    }
}
```

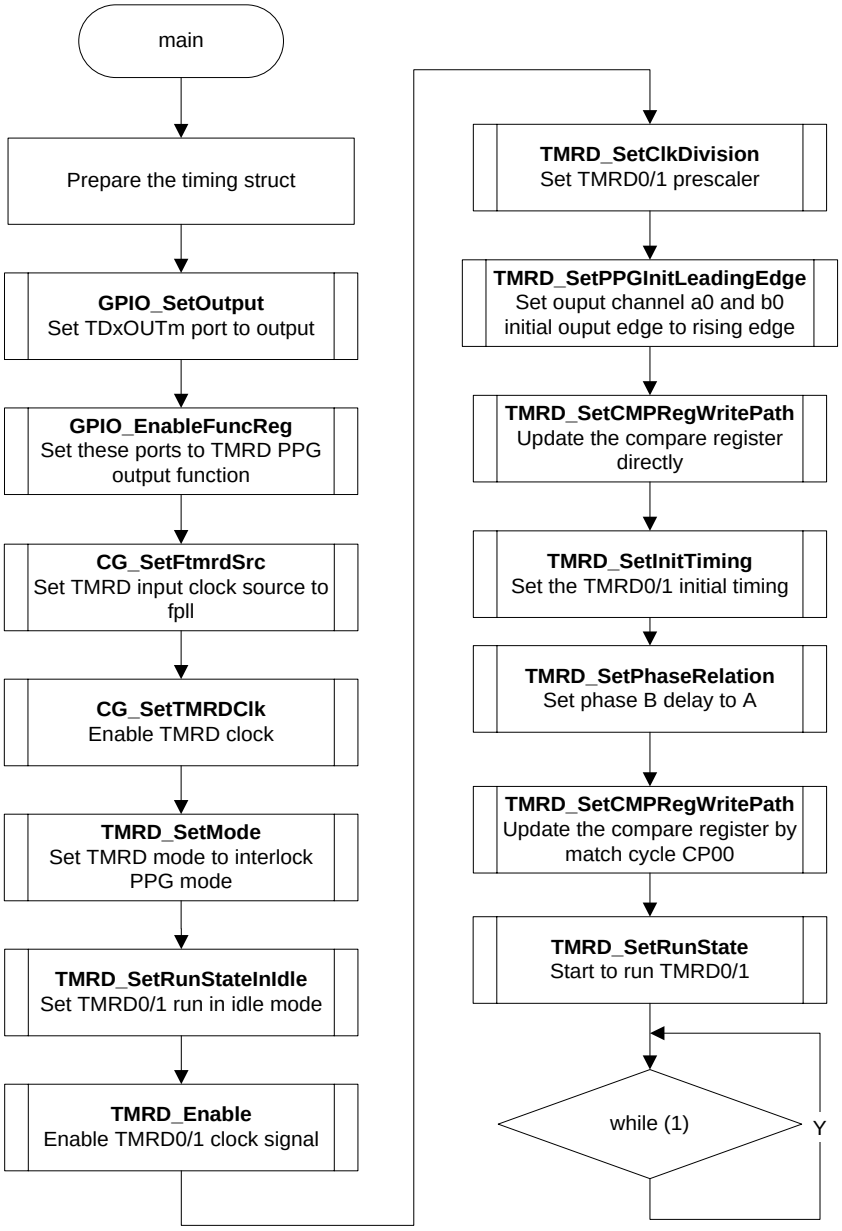
7-14-2 連動 PPG 出力

ペリフェラルドライバ (TMRD, CG, 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. TMRD 連動 PPG モード
2. TMRD 初期化と動作プロセス
3. TMRD PPG モードパラメータ演算方法

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、TMRD パラメータ構造を初期化します。方法は以下のようになります。

500us のサイクル:

fosc = 10MHz(お外部高速クロックの場合)
fpll = 8 * fosc = 80MHz(このサンプルプログラムでは PLL の多重化の値は 8 逓倍です。)
ftmrd = fpll = 80MHz(この例では、ftmrd = fpll です。詳細は CG_SetFtmrdSrc()を参照してください)
設定値は 500us*ftmrd = 40000 = 0x9C40U です。

50%のデューティ:

LeadingTiming0 は 0x0000 に、TrailingTiming は Cycle/2=0x4D20(250us)に設定してください。
LeadinTiming1 と TrailingTiming1 は、使用しませんので、これらの値を 0x0000 にしてください。

位相シフト 120 度:
位相シフトは、下記の公式で演算できます。

θ = 360 degree*(PhaseShiftTiming/(Cycle+1))

このサンプルプログラムでは PhaseShiftTiming は、Cycle/3 = 0x3415(167us)に設定します。

```
#define TIME_CYCLE                ((uint16_t)0x9C40U)     /* 500us */
#define TIME_DUTY                 ((uint16_t)0x4E20U)     /* 250us */
#define TIME_PHASE_SHIFT          ((uint16_t)0x3415U)     /* 500/3= 167us */
…
TMRD_TimingTypeDef timestruct = { 0U };

timestruct.Cycle = TIME_CYCLE;
timestruct.TrailingTiming0 = TIME_DUTY;
timestruct.PhaseShiftTiming = TIME_PHASE_SHIFT;
```

このサンプルプログラムでは、TMRD PPG 出力への TDxOUTm ポートは、ポート E 4,5,6,7 に設定します。

```
/* TD0OUT0, TD0OUT1, TD1OUT0, TD1OUT1 port setting */
GPIO_SetOutput(GPIO_PE, GPIO_BIT_4 | GPIO_BIT_5 | GPIO_BIT_6 |
                    GPIO_BIT_7);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_3, GPIO_BIT_4 |
                        GPIO_BIT_5 | GPIO_BIT_6 | GPIO_BIT_7);
```

CG_SetFtmrdSrc()をコールして、TMRD クロックを指定します。次に CG_SetTMRDClk()をコールして、ソースクロックを有効にします。

```
/* TMRD CG setting */
CG_SetFtmrdSrc(CG_FTMRD_SRC_FPLL);          /* clock is set to fpll */
CG_SetTMRDClk(ENABLE);                      /* enable the TMRDCLK */
```

TMRD_SetMode()を使用して、TMRD モードを設定します。このサンプルプログラムでは、連動 PPG モードを使用します。

```
/* TMRD mode setting */
TMRD_SetMode(TMRD_MODE_INTERLOCK_PPG);
```

TMRD_SetRunStateInIdle()を使用して、TMRD0/1 を IDLE モードにします。

```
/* TMRD IDLE mode */
TMRD_SetRunStateInIdle(TSB_TD0, TMRD_RUN);
TMRD_SetRunStateInIdle(TSB_TD1, TMRD_RUN);
```

その後、TMRD クロック信号をイネーブルにし、API TMRD_Enable() と TMRD_SetClkDivision()を使用して、プリスケーラを設定します。連動 PPG モードにおいて、TMRD1 のプリスケーラは、TMRD0 の設定値と同じです。TMRD0 プリスケーラ設定のみ行ってください。

```
/* TMRD enable clock signal */
TMRD_Enable(TSB_TD0);
TMRD_Enable(TSB_TD1);
/* Set the TMRD prescaler */
TMRD_SetClkDivision(TSB_TD0, TMRD_CLK_DIV_1);
```

PPG 出力初期波形エッジを設定します。チャネル a0 と b0 の両方を TMRD_SetPPGInitLeadingEdge()を使用して立ち上がりエッジに設定します。

```
/* Set the PPG output edge */
TMRD_SetPPGInitLeadingEdge(TMRD_PPG_CHANNEL_A0,
                                TMRD_WAVE_EDGE_RISING);
TMRD_SetPPGInitLeadingEdge(TMRD_PPG_CHANNEL_B0,
```

```
                                        TMRD_WAVE_EDGE_RISING);
```

TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_DIRECT)を使用した後、コンペアレジスタを直接更新します。同値は、TMRD_SetInitTiming ()を使用して、データがタイマレジスタに書き込まれた場合、対応するコンペアレジスタに直接書き込まれます。タイミング構造体は、本章の開始部分を参照してください。

```
/* Direct update the compare register */
TMRD_SetCMPRegWritePath(TSB_TD0, TMRD_CMP_WRITE_DIRECT);
TMRD_SetCMPRegWritePath(TSB_TD1, TMRD_CMP_WRITE_DIRECT);

/* Set the value to timer register */
TMRD_SetInitTiming(TSB_TD0, &timestruct);
TMRD_SetInitTiming(TSB_TD1, &timestruct);
```

その後 TMRD_SetPhaseRelation()を使用し、位相 A と B の位相関係を設定します。このサンプルプログラムでは、位相 B は位相 A より遅延します。

```
/* Set the relation of phase A and B */
TMRD_SetPhaseRelation(TMRD_PHASE_DELAY_OR_SAME);
```

次に TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_INDIRECT)を使用して、アップカウンタがサイクル時間と一致時のタイマレジスタからコンペアレジスタへの書き込みをイネーブルにします。

```
/* Indirct update the compare register */
TMRD_SetCMPRegWritePath(TSB_TD0, TMRD_CMP_WRITE_INDIRECT);
TMRD_SetCMPRegWritePath(TSB_TD1, TMRD_CMP_WRITE_INDIRECT);
```

最後に、TMRD_SetRunState()を使用して、TMRD0/1 を実行し TMRD0 を開始し、デッドループに入ります。連動 PPG モードにて、TMRD1 は TMRD0 の COUNTER0 と連動して動作を開始します。TMRD_SetRunState()により TMRD1 の実行を設定することは、無効となります。そのため、動作させるために TMRD1 を設定する必要はありません。

```
/* Start to run TMRD */
TMRD_SetRunState(TSB_TD0, TMRD_RUN);
```

7-15 WDT

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

この例では以下を行います。

1. WDT の初期化
2. DEMO1 では、オーバーフロー前に WDT クリアを行わず、NMI 割り込みを発生させます。
3. DEMO2 では、オーバーフロー前に WDT クリアを行い、常時 LED を点滅させます。

• サンプルプログラムのコードと説明

以下のコードは WDT の初期化の例です。検出時間が 2^25/fsys に設定されオーバーフロー時に NMI 割り込みを発生します。

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

WDT を初期化し、その後 WDT を有効にします。

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

DEMO1 では、NMI 割り込みの発生を待ちます。

```
while(1)
{
    if (fIntNMI == 1U) {
        fIntNMI = 0U;
        /* Do something here */
    }else {
        /* Do nothing */
    }
}
```

DEMO1 では、NMI 割り込み発生時に WDT を禁止にし、LED の点滅を停止します。

```
WDT_Disable();
```

DEMO2 では、WDT クリアを行い、常に LED を点滅させます。

```
WDT_WriteClearCode();
```