TOSHIBA

TX04 ペリフェラルドライバ使用例 (TMPM440)

第一版

2017 年9月

東芝デバイス＆ストレージ株式会社

CMDR-M440UE-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。

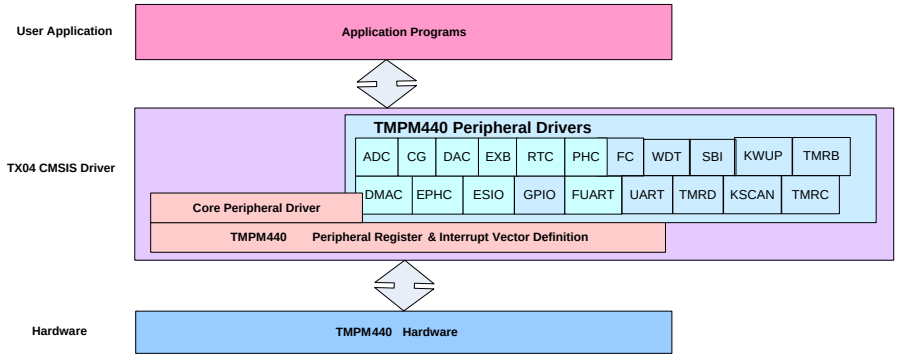
目次

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TMPM440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

TX04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
ADC	ユニット A	使用(ADC サンプル)
	ユニット B	未使用
	ユニット C	未使用
CG	-	使用(CG サンプル)
低消費電力モード	-	IDLE/STOP1/STOP2 モード
外部割込み	-	未使用
シリアルチャネル	SIO0	使用(UART リターゲットサンプル、DMAC_UART サンプル、SIO サンプル)
	SIO1	使用(DMAC_UART サンプル、SIO サンプル)
	上記以外のチャネル	未使用
TMRB	TMRB0	使用(TMRB: 汎用タイマ)
	TMRB12	使用(TMRB: PPG 出力)
	上記以外の TMRB	未使用
WDT	WDT	使用(WDT サンプル)
PHC	2 チャネル	使用(PHC0 サンプル)
SBI	SIB0	使用(SBI SLAVE サンプル)
DMAC	DMACA	使用(DMAC_UART サンプル)
	上記以外のユニット	未使用
PSC	-	使用(PSC サンプル)

4 端子用途

本サンプルプログラムは、TMPM440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。

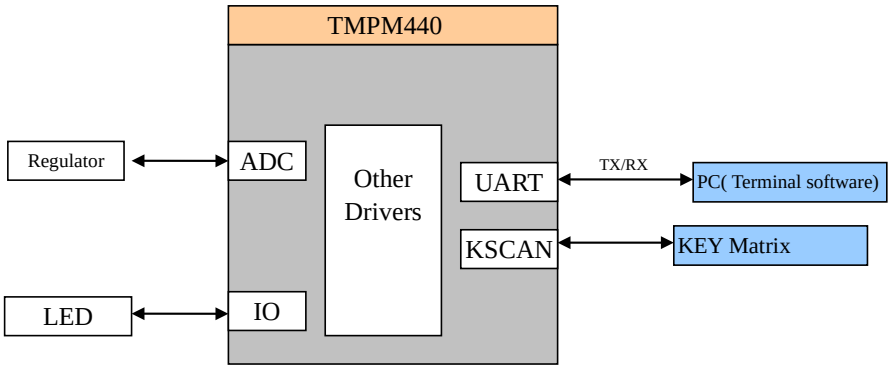
端子 No.	機能名	用途
A7	PAD0	SW0
C8	PAD1	SW1
B7	PAD2	SW2
A6	PAD3	SW3
G7	PAD4	SW4
C6	PAD5	SW5
B7	PAD6	SW6
A5	PAD7	SW7
M19	PT0	LED0
M18	PT1	LED1
N20	PT2	LED2
K14	PT3	LED3
N19	PT4	LED4
L15	PT5	LED5
N18	PT6	LED6
P20	PT7	LED7
L3	PU2	PHC サンプル 出力 0
M6	PU3	PHC サンプル 出力 1
L1	PU4	PHC0IN0
M2	PU5	PHC0IN1
L19	PY4	TD0OUT0
L18	PY5	TD1OUT0
K3	DAAOUT	DAAOUT
W17	PA0	EBIF データバス D0
Y18	PA1	EBIF データバス D1
V18	PA2	EBIF データバス D2
U18	PA3	EBIF データバス D3
W19	PA4	EBIF データバス D4
Y19	PA5	EBIF データバス D5
W20	PA6	EBIF データバス D6
W18	PA7	EBIF データバス D7
V14	PB0	EBIF データバス D8
W14	PB1	EBIF データバス D9
P13	PB2	EBIF データバス D10
R14	PB3	EBIF データバス D11
E20	PH4	UART0 送信
F20	PH5	UART0 受信
A14	PK4	UART1 送信
A13	PK5	UART1 受信
G20	PH6	SIO SCLK0
A15	PK6	SIO SCLK1

端子 No	機能名	用途
B15	PW4	TMRB12 出力
Y15	PB4	EBIF データバス D12
V15	PB5	EBIF データバス D13
W15	PB6	EBIF データバス D14
P14	PB7	EBIF データバス D15
V12	PC0	EBIF アドレスバス A0
W12	PC1	EBIF アドレスバス A1
R12	PC2	EBIF アドレスバス A2
Y13	PC3	EBIF アドレスバス A3
V13	PC4	EBIF アドレスバス A4
W13	PC5	EBIF アドレスバス A5
R13	PC6	EBIF アドレスバス A6
Y14	PC7	EBIF アドレスバス A7
R18	PD0	EBIF アドレスバス A8
N15	PD1	EBIF アドレスバス A9
T19	PD2	EBIF アドレスバス A10
U20	PD3	EBIF アドレスバス A11
T18	PD4	EBIF アドレスバス A12
U19	PD5	EBIF アドレスバス A13
V20	PD6	EBIF アドレスバス A14
V19	PD7	EBIF アドレスバス A15
L14	PE0	EBIF アドレスバス A16
P19	PE1	EBIF アドレスバス A17
M15	PE2	EBIF アドレスバス A18
R20	PE3	EBIF アドレスバス A19
P18	PE4	EBIF アドレスバス A20
M14	PE5	EBIF アドレスバス A21
R19	PE6	EBIF アドレスバス A22
T20	PE7	EBIF アドレスバス A23
N14	PF0	EBIF バス制御 RD
Y16	PF1	EBIF バス制御 WR
V16	PF2	EBIF バス制御 BELL
R15	PF3	EBIF バス制御 BELH
W16	PF5	EBIF バス制御 CS0
H19	PR2	EPHC サンプル 出力 0
H15	PR3	EPHC サンプル 出力 1
K20	PR6	EPHC0IN0
K19	PR7	EPHC0IN1
J19	PR4	SBI SLAVE サンプル SCL0
J18	PR5	SBI SLAVE サンプル SDA0

Pin No.	Name	Usage
C3	PAA0	AINA0
H18	PR0	FUART TXD6
H20	PR1	FUART RXD6
H19	PR2	FUART CTS6
H15	PR3	FUART RTS6

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:
   TMPM440 評価ボード

2. 開発ツール:
   - IAR 社:
     1) J-Link: IAR J-Link-7.0/8.0
     2) IDE:IAR Embed Workbench for ARM version 6.40.2
   - KEIL 社:
     1) u-Link: Realview ULINK2
     2) IDE:KEIL uVision MDK version 4.54

   補足: PSC サンプルプログラムの開発環境は以下です。
   - IAR 社:
     1) J-Link: IAR J-Link-ARM 7.0/ J-Link 6.0
     2) IDE: IAR Embed Workbench 6.50.1
   - KEIL 社:
     3) IDE:KEIL uVision MDK version 4.60

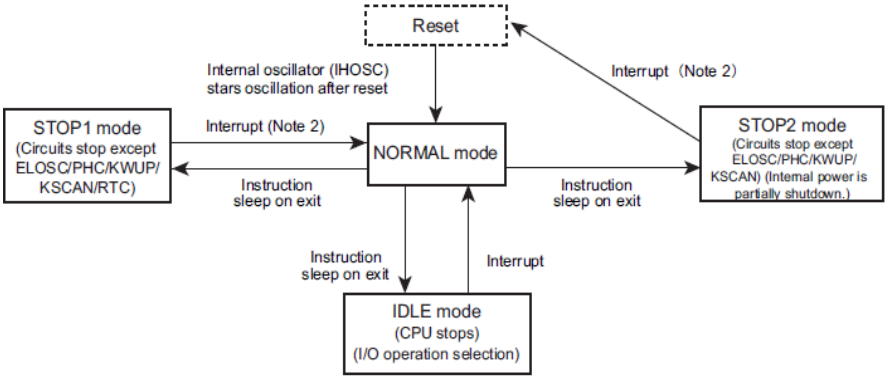
6 機能説明

動作モード

TMPM440 には 4 つの動作モードがあります: NORMAL, IDLE, STOP1, STOP2 モード
IDLE と STOP モードは低消費電力モードです。

- **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。
以下に、モードの変遷図を表します。



6-1 ADC

6-1-1 ADC データリード

PAA1/AIN01 に接続されたポテンションメータの値を変更します。測定された電圧値により、LED の点滅間隔を変更します。出力電圧が高くなるほど点滅間隔が短くなります。

6-2 CG

6-2-1 NORMAL<->STOP1 モード変更

CPU 動作モードを変更するサンプルプログラムです。NORMAL モードと STOP1 モードの 2 つのモードを使用します。

現在のモード	アクション (Key)	動作	LED 表示
NORMAL	Key1 を押す (Low アクティブ)	NORMAL→ STOP1	LED1,2,3,4 オフ
STOP1	Key2 を押す (Low アクティブ)	STOP1 →NORMAL	LED1,2,3,4 オン

補足:
Key1 は R11 端子に接続されており、PP6 を内部プルアップに設定し、Low アクティブで STOP モードに入るために使用します。
Key2 は Y12 端子に接続されており、PP7 を内部プルアップに設定し、Low アクティブで STOP モードを解除する割り込み要因 INT0 として使用します。

6-2-2 NORMAL<->STOP2 モード変更

NORMAL モードと STOP2 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	LED 表示
NORMAL	SW0 を OFF	NORMAL→ STOP2	LED7 オフ
STOP2	まず SW0 を ON し、次に外部割り込みを発生させる SW*を ON します。	STOP2→NORMAL	LED7 オン

補足:
外部割り込みを発生させる SW はキーマトリクス(KSOUT0×KSIN0～7)です。

6-2-3 NORMAL<->IDLE モード変更

NORMAL モードと IDLE モードの 2 つのモードを使用します。

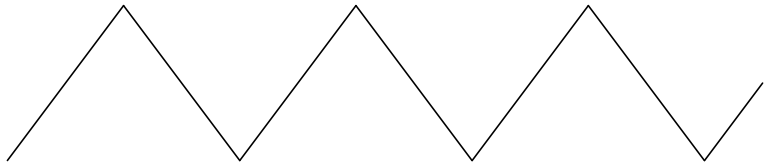
現在のモード	アクション (Key)	動作	LED 表示
NORMAL	SW0 を OFF	NORMAL→IDLE	LED7 オフ
IDLE	まず SW0 を ON し、次に外部割り込みを発生させる SW*を ON します。	IDLE→NORMAL	LED7 オン

補足:
外部割り込みを発生させる SW はキーマトリクス(KSOUT0×KSIN0～7)です。

6-3 DAC

6-3-1 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。



6-4 DMAC

6-4-1 メモリから周辺回路

UART0 から UART1 へのデータ転送を行う処理が実装されています。“TOSHIBA” の文字列データを UART0 から UART1 へ送信されます。
DMAC により、RAM から UART0 データレジスタにデータが送信され、文字列の各文字データは UART0 経由で送信され、UART1 で受信します。UART1 のデータ受信後、データは文字配列に保存されます。

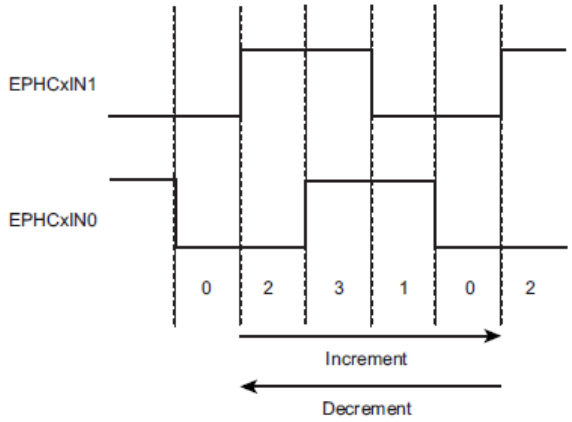
デバッガの変数ウィンドウに文字配列変数を登録して、受信されたデータを確認します。

補足: サンプルソフトウェアを使用するには、TMPM440 評価ボードの設定変更を行います:
UART0(E20, PH4)は送信側、UART1(A13, PK5)は受信側としてワイヤ接続します。

6-5 EPHC

6-5-1 EPHC カウンタ

EPHC の 4 逓倍モードを使用したサンプルプログラムです。
GPIO を使用し、2 相入力 EPHC0IN0 および EPHC0IN1 を同時に入力します。
入力波形は以下のような図 (EPHCxCNT<MA2DIR>="1" (負方向)) になります。



アップ/ダウンカウンタは、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。
カウンターの結果は、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:
EPHC 端子:
PR6 (EPHC0IN0)
PR7 (EPHC0IN1)

PR6 端子 (k20 EPHC0IN0 端子)と PR2 端子 (H19 PortR 出力端子)を接続します。
PR7 端子 (k10 EPHC0IN1 端子)と PR3 端子(H15 PortR 出力端子)を接続します。

6-6 ESIO

6-6-1 基本転送

ESIO の基本機能を使用したサンプルプログラムです。
ESIO0 上でシングル転送を使用しています(TX と RX を接続します)。データ送信が行われると LED が点滅します。ユーザーは BITRATE_MIN を定義することで点滅状況を確認することができます。

補足:
A8 端子(PJ0, EXIO_TX0)と B8 端子(PJ4, ESIO_RX0)を接続します。

6-7 EXB

6-7-1 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。接続端子については**補足**を参照してください。

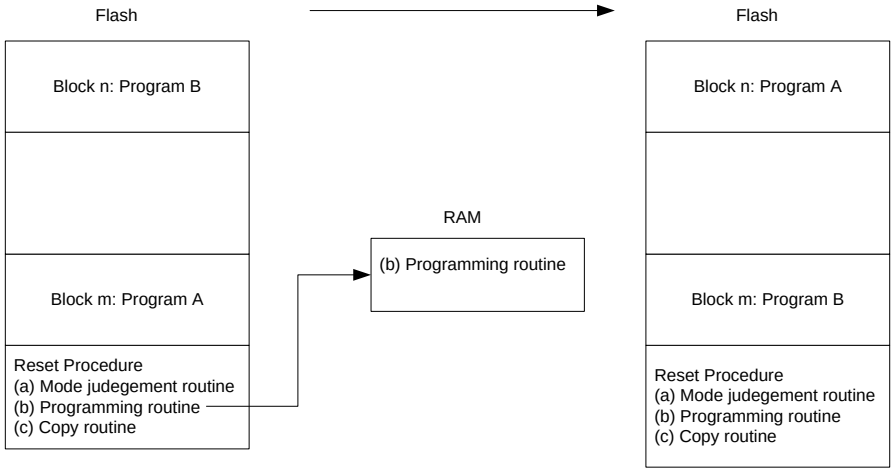
補足:
1. TMPM440 評価ボードの AD[15:0]と IS62WV51216BLL SRAM の AD[15:0]を接続します。
2. TMPM440 評価ボードの A16 と IS62WV51216BLL SRAM の A15 を接続します。
3. TMPM440 評価ボードの CS0、WE、OE、ALE、BELLn と IS62WV51216BLL SRAM の CS0、WE、OE、ALE、BELLn を接続します。

6-8 FC

6-8-1 Flash スワップ

Flash のドライバ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ーブートモードで実行します。
ーリセット動作: モード判定、書き込みルーチン(フラッシュ API)、転送ルーチンで構成されます。
・モード判定ルーチン: ユーザブートモードまたはノーマルモードの判定を行います。

・転送ルーチン: 書き込みルーチンを Flash から RAM へ転送します。
・書き込みルーチン: RAM 上で動作し、フラッシュ上のプログラム A とプログラム B のコードを入れ替えます。
　　ープログラム A/B (リセット動作)は事前にフラッシュメモリに書き込まれています。
　　ープログラム A/B (A: 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SW0 キーはリセット動作のモード判定に使います。
　SW0 キーが押されていない場合 → Normal モードです。
　SW0 キーが押された場合 → ユーザブートモードです。

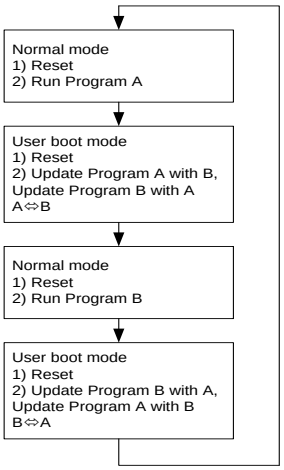


動作シーケンス

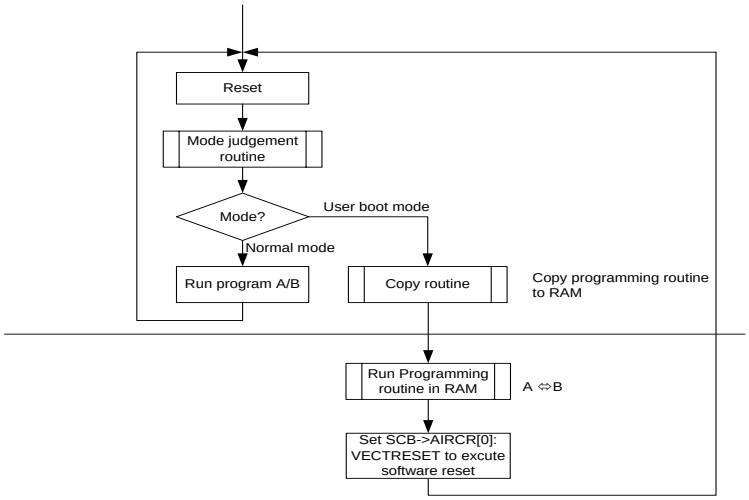
(1) 電源投入
SW0 を OFF した状態で電源投入またはリセット端子を ON してください。まずプログラム A が実行され、LED 0 が点滅します。

(2) SW0 を ON している間にリセット端子を ON してください。
LED2 が点灯したら SW0 を OFF します。

(3) フラッシュ内のプログラム A と B が入れ替わると、プログラム B は自動的にリセットし、
LED1 が点滅します。



・サンプルプログラムのフローチャート



ユーザブートモードに入ると、LED2 が点灯します。

6-9 FUART

6-9-1 ループバック

Full UART のドライバを使用して、Full UART 送信と FIFO 受信を行うサンプルプログラムです。

本プログラムは 64 種の異なるデータを Full UART チャネル 6 の TXD6 端子から送信し、RXD6 端子からデータを受信します。トグルスイッチ SW0 がオンの場合、本プログラムは受信 FIFO からのデータ読み出しを開始します。データの読み出しは受信 FIFO が空になるまで止まりません。トグルスイッチ SW0 が数回オフ、オンとなった場合、本プログラムは RXD6 が受信した全データの読み出しを完了しています。その後プログラムは全受信データを送信データと比較します。
受信データが送信データと同一の場合、LED6 が点灯します。
受信データが送信データと異なる場合、LED5 が点灯します。
本プログラムは、ハードウェアフロー制御機能の確認のため 2 回実行することができます。
1 回目:
受信データが送信データと同一の場合 RTS と CTS ハードウェアフロー制御を有効にしてください。
2 回目:
受信データが送信データと異なる場合 RTS と CTS ハードウェアフロー制御を無効にしてください。

補足:
TX6 端子 (H18) と RX6 端子 (H20) を接続してください。
RTS6 (H15) 端子と CTS6 端子 (H19) を接続してください。

6-10 GPIO

6-10-1 GPIO による LED 制御

GPIO のドライバを使用し、スイッチの ON/OFF で LED が点灯/消灯するサンプルプログラムです。

6-11 KSCAN

6-11-1 KSCAN サンプル

KSCAN と GPIO のドライバを使用して、スイッチ 0~7 を 1 つずつ押すと、LED0~LED7 が点灯するサンプルプログラムです。

6-12 KWUP

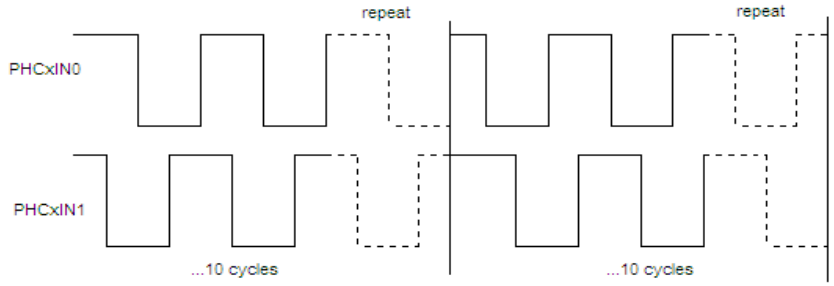
6-12-1 低消費電力モードの解除

KWUP と GPIO のドライバを使用して、スイッチ 1 (PAD1)を ON すると、CPU は低消費電力モードとなり(LED は消灯します)、スイッチ 0(PAD0)を ON すると、低消費電力モードが解除されるサンプルプログラムです。

6-13 PHC

6-13-1 PHC カウンタ

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。
カウンターの結果は、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:
本サンプルソフトウェアを実行するためには、セッティング済みの TMPM440 評価ボードが必要です。

PHC 端子:
PU4 端子 (PHC0IN0)
PU5 端子 (PHC0IN1)

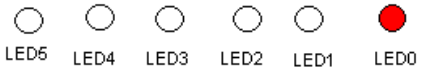
PU4 端子 (PHC0IN0) と PU2 端子 (PortU のビット 2 出力端子)を接続します。
PU5 端子 (PHC0IN1) と PU3 端子 (PortU のビット 3 出力端子)を接続します。

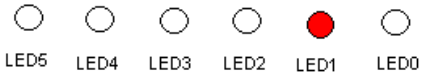
6-14 RTC

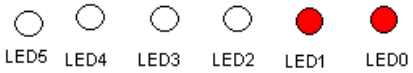
6-14-1 RTC の基本動作

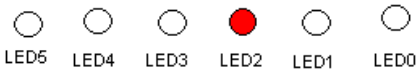
2 段目の LED を表示するには、内部の RTC を使用してください。

LED 表示例

1s: 

2s: 

3s: 

4s: 

……

59s (hex: 0x3B = 0b00111011)

6-15 SBI

6-15-1 SBI スレーブ

I2C バスマスターとスレーブモードを使用したサンプルプログラムです。

2 つの評価ボード上で 2 つの I2C バスを接続します。片方の評価ボードはマスターとして使用し、もう片方の評価ボードはスレーブとして使用します。
SBI 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 (SBI0) は、“TOSHIBA”の文字を I2C マスタ (SBI0) から受信します。
受信データは変数 gI2CrxData[]をデバッガに登録することで確認できます。

補足: 2 つの評価ボード間でデータ送信を行うために、下記のような端子接続を行ってください。
2 つの PR4 端子 (SCL) を接続します。
2 つの PR5 端子 (SDA) を接続します。
2 つの GND 端子は同じグループにします。

6-16 TMRB

6-16-1 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。
このタイマーを使い 1s(500ms でオン、500ms でオフ)間隔で LED 点滅を行います。

6-16-2 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、UART に出力します。
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 を切り替えるを押すことでデューティーを変更します。
10% → 25% → 50% → 75% → 90% → 10%

6-17 TMRC

6-17-1 アップカウンタ

TMRC は 32 ビットのバイナリカウンターです。カウンター値は、ソフトウェアによりキャプチャレジスタへ挿入されます。TCTBTRUN<TBTCAP>へ"1"が書き込まれるたびに、そのときのカウンター値は TCTBTCP レジスタへ挿入されます。カウンターがオーバーフローすると、オーバーフロー割り込み INTTCTBT が発生し、カウンター値は"0"にクリアされた後、カウントを開始します。

6-18 TMRD

6-18-1 汎用タイマ

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 1ms です。
このタイマーを使い 1s(500ms でオン、500ms でオフ)間隔で LED 点滅を行います。

6-18-2 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 PY4/TD0OUT0 と PY6/TD1OUT0 をオシロスコープで確認することができます。

6-19 SIO/UART

6-19-1 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() 関数を用いて、シリアルポートからデータを出力します。

6-19-2 UART FIFO

UART0 から"TMPM4401"というデータを FIFO を使用して UART3 へ送信し、同時に UART3 から "TMPM4402"というデータを FIFO を使用して UART0 へ送信するサンプルプログラムです。
ResetIdx() 関数の前にブレークポイントを設定しておくと、RxBuffer=”TMPM4402” と RxBuffer1="TMPM4401"となった場合にブレークポイントで停止します。

6-19-3 SIO サンプル

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。)

補足: 端子 G20(PH6, SIO0_SCLK)と端子 A15(PK6, SIO1_SCLK)を接続します。
端子 E20(PH4, SIO0_TXD0)と端子 A13(PK5, SIO1_RXD1)を接続します。
端子 A14(PK4, SIO1_TXD1)と端子 F20(PH5, SIO0_RXD0)を接続します。

6-20 WDT

6-20-1 WDT サンプル

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

DEMO1:
タイマーオーバーフロー時に NMI 割り込みを発生し、WDT をクリアします。

DEMO2:
ウォッチドッグタイマ制御レジスタにクリアコードを書き込み、ウォッチドッグタイマカウンタをクリアします。

6-21 PSC

本サンプルプログラムは、DMACを使用してPSCのコードとデータをPSC用コードRAMとデータRAMに転送し、TMRB割り込みを起動トリガとしてPSCを動作させる例です。

7 ソフトウェア

本ソフトウェアは、TMPM440 の主要機能を TMPM440 評価ボード上で動作確認するためのサンプルプログラムです。

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

ワークスペース構造とプロジェクト名は、以下の通りです:

```
\---TMPM440
    +---Libraries
    |   +---TX04_CMSIS
    |   |   |   system_TMPM440.c
    |   |   |   system_TMPM440.h
    |   |   |   TMPM440.h
    |   |   \---startup
    |   |       +---arm
    |   |       |       startup_TMPM440.s
    |   |       \---iar
    |   |               startup_TMPM440.s
    |   \---TX04_Periph_Driver
    |       +---inc
    |       |       tmpm440_adc.h
    |       |       tmpm440_cg.h
    |       |       tmpm440_dac.h
    |       |       tmpm440_dmac.h
    |       |       tmpm440_ephc.h
    |       |       tmpm440_esio.h
    |       |       tmpm440_exb.h
    |       |       tmpm440_fc.h
    |       |       tmpm440_fuart.h
    |       |       tmpm440_gpio.h
    |       |       tmpm440_kscan.h
    |       |       tmpm440_kwup.h
    |       |       tmpm440_phc.h
    |       |       tmpm440_rtc.h
    |       |       tmpm440_sbi.h
    |       |       tmpm440_tmrb.h
    |       |       tmpm440_tmrc.h
    |       |       tmpm440_tmrd.h
    |       |       tmpm440_uart.h
```

```
|       |       tmpm440_wdt.h
|       |       tx04_common.h
|       \---src
|       |       tmpm440_adc.c
|       |       tmpm440_cg.c
|       |       tmpm440_dac.c
|       |       tmpm440_dmac.c
|       |       tmpm440_ephc.c
|       |       tmpm440_esio.c
|       |       tmpm440_exb.c
|       |       tmpm440_fc.c
|       |       tmpm440_fuart.c
|       |       tmpm440_gpio.c
|       |       tmpm440_kscan.c
|       |       tmpm440_kwup.c
|       |       tmpm440_phc.c
|       |       tmpm440_rtc.c
|       |       tmpm440_sbi.c
|       |       tmpm440_tmrb.c
|       |       tmpm440_tmrc.c
|       |       tmpm440_tmrd.c
|       |       tmpm440_uart.c
|       |       tmpm440_wdt.c
\---Project
    +---Examples            //only 1 examples listed here
    |   +---ADC
    |   |   \---ADC_Data_Read
    |   |       +---App
    |   |       |       main.c
    |   |       +---IAR
    |   |       |       ADC_Data_Read.ewd
    |   |       |       ADC_Data_Read.ewp
    |   |       |       ADC_Data_Read.eww
    |   |       \---KEIL
    |   |                ADC_Data_Read.uvopt
    |   |                ADC_Data_Read.uvproj
    |   \---Workspace
    |       +---IAR
    |       |       Examples_for_M440_Driver.eww
    |       \---KEIL
    |                Examples_for_M440_Driver.uvmpw
    \---Template
        +---IAR
```

```
|           TMPM440FEXBG_Flash.icf
\---KEIL
            tmpm440.sct
```

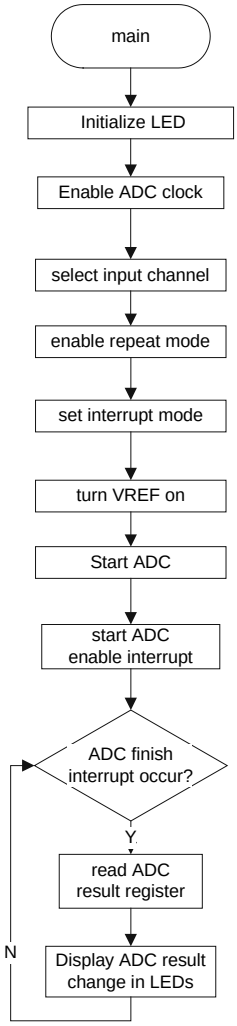
7-1 ADC

7-1-1 例: ADC データリード

TX04 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. ソフトウェアトリガによる AD 変換を開始し、AD 変換結果を読み出します。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず ADC のクロック設定を行い、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 */
CG_SetPLL1ForADC(ENABLE);

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

/* select ADC input channel */
ADC_SetInputChannel(TSB_ADA, ADC_AN_01);
```

```
/* Enable ADC repeat mode */
ADC_SetRepeatMode(TSB_ADA, ENABLE);

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

/* Turn VREF on */
ADC_SetVref(TSB_ADA, 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(TSB_ADA);              /* Start ADC in unit A */

/* enable ADA interrupt */
NVIC_EnableIRQ(INTADA_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(TSB_ADA, ADC_REG_00);
```

7-2 CG

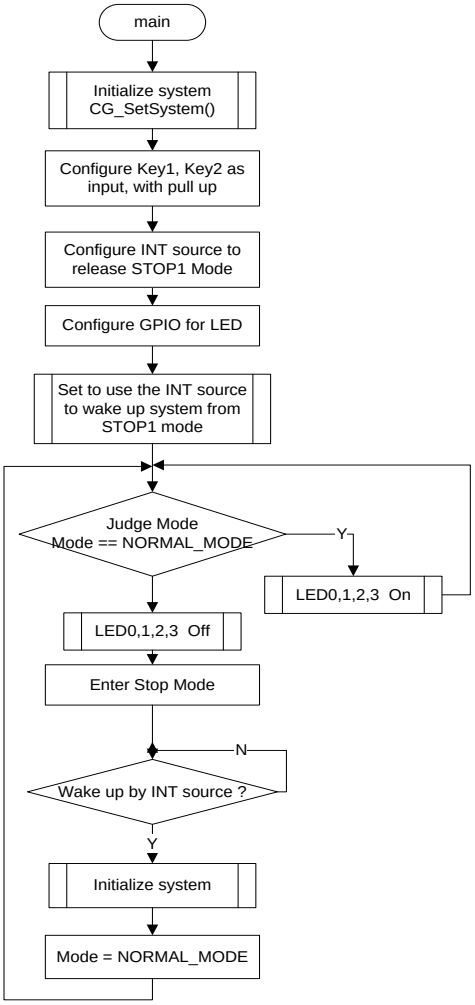
7-2-1 例: NORMAL<->STOP1 モード変更

TX04 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

- **フローチャート**



- **サンプルプログラムのコードと説明**

通常の CG の初期化(リセット後)

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

```
void CG_SetSystem(void)
{
    if (CG_GetFoscSrc()==CG_FOSC_OSC_INT){
        /* Switch over from IHOSC to EHOSC*/
        switchFromIHOSCtoEHOSC();
    }

    /* Set up pll and wait 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 stop */
    CG_SetSTBYMode(CG_STBY_MODE_STOP1);

    /* Set pin status in stop mode to "active" */
    CG_SetPinStateInStop1Mode(ENABLE);
}
```

書式変更

書式変更

スイッチ、LED 端子、外部割込み端子の設定:

スイッチ、LED、外部割り込みの I/O 端子設定を行います。

```
    /* Configure Key1(PP6 ,pinR11), Key2(PP7,pinY12) as input, with pull up */
    TSB_PP->IE |= 0x03U << 6U ;
    TSB_PP->PUP |= 0x03U << 6U ;

    /* Configure INT0 function on pin PP7(Key2) */
    TSB_PP->FR3 |= 0x01U << 7U;

    /* Configure GPIO for LED */
    LED_Init();
```

低消費電力モードを解除するための外部割込み端子設定:

低消費電力モード解除のための外部割り込み INT0 を設定します。保留状態要求割り込みのクリアのあと、INT0 をイネーブルにします。

```
    CG_ClearINTReq(CG_INT_SRC_0);

    /* Set to use INT0 (pin Y12(PP7), pull up and low active) to wake up system from
STOP mode */
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_0,
                            CG_INT_ACTIVE_STATE_FALLING, ENABLE);

    NVIC_ClearPendingIRQ(INT0_IRQn);
    NVIC_EnableIRQ(INT0_IRQn);
```

STOP モード設定:

STOP モードに入る設定を行います。ウォームアップ時間を設定し、__WFI()命令を使用して STOP モードに入ります。

```
/* CG_NormalToStop */
void CG_NormalToStop(void)
{
	/* Set CG module: Normal ->Stop mode */
	CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT,CG_WUODR_EXT);
	/* Enter stop mode */
	__WFI_wait();
}
```

コール元の割り込みが禁止状態である場合があるため、NOP 命令を 8 個挿入します。

```
void __WFI_wait()
{
	__WFI();
	__NOP();
	__NOP();
	__NOP();
	__NOP();
	__NOP();
	__NOP();
	__NOP();
	__NOP();
}
```

マルチクロック回路の許可

まず PLL を設定します。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 CG_EnableClkMulCircuit(void)
{
	Result retval = ERROR;
	WorkState st = BUSY;
	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_FPLL);
	} else {
		/*Do nothing */
	}

	return retval;
}
```

7-2-2　例: NORMAL<->STOP2 モード変更

TX04 ペリフェラル・ドライバ(CG, GPIO, KSCAN)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 STOP2 モードの切り替え方法

- **サンプルプログラムのコードと説明**

スイッチ、LED、KSCAN 用端子の設定を行います。

```
    SW_Init();
    LED_Init();

    configGPIO_KSCAN(ENABLE, DISABLE);   /* Set KSCAN's GPIO */
```

リセット要因が STOP2 モードの解除でなければ、KSCAN の初期化と低消費電力モード解除用割り込みの設定を行い、KSCAN 動作を開始します。なお、この例では低速クロックを KSCAN 動作クロックとして使用します。

```
    configCG_FS(WARMTIME);

    KSCAN_SetSCLK(KSCAN_KSCL_FS);     /*select fs for ksclk */

    configKSCAN();                /* KSCAN initial */

    CG_ClearINTReq(CG_INT_SRC_INTKSCAN);
    NVIC_ClearPendingIRQ(INTKSCAN_IRQn);

    NVIC_EnableIRQ(INTKSCAN_IRQn);

    KSCAN_SetINTReq(ENABLE);

    startKSCAN(ENABLE);
```

リセット要因が STOP2 モードの解除であれば、低消費電力モード解除用割り込みの設定を行い、ポートキープを解除します。

```
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_INTKSCAN,
CG_INT_ACTIVE_STATE_RISING, DISABLE);

    NVIC_EnableIRQ(INTKSCAN_IRQn);

    CG_SetPortKeepInStop2Mode(DISABLE);
```

LED7 を点灯し、while()ループにて SW0 の状態を取得します。
SW0 が OFF であれば何もせず、ON であれば LED7 を消灯し、STOP2 モードへ移行します。

```
    LED_On(LED7);
    while (1U) {
        if (SW_Get(SW0) == 0U) {
            LED_Off(LED7);          /* LED7 is off before enter stop2 */

            enterSTOP2();
```

```
        }
    }
```

STOP2 モードの解除要因として KSCAN 割り込みを許可します。

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

STOP2 モードに移行する準備を行います。
まず低消費電力モードとして STOP2 を選択し、次に PLL をオフしてから内部クロックに切り替えます。その後、ポートキープ機能を許可します。

```
    /*Set standby mode as Stop2 */
    CG_SetSTBYMode(CG_STBY_MODE_STOP2);

    TSB_CG->PLLSEL &= 0xfffffffe;
    do {
        tmp = (TSB_CG->PLLSEL & 0x00000001);
    } while (0x00000000 != tmp);

    /* Clock(external) Setup */
    TSB_CG->OSCCR &= OSCCR_WUODR_MASK;
    TSB_CG->OSCCR |= OSCCR_WUODR_EXT;

    if (TSB_CG_OSCCR_XEN2 == 0U) {
        TSB_CG->OSCCR |= 0x00010000;

        TSB_CG->OSCCR |= 0x00080000;

        TSB_CG->OSCCR |= 0x00000001;

        /*wait warm-up finish */
        do {
            tmp = (TSB_CG->OSCCR & 0x00000002);
        } while (0x00000000 != tmp);
    }

    TSB_CG->OSCCR &= 0xfffdffff;

    TSB_CG->STBYCR |= 0x00020000;
```

最後に__WFI()命令を使用して STOP2 モードに入ります。

```
    __WFI();
```

7-2-3 例: NORMAL<->IDLE モード変更

TX04 ペリフェラル・ドライバ(CG, GPIO, KSCAN, TMRB)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 IDLE モードの切り替え方法

- **サンプルプログラムのコードと説明**

スイッチ、LED、KSCAN 用端子の設定を行います。

```
    SW_Init();
    LED_Init();

    /* Set KSCAN's GPIO */
    configGPIO_KSCAN(ENABLE, DISABLE);
```

KSCAN の初期化を行い、KSCAN 動作を開始します。なお、この例ではタイマ出力クロックを KSCAN 動作クロックとして使用します。

```
    /* TB19OUT(fPLL:1/1) */
    configTMRB_TB19OUT(TMRB19TIME_fPLL_1_1);

    KSCAN_SetSCLK(KSCAN_KSCL_TBOUT);

    /* KSCAN initial */
    configKSCAN();

    /* KSCAN interrupt */
    CG_ClearINTReq(CG_INT_SRC_INTKSCAN);
    NVIC_ClearPendingIRQ(INTKSCAN_IRQn);

    NVIC_EnableIRQ(INTKSCAN_IRQn);

    KSCAN_SetINTReq(ENABLE);

    startKSCAN(ENABLE);
```

LED7 を点灯し、while()ループにて SW0 の状態を取得します。
SW0 が OFF であれば何もせず、ON であれば、LED7 を消灯し、IDLE モードへ移行します。

```
    LED_On(LED7);
    while (1U) {
        if (SW_Get(SW0) == 0U) {
            /* LED7 is off before enter IDLE*/
            LED_Off(LED7);

            enterIDLE();
        }
    }
```

IDLE モードの解除要因として KSCAN 割り込みを許可します。

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

DILE モードに移行する準備を行います。
まず低消費電力モードとして IDLE を選択します。

```
    /*Set standby mode as IDLE */
    CG_SetSTBYMode(CG_STBY_MODE_IDLE);
```

最後に__WFI()命令を使用して IDLE モードに入ります。

```
    __WFI_wait();
```

コール元の割り込みが禁止状態である場合があるため、NOP 命令を 8 個挿入します。

```
void __WFI_wait()
{
    __WFI();
```

```
    __NOP();
    __NOP();
    __NOP();
    __NOP();
    __NOP();
    __NOP();
    __NOP();
    __NOP();
}
```

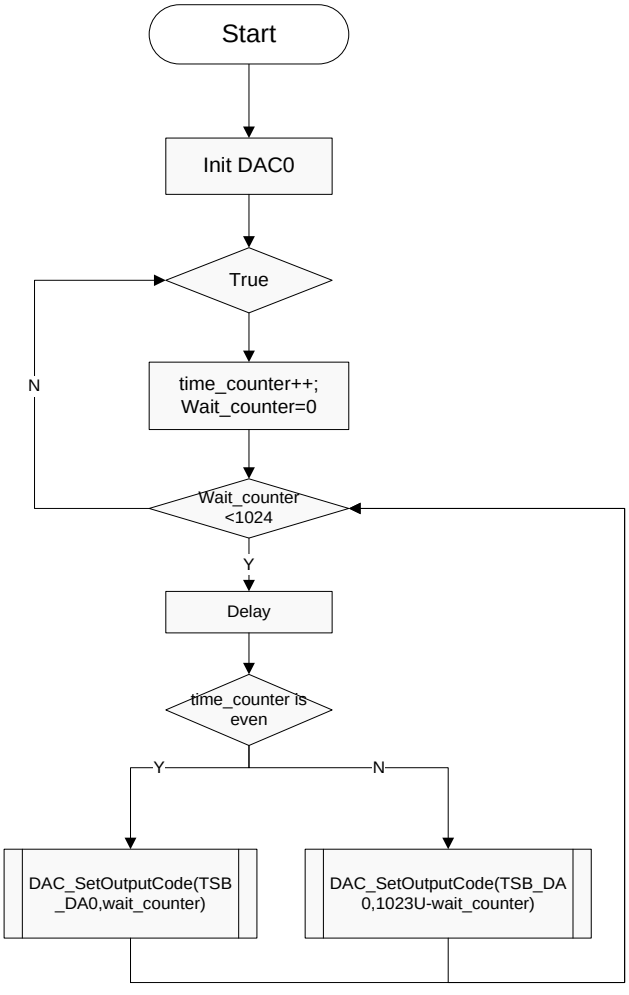
7-3 DAC

7-3-1 例: のこぎり波形出力

TX04 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_SetVOutHoldTime(TSB_DAA);
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

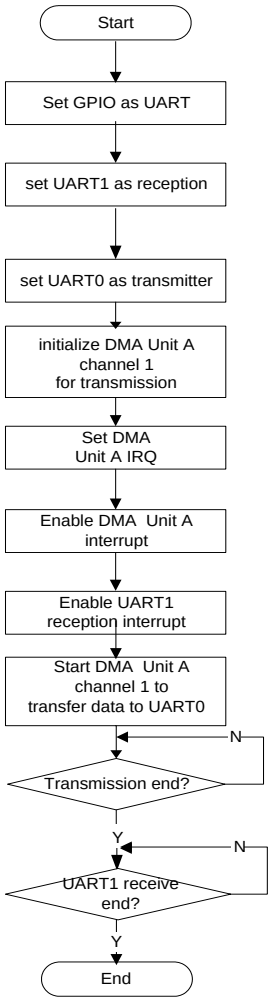
7-4-1 例: メモリから周辺回路

TX04 ペリフェラルドライバ(DMAC, GPIO, SIO) を使用したサンプルプログラムです。

以下の例が含まれます:

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

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

まず、データ転送のために UART0 の設定と許可を行います。

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

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

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

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

DMAC ユニット A の割り込みを許可します。

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

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

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

DMAC 転送が終わるまで待ちます。

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

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

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

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

7-5 EPHC

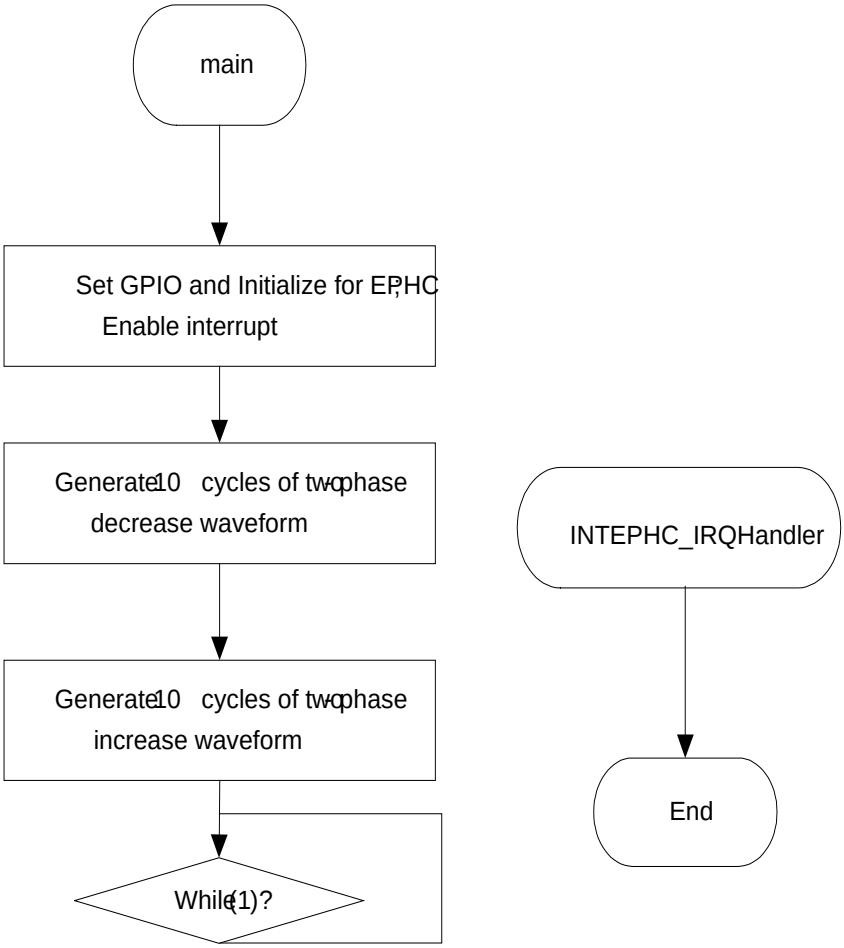
7-5-1 例: EPHC サンプル

TX04 ペリフェラル・ドライバ(EPHC, 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. EPHC アップ/ダウンカウンターは 4 逓倍モードで動作します。

- **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、GPIO の初期化を行います。

PR6 - EPHC0IN0 機能入力

PR7 - EPHC0IN1 機能入力

PR2 – GPIO 出力

PR3 – GPIO 出力

```
#define     GPIO_EPHC0              GPIO_PR
#define     EPHC0IN0_BIT            GPIO_BIT_6
#define     EPHC0IN1_BIT            GPIO_BIT_7
#define     GPIO_EPHCOUT            GPIO_PR
#define     EPHC0IN0_OUT_BIT        GPIO_BIT_2
#define     EPHC0IN1_OUT_BIT        GPIO_BIT_3
```

```
    GPIO_SetInput(GPIO_EPHC0, (EPHC0IN0_BIT | EPHC0IN1_BIT));
    GPIO_SetOutput(GPIO_EPHCOUT, (EPHC0IN0_OUT_BIT | EPHC0IN1_OUT_BIT));
    GPIO_EnableFuncReg(GPIO_EPHC0,  GPIO_FUNC_REG_3,  (EPHC0IN0_BIT  |
EPHC0IN1_BIT));
    GPIO_DisableFuncReg(GPIO_EPHCOUT,                    GPIO_FUNC_REG_2,
(EPHC0IN0_OUT_BIT | EPHC0IN0_OUT_BIT));
```

EPHC 初期化構造の準備を行います。

```
    EPHC_InitTypeDef InitStruct;

    InitStruct.CountDownEdgeSelForP1 = EPHC_CNT_MA1DN_NOCNTDOWN0;
    InitStruct.CountUpEdgeSelForP1 = EPHC_CNT_MA1UP_NOCNTUP0;
    InitStruct.InputClkSelection = EPHC_CNT_BRCK_FC;
    InitStruct.PhaseSelection = EPHC_CNT_PBDIR_POSITIVEPHASE;
    InitStruct.ModeSetting = EPHC_CNT_MA12_PULSE_2;
    InitStruct.DirectionSetting = EPHC_CNT_MA2DIR_NEGATIVEDIR;
    InitStruct.NoiseFilterCtrl = EPHC_CNT_NOISEFILTER_NONE;
    InitStruct.CountClearCtrl = EPHC_COUNT_CLR;
```

EPHC モジュールをイネーブルにし、初期化構造を指定のレジスタへ設定します。全割り込みをイネーブルにし、最後に EPHC の実行を開始します。

```
    EPHC_Enable(TSB_EPHC);
    EPHC_Init(TSB_EPHC, &InitStruct);
    EPHC_DisableInterrupt(TSB_EPHC, EPHC_IE_INT_ALL);
    EPHC_EnableInterrupt(TSB_EPHC,EPHC_IE_INT_ALL);
    NVIC_EnableIRQ(INTEPHC_IRQn);
    EPHC_ASetRunState(TSB_EPHC, EPHC_RUN);
```

その後、カウンターにアンダーフローあるいはオーバーフローが発生すると、メインルーチンは“While(1)”に入り、割り込みが発生します。

```
    void INTEPHC_IRQHandler(void)
    {
        /* if overflow or underflow interrupt occurred,clear EPHC0ADAT to 0. */
    }
```

7-6 ESIO

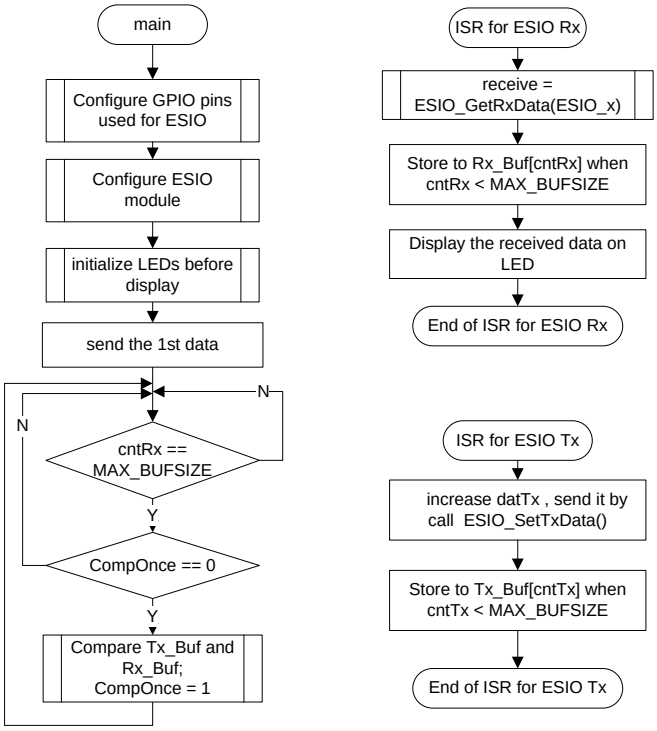
7-6-1 例: 基本転送

TX04 ペリフェラルドライバ(ESIO)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. ESIO 基本動作設定
2. Tx と Rx の ESIO 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、固有ポートの CR、FR1、IE レジスタを設定することにより、ESIO 用の GPIO 端子を設定し、ディスプレイ用の LED ポートを初期化します。

そして、ドライバー関数をコールすることにより、ESIO モジュールを設定します。

```
void ConfigESIO(void)
{
    /* Single transfer */
    uint8_t transfer_num = 1;
```

```
    /* maximum clock: clk = phiT0/4, divider=1 */
    ESIO_BaudClock clk = ESIO_PHIT0_DIVIDE_4;
    uint8_t divider = 1U;

    ESIO_InitTypeDef init = { 0U };

    ESIO_Enable(ESIO_x);
    /* ESIO_SWReset(ESIO_x); */

    ESIO_SetTxRxCtrl(ESIO_x, ENABLE);
    ESIO_SelectMode(ESIO_x, ESIO_MODE_SIO);
    ESIO_SelectTransferMode(ESIO_x, ESIO_TRMODE_TXRX);

    ESIO_SetTransferNum(ESIO_x, transfer_num);

    ESIO_SetFIFOLevelINT(ESIO_x, ESIO_TX , 0U);
    ESIO_SetFIFOLevelINT(ESIO_x, ESIO_RX , 0U);

    /* enable ESIO interrupt */
    ESIO_SetINT(ESIO_x, ESIO_INT_ALL, ENABLE);

#ifdef BITRATE_MIN
    clk = ESIO_PHIT0_DIVIDE_1024;
    divider = 16U;
#endif
    ESIO_SetBaudRate(ESIO_x, clk, divider );

    init.DataDirection = ESIO_LSB_FIRST;
    init.FrameLength = sizeof(datTx) * 8U;          /* size of datTx in main() */
    init.CycleBetweenFrames = 1U;
    init.NumberOfLINE = 1U;
    init.ClkPolority = ESIO_CLKPOL_LOW;
    init.ShortestIdleCycle = 1U;
    init.CS0ActiveLevel = ESIO_CS_ACTIVE_LOW;
    init.DelayCycleNum_CS_SCLK = 1U;
    init.DelayCycleNum_Negate_CS = 1U;
    init.HorizontalParityCheck = DISABLE;
    /* init.HorizontalParity = ESIO_PARITY_ODD; */
    init.VerticalParityCheck = DISABLE;
    ESIO_Init(ESIO_x, &init);

    NVIC_ClearPendingIRQ(INTE0RX_IRQn );
    NVIC_EnableIRQ(INTE0RX_IRQn);

    NVIC_ClearPendingIRQ(INTE0TX_IRQn);
    NVIC_EnableIRQ(INTE0TX_IRQn);

    NVIC_ClearPendingIRQ(INTE0ERR_IRQn);
    NVIC_EnableIRQ(INTE0ERR_IRQn);

    __enable_irq();
}
```

そして、TX 割り込みを発生させるため第 1 番目のデータを送信します。

```
    Tx_Buf[0] = datTx;
    ESIO_SetTxData(ESIO_x, datTx);
```

TX の割り込みサービスルーチンにて、残りのデータ送信を継続します。

```
void INTE0TX_IRQHandler(void)
{
      datTx++;
      ESIO_SetTxData(ESIO_x, datTx);
      if (cntTx < MAX_BUFSIZE) {
            Tx_Buf[cntTx] = datTx;
            cntTx++;
      }
}
```

データ受信のために Rx の割り込みサービスルーチンを使用します。

```
void INTE0RX_IRQHandler(void)
{
      uint8_t tmp = 0U;

      receive = ESIO_GetRxData(ESIO_x);
      if (cntRx < MAX_BUFSIZE) {
          Rx_Buf[cntRx] = (uint8_t)receive;
          cntRx++;
     }
     /* adjust the loop blinkging speed */
     tmp = (uint8_t) (receive >> 4U);
     LED_Display(tmp);
}
```

7-7 EXB

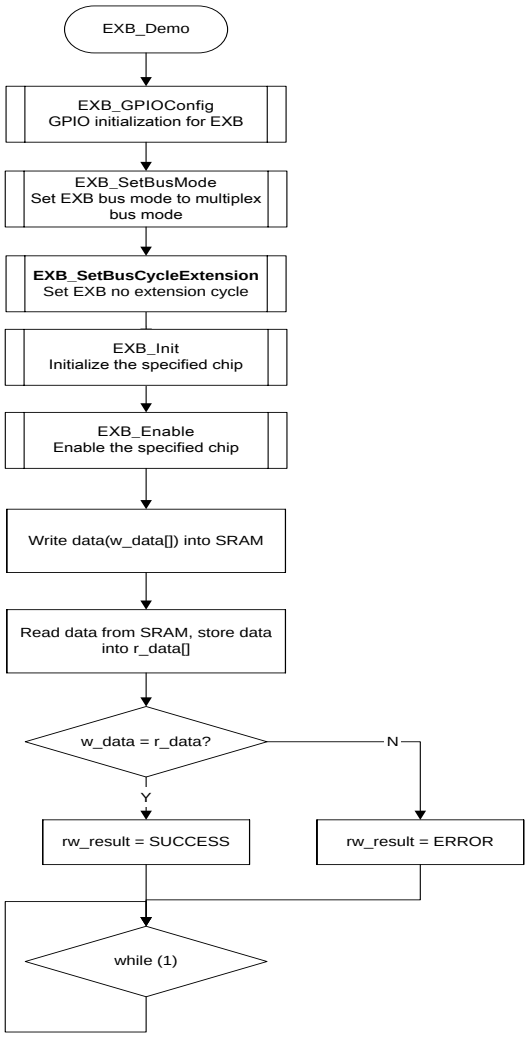
7-7-1 例: SRAM のリード/ライト

EXB と GPIO を使って SRAM のリード/ライトを行う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。

```
	uint32_t w_data[TEST_DATA_LEN] = { 0U };
	uint32_t r_data[TEST_DATA_LEN] = { 0U };
	uint16_t rw_cnt = 0U;
	uint8_t chip = EXB_CS0;
	uint8_t BusMode = EXB_BUS_SEPARATE;
	uint8_t Cycle = EXB_CYCLE_NONE;

#ifdef SRAM_RW
	uint32_t addr = NULL;
	uint16_t i = 0U;
#endif

	EXB_InitTypeDef InitStruct = { 0U };

	InitStruct.AddrSpaceSize = EXB_1M_BYTE;
	InitStruct.StartAddr = EXB_SRAM_START_ADDR;
	InitStruct.BusWidth = EXB_BUS_WIDTH_BIT_16;
	/* Set cycles time according to AC timing of SRAM datasheet,base clock: EXBCLK(fsys)
*/
	InitStruct.Cycles.InternalWait = EXB_INTERNAL_WAIT_4;
	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 successful 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 */
	}
```

7-8 FLASH

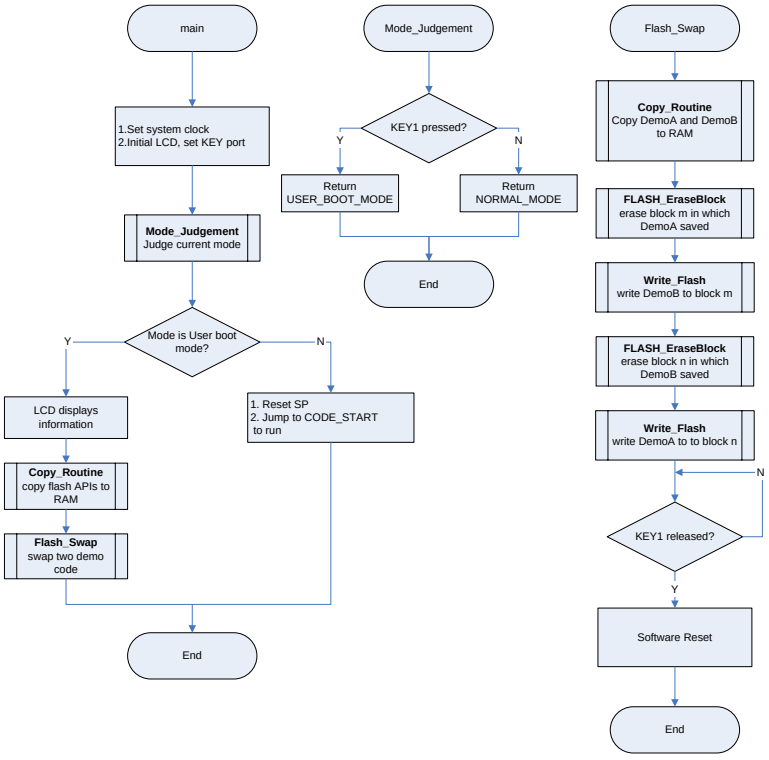
7-8-1 例: Flash スワップ

TX04 ペリフェラル・ドライバ(FLASH, 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. FLASH メモリのオンボードプログラミング (書き込み/消去)
2. 動作モード:シングルチップモード (ノーマルモード、ユーザーブートモード)
3. ユーザーブートモードを使用し、Flash メモリのコードを更新。

- **フローチャート**



- **サンプルプログラムのコードと説明**

まず LED と SW0 を初期化します。リセット時に現在のモード判定を SW0 で行います。

```
    LED_Init();
```

```
    SW_Init();
```

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

リセット時に SW0 が ON でない場合、ルーチンプログラムはノーマルモードになります。その後、ルーチンプログラムは SP をリセットし、“CODE_START” にジャンプします。サンプル A はブロック m に含まれるアドレス“CODE_START”を保存し、サンプル A が動作します (LED0:点滅)。

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

リセット時に SW0 が ON の場合、ルーチンプログラムはユーザ Boot モードになります。その後、ルーチンプログラムは、フラッシュ操作を行う API を Flash ROM 内のアドレス
“FLASH_API_ROM” から、RAM 内のアドレス“FLASH_API_RAM”へコピーします。
(Flash ROM 内で実行しながら Flash ROM の書き換えや消去が行えないため)

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

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

Flash_Swap() 関数では、まずサンプル A とサンプル B のプログラムを Flash ROM から RAM にコピーします。次に、Flash ROM の消去/書き込みを行うため、関数 FC_EraseBlock()と Write_Flash()をコールします。サンプル A とサンプル B のプログラムは Flash ROM 内にて差し替えられます。

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

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

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

```
    } else {
        return ERROR;
    }

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

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

SW0 を OFF すると、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、サンプルプログラムが再度動作しノーマルモードになります。サンプル A とサンプル B のプログラムが差し替えられるため、アドレス“CODE_START”はサンプル B のスタートアドレスになります。その後、サンプル B のプログラムを実行します(LED1 が点滅)。

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

    NVIC_SystemReset();  /* software reset */
```

Flash メモリ動作関数 FC_EraseBlock() は自動的に指定されたブロックを消去します。このブロックは最初に引数 “block_addr” で指定します。まず、この関数で引数“block_addr” を確認します。次に、Flash ドライバ関数の 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;
    }
```

7-9 FUART

7-9-1 例: ループバック

TX04 ペリフェラルドライバ(FUART, GPIO)を使用したサンプルプログラムです。

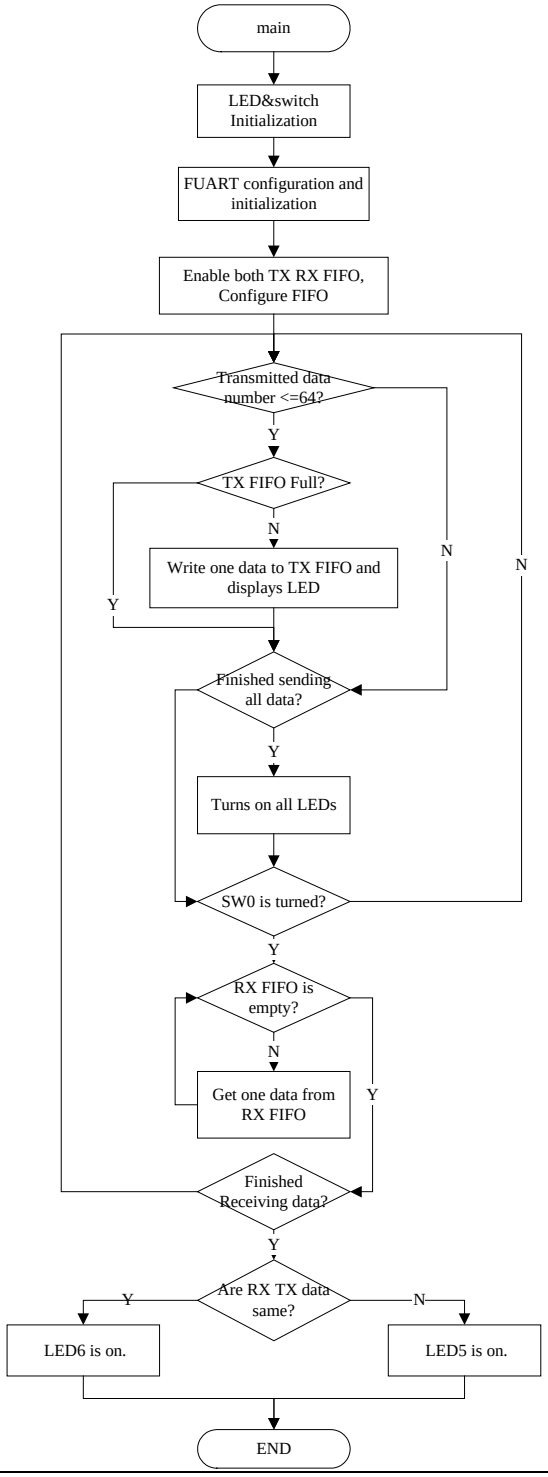
本プログラムはハードウェアフロー制御機能を確認するために 2 回実行します。

1 回目:RTS と CTS ハードウェアフロー制御をイネーブルにします。
2 回目:RTS と CTS ハードウェアフロー制御をディセーブルにします。

以下の例が含まれます:

1. LED、スイッチの初期化、Full UART の設定と初期化
2. Full UART 送信データ処理
3. データ受信のため、SW0 を ON にする
4. データ受信終了後、受信データを送信データと比較

- **フローチャート**


main
LED&switch Initialization
FUART configuration and initialization
Enable both TX RX FIFO, Configure FIFO
Transmitted data number <=64?
Y
TX FIFO Full?
N
Write one data to TX FIFO and displays LED
Y
N
N
Finished sending all data?
Y
Turns on all LEDs
SW0 is turned?
Y
RX FIFO is empty?
N
Get one data from RX FIFO
Y
Finished Receiving data?
Y
Are RX TX data same?
Y
N
LED6 is on.
LED5 is on.
END

- **サンプルプログラムのコードと説明**

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

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

まず、LED とスイッチを初期化します。

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

FUART0 用 GPIO を設定します。

```
	/* Configure port PR0 to be TXD06 */
	TSB_FUART->FR1 |= GPIO_BIT_0;
	TSB_FUART->IE &= ~GPIO_BIT_0;
	TSB_FUART->CR |= GPIO_BIT_0;

	/* Configure port PR1 to be RXD06 */
	TSB_FUART->FR1 |= GPIO_BIT_1;
	TSB_FUART->IE |= GPIO_BIT_1;
	TSB_FUART->CR &= ~GPIO_BIT_1;

	/* Configure port PR2 to be CTS06 */
	TSB_FUART->FR1 |= GPIO_BIT_2;
	TSB_FUART->IE |= GPIO_BIT_2;
	TSB_FUART->CR &= ~GPIO_BIT_2;

	/* Configure port PR3 to be RTS06 */
	TSB_FUART->FR1 |= GPIO_BIT_3;
	TSB_FUART->IE &= ~GPIO_BIT_3;
	TSB_FUART->CR |= GPIO_BIT_3;
```

FUART_InitTypeDef 構成を生成し、全データフィールドを入力します。その後 FUART0 を初期化します。

```
	FUART_InitTypeDef myFUART;

	myFUART.BaudRate = 300U;
	myFUART.DataBits = FUART_DATA_BITS_8;
	myFUART.StopBits = FUART_STOP_BITS_1;
	myFUART.Parity = FUART_1_PARITY;
	myFUART.Mode = FUART_ENABLE_TX | FUART_ENABLE_RX;

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

	FUART_Init(FUART0, &myFUART);
```

FUART0 をイネーブルにし FIFO を設定します。

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

その後 FUART0 はデータ送信を開始します。Full UART は 64 個のデータ値を送信したのち、送信 FIFO が正常あるいは空の場合、データを送信します。各データが送信されると、 LED はデータを表示します。全データが送信されると、全 LED が点灯します。

```
        if (cntTx < MAX_BUFSIZE) {
            FIFOStatus = FUART_GetStorageStatus(FUART0, FUART_TX);
            if ((FIFOStatus == FUART_STORAGE_EMPTY)
                || (FIFOStatus == FUART_STORAGE_NORMAL)) {
                FUART_SetTxData(FUART0, Tx_Buf[cntTx]);
                LED_TXDataDisplay(Tx_Buf[cntTx]);
                cntTx++;
                if (62U == cntTx) {
                    LED_On(LED_ALL);       /* sending data is finished */
                }
            }
        }
```

SW0 が ON するごとに、プログラムは受信 FIFO からのデータ読み出しを開始します。 データが存在している場合、プログラムは FIFO が空になるまでデータの読み出しを継続します。SW0 が ON であり、データが存在しない場合には、受信データと送信データの比較を開始します。受信データが送信データと同一の場合、LED6 が点灯します。受信データと送信データと異なる場合、LED5 が点灯します。

```
    SW0_last = SW0_this;
        SW0_this = SW_Get(SW0);
        if (SW0_last != SW0_this) {        /* turn the switch SW0 */
            while (FUART_STORAGE_EMPTY
                   != FUART_GetStorageStatus(FUART0, FUART_RX)) {
                receive = FUART_GetRxData(FUART0);
                Rx_Buf[cntRx] = receive;
                cntRx++;
            }
            rxlast = rxthis;
            rxthis = cntRx;

            if (rxlast != rxthis) {        /* there are some data that has been received    */
                LED_Off(LED_ALL);
                LED_On(LED7);
            } else {                    /* receiving data is finished */
                result = Buffercompare(Tx_Buf, Rx_Buf, MAX_BUFSIZE);
                if (result == SAME) {
                    LED_Off(LED_ALL);
                    LED_On(LED6);
                    while (1) {
                    }
                } else {
                    /* received data are different with trnsmitted data */
                    /* RTS and CTS flow control doesn't work */
                    LED_Off(LED_ALL);
                    LED_On(LED5);
                    while (1) {
                    }
                }
            }
        }
```

7-10 GPIO

7-10-1 例: GPIO による LED 制御

TX04 ペリフェラルドライバ(GPIO)を使用したサンプルプログラムです。

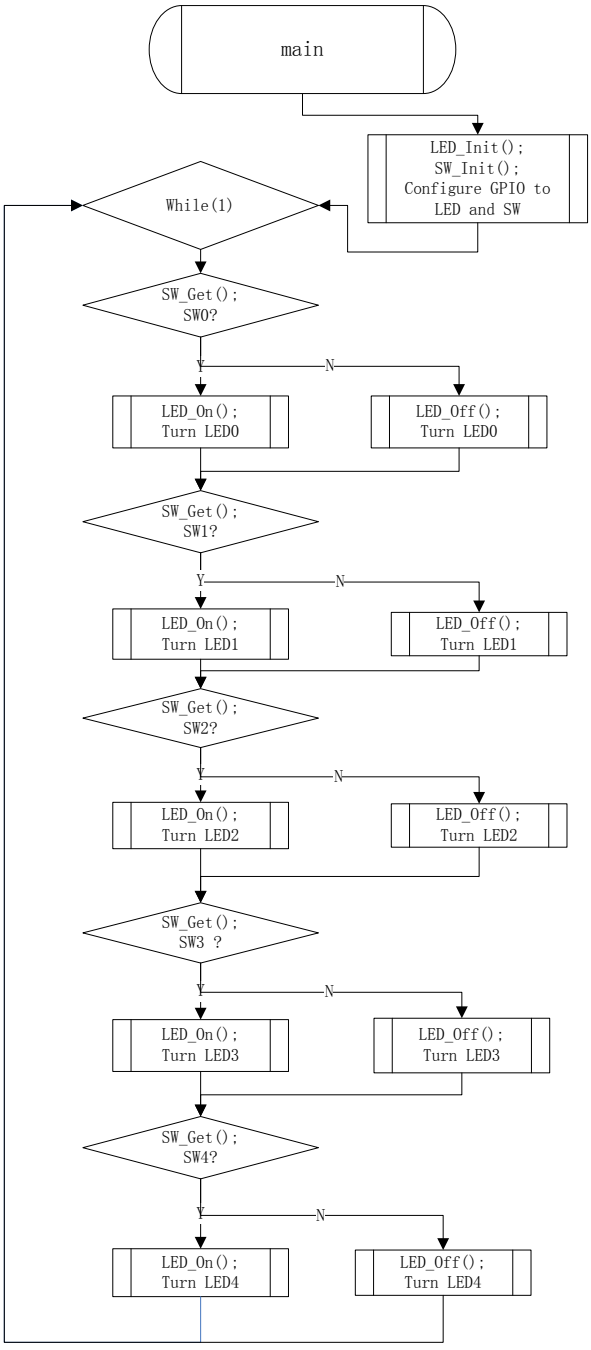
この例では以下を行います。

1. GPIO の初期化
2. GPIO へのデータ出力
3. GPIO からのデータ入力

SW, LED, LCD の接続:

端子	機能
PAD0	SW0
PAD1	SW1
PAD2	SW2
PAD3	SW3
PAD4	SW4
PT0	LED0
PT1	LED1
PT2	LED2
PT3	LED3
PT4	LED4

- **フローチャート:**


main
LED_Init();
SW_Init();
Configure GPIO to
LED and SW
While(1)
SW_Get();
SW0?
Y
N
LED_On();
Turn LED0
LED_Off();
Turn LED0
SW_Get();
SW1?
Y
N
LED_On();
Turn LED1
LED_Off();
Turn LED1
SW_Get();
SW2?
Y
N
LED_On();
Turn LED2
LED_Off();
Turn LED2
SW_Get();
SW3 ?
Y
N
LED_On();
Turn LED3
LED_Off();
Turn LED3
SW_Get();
SW4?
Y
N
LED_On();
Turn LED4
LED_Off();
Turn LED4

- **サンプルプログラムのコードと説明:**

まず、GPIO_SetOutput(GPIO_Port GPIO_x, uint8_t Bit_x)を使用し LED に対する GPIO を設定します。次に GPIO_SetInput(GPIO_Port GPIO_x, uint8_t Bit_x)を使用し、スイッチに対する GPIO を設定します。

```
/* LED0~LED4 */
uint8_t tmp = 0U;
GPIO_SetOutput(LED_DATA_PORT, LED_ALL);
tmp = GPIO_ReadData(LED_DATA_PORT);
tmp &= (~LED_ALL);
GPIO_WriteData(LED_DATA_PORT,tmp);
/* SW0~4 */
GPIO_SetInput(SW_PORT, GPIO_BIT_ALL);
```

スイッチ制御による LED の ON/OFF を行います。

GPIO_WriteDataBit(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t BitValue)を使用し、LED を点灯します。

```
uint32_t tmp;
tmp = GPIO_ReadData(LED_DATA_PORT);
tmp |= led;
GPIO_WriteData(LED_DATA_PORT, tmp);
```

GPIO_WriteDataBit(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t BitValue)を使用し、LED を消灯します。

```
uint32_t tmp;
tmp = GPIO_ReadData(LED_DATA_PORT);
tmp &= ~led;
GPIO_WriteData(LED_DATA_PORT, tmp);
```

7-11 KSCAN

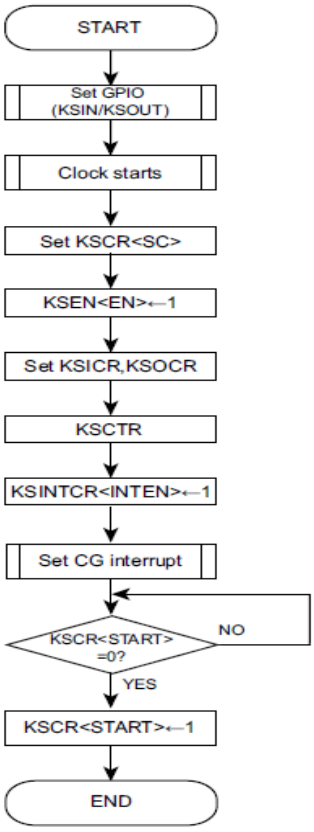
7-11-1 例: KSCAN サンプル

KSCAN、GPIO、CG のドライバを使用したサンプルプログラムです。

本例には以下を含みます。

1. GPIO 初期化
2. KSCAN 設定
3. KSCAN 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 LED を初期化します。

```
LED_Init();
```

次に fs を許可します。

```
CG_SetFs(ENABLE);
CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT,WARMTIME);
do {
    st = CG_GetWarmUpState();
} while (st != DONE);
```

その後、KSCAN に対する GPIO の設定を行います。

```
KSCAN_GPIOConfig();
```

ksclk と fs を選択します。

```
KSCAN_SetSCLK(KSCAN_KSCL_FS);
```

KSCAN 回路を有効にします。

```
KSCAN_Enable();
```

KSCAN 入力と KSCAN 出力の設定を行います。

```
KSCAN_SetInputCtrlMask(KSCAN_CH_0,DISABLE); /*All channels are not masked*/

KSCAN_SetOutputCtrl((~KSCAN_CH_0),ENABLE);
```

ストローブ信号の設定を行います。

```
KSCAN_SetStrobeOutput(2U);
KSCAN_SetStrobeWidth(3U);
```

チャタリング除去時間の設定を行います。

```
KSCAN_SetChatCancelTime(60U);    /*Sets a chattering cancel time*/
```

同一ラインに対する KSCAN 入力値の比較回数を選択します。

```
KSCAN_SetConsecutiveMatches(KSCAN_MATCH_2C);
```

KSCAN 割り込みを許可します。

```
KSCAN_SetINTReq(ENABLE);
```

CG の割り込み制御回路の設定を行います。

```
CG_SetSTBYReleaseINTSrc(CG_INT_SRC_INTKSCAN,CG_INT_ACTIVE_STATE_RISING,ENABLE);
NVIC_EnableIRQ(INTKSCAN_IRQn);
```

KSCAN 動作を開始します。

```
KSCAN_Start();
```

7-12 KWUP

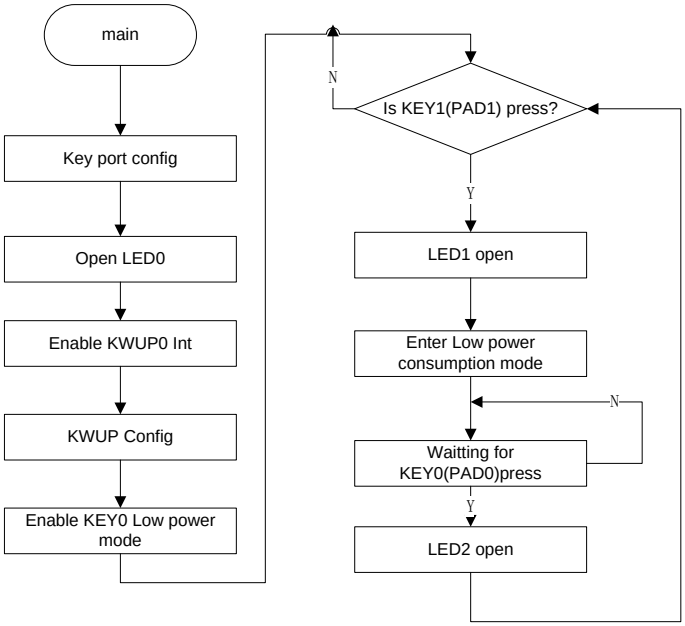
7-12-1 例: 低消費電力モードの解除

KWUP と GPIO のドライバを使用したサンプルプログラムです。

以下の例を含みます:

1. KWUP 設定
2. 低電力モードへの移行と解除

- **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 KWUP 用の初期値を設定し、KWUP 用の GPIO を設定します。

```
KWUP_SettingTypeDef keySetting;
KWUP_PortStatus portStatus;
TSB_KWUP_TypeDef * TSB_KWUPx;

 CG_SetSTBYMode(CG_STBY_MODE_STOP1);
 /* set PAD0,PAD1 to Key0,1 */
 TSB_PAD_FR1_PAD0F1 = 1U;
 TSB_PAD_PUP_PAD0UP = 1U;
 TSB_PAD_IE_PAD0IE = 1U;

 TSB_PAD_FR1_PAD1F1 = 1U;
```

```
        TSB_PAD_PUP_PAD1UP = 1U;
        TSB_PAD_IE_PAD1IE = 1U;

        LED_Init();
        LED_On(LED0);

        /* enable KWUP interrupt */
        NVIC_ClearPendingIRQ(INTKWUPA_IRQn);
        NVIC_EnableIRQ(INTKWUPA_IRQn);

        keySetting.KeyN = KWUP_INPUT_0;
        keySetting.PullUpCtrl = KWUP_PUP_CTRL_BY_STATIC;
        keySetting.ActiveState = KWUP_ACTIVE_BY_RISING_EDGE;
        keySetting.INTNewState = ENABLE;
       TSB_KWUPx=TSB_KWUPA;
        KWUP_SetConfig(TSB_KWUPx,&keySetting);

        KWUP_SetPullUpConfig(TSB_KWUPx,KWUP_CYCLES_4_FS,
                                        KWUP_CYCLES_256_FS);
```

KEY1 (PAD1)を押すと、CPU は低電力モードに移行し、LED は消灯します。その後 KEY0(PAD0)を押すと、低電力モードを解除します。

```
    /* enable low power mode release interrupt to INTKWUP0 */
    TSB_CG->IMCGF = 0x100000U;
    TSB_CG->IMCGF += 0x010000U;

    do {
        portStatus = KWUP_GetPortStatus(TSB_KWUPx);

        if (portStatus.Bit.Key1 == 0U) {            /* press key(PAD1) */
            LED_On (LED1);
            /* enter low power mode */
            __WFI_wait();

            /* press KEY(PAD0) to release low power mode */

            LED_On(LED2);
        }
    }
    while (1);
```

コール元の割り込みが禁止状態である場合があるため、NOP 命令を 8 個挿入します。

```
    void __WFI_wait()
    {
        __WFI();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
    }
```

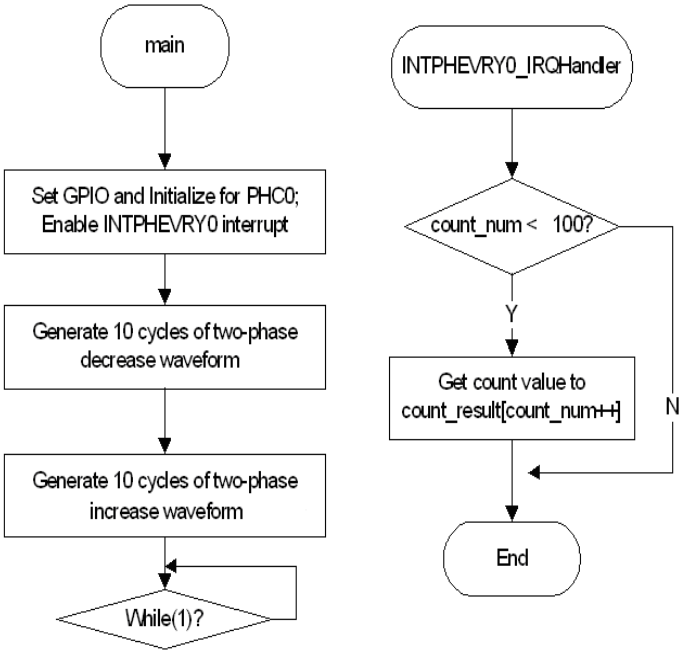
7-13 PHC

7-13-1 例: PHC サンプル

PHC と GPIO のドライバを使用したサンプルプログラムです。

以下の例を含みます。
・ 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。
PU4 - PHC0IN0 機能、入力
PU5 - PHC0IN1 機能、入力
PU2 - GPIO 出力
PU3 - GPIO 出力

```
#define     GPIO_PHC0           GPIO_PU
#define     PHC0IN0_BIT         GPIO_BIT_4
#define     PHC0IN1_BIT         GPIO_BIT_5
#define     GPIO_PHCOUT         GPIO_PU
#define     PHC0IN0_OUT_BIT     GPIO_BIT_2
```

```
#define        PHC0IN1_OUT_BIT        GPIO_BIT_3

    GPIO_SetInput(GPIO_PHC0, (PHC0IN0_BIT | PHC0IN1_BIT));
    GPIO_SetOutput(GPIO_PHCOUT, (PHC0IN0_OUT_BIT | PHC0IN1_OUT_BIT));
    GPIO_EnableFuncReg(GPIO_PHC0,GPIO_FUNC_REG_1,(PHC0IN0_BIT|
PHC0IN1_BIT));
    GPIO_DisableFuncReg(GPIO_PHCOUT,GPIO_FUNC_REG_1,  (PHC0IN0_OUT_BIT |
PHC0IN0_OUT_BIT));
```

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。

```
    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ー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 */
        }
    }
```

7-14 RTC

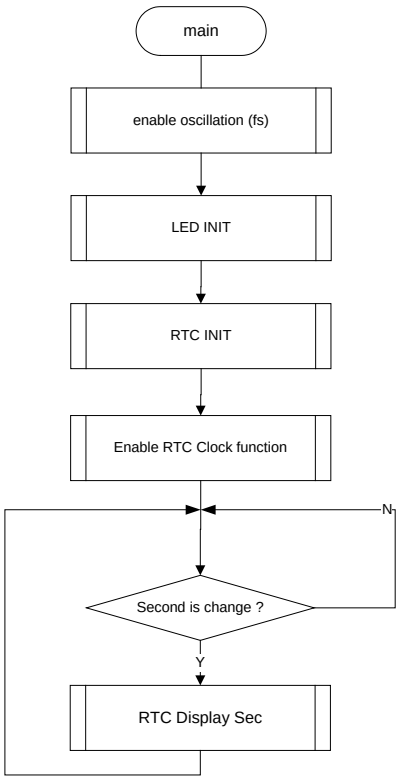
7-14-1 例: RTC サンプル

RTC ドライバ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. RTC の初期化
2. RTC 秒の取得

- **フローチャート:**



- **サンプルプログラムのコードと説明:**

RTC の初期化で、RTC_DateTypeDef と RTC_TimeTypeDef 構造を作成し、全データ項目を設定します。ここでは 2010/10/22 23:59:00, 24 時間表示フォーマットを初期設定としています。

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

    Time_Struct.HourMode = RTC_24_HOUR_MODE;
    Time_Struct.Hour = (uint8_t) 23U;
    Time_Struct.Min = (uint8_t) 59U;
    Time_Struct.Sec = (uint8_t) 0U;
```

クロックとアラーム機能を禁止します。

```
    RTC_DisableClock();
```

RTC 秒カウンタのリセット、RTCINT の許可を行います。

```
    RTC_ResetClockSec();
    RTC_SetRTCINT(DISABLE);
```

RTC の時間と日付の値を設定します。

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

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

```
    /* Enable RTC Clock function */
    RTC_EnableClock();
```

第 2 番目の値がバイナリで LED ディスプレイに送信されます。

下記に、第 2 番目の値の取得方法と表示方法を表します。

```
    Sec = RTC_GetSec();
    /* Display */
    TSB_LED->DATA = Sec
```

7-15 SBI

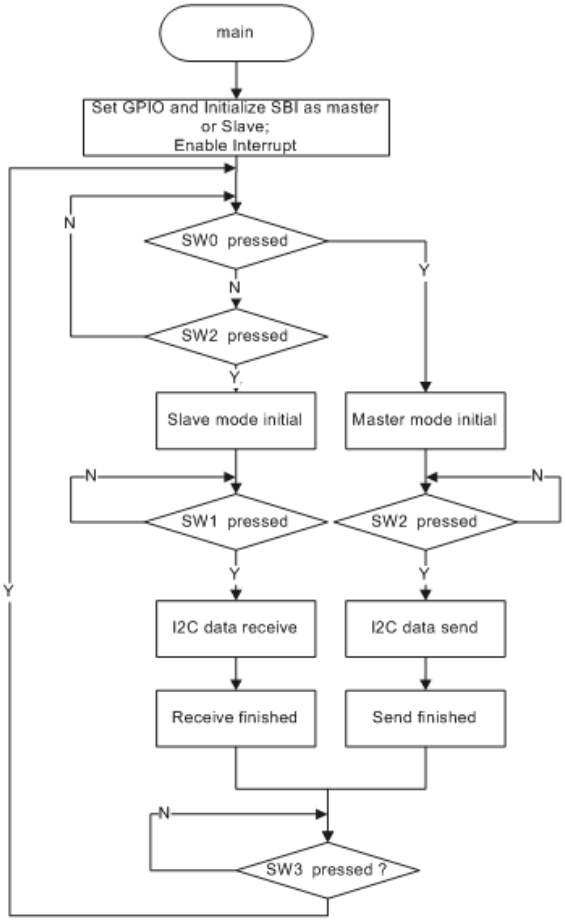
7-15-1 例: SBI スレーブ

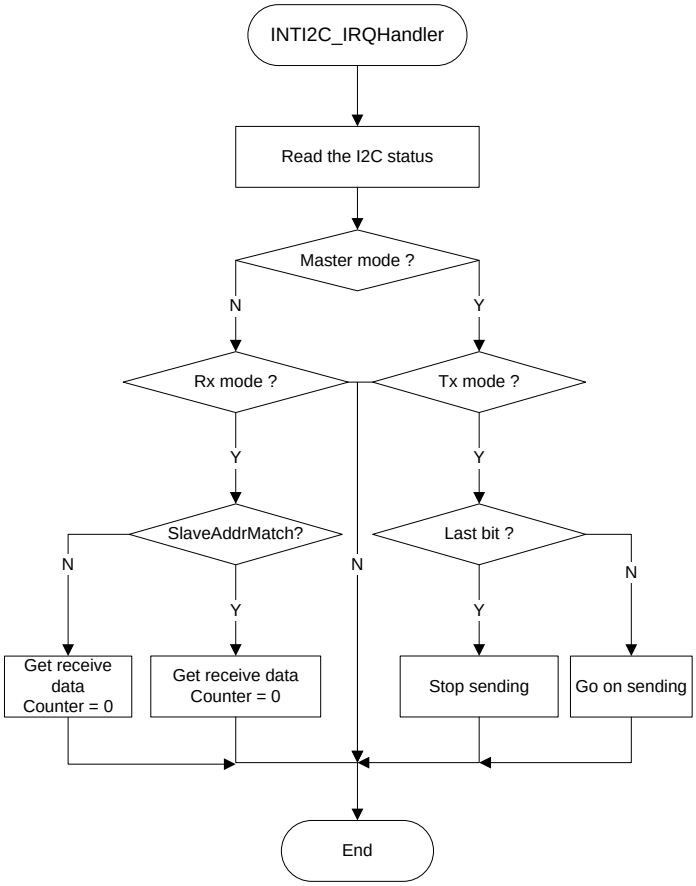
TX04 ペリフェラルドライバ(SBI, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

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、SBI2 用 GPIO を I2C モードに設定します。

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

次に、 SBI マスタ/スレーブモードをイネーブルし、初期化します。その後 INTI2C をイネーブルにします。

```
/* Master mode */
myI2Cm.I2CSelfAddr = SELF_ADDR;
myI2Cm.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2Cm.I2CACKState = ENABLE;
myI2Cm.I2CClkDiv = SBI_I2C_CLK_DIV_328;
/* Slave mode */
myI2Cs.I2CSelfAddr = SLAVE_ADDR;
myI2Cs.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2Cs.I2CACKState = ENABLE;
myI2Cs.I2CClkDiv = SBI_I2C_CLK_DIV_328;
gSBIMode = MODE_SBI_I2C_IDLE;
NVIC_EnableIRQ(INTI2C_IRQn);
```

上記設定を行った後、I2C 送信開始します。
マスタモードを初期化し、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。

```
case MODE_SBI_MASTER_INITIAL:
        /* Initialize SBI channel to Master mode */
        gI2CTxDataLen = 7U;
        gI2CTxData[0] = gI2CTxDataLen;
        gI2CTxData[1] = 'T';
        gI2CTxData[2] = 'O';
        gI2CTxData[3] = 'S';
        gI2CTxData[4] = 'H';
        gI2CTxData[5] = 'I';
        gI2CTxData[6] = 'B';
        gI2CTxData[7] = 'A';
        SBI_Enable(TSB_SBI0);
        SBI_SWReset(TSB_SBI0);
        SBI_InitI2C(TSB_SBI0, &myI2Cm);
        gSBIMode = MODE_SBI_I2C_START;
        break;
```

スレーブモードを初期化し、RX バッファをクリアします。

```
case MODE_SBI_ SLAVE_INITIAL:
    /* Initialize SBI channel to Slave mode */
    gI2CWCnt = 0U;
    for (gICnt = 0U; gICnt < 8U; gICnt++) {
        gI2CRxData[gICnt] = 0U;
    }
    SBI_Enable(TSB_SBI0);
```

```
            SBI_SWReset(TSB_SBI0);
            SBI_InitI2C(TSB_SBI0, &myI2Cs);
            gSBIMode = MODE_SBI_I2C_ RX;
            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_SBI2) 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_TX;
             LED_On(LED0);
      } else {
            /* Do nothing */
      }
      break;
```

RX の終了を確認するために gI2CRCnt を読み取ります。その後
MODE_SBI_SLAVE_FINISHED へ進みます。

```
case MODE_SBI_I2C_RX:
      LED_On(LED4);
      if (gI2CRCnt > gI2CRxData[0]) {
            gI2CRxDataLen = gI2CRxData[0];
            gI2CRCnt = 0U;
            gSBIMode = MODE_SBI_SLAVE_FINISHED;
       } else {
          /* Do nothing */
       }
       break;
```

TX の終了を確認するために gI2CWCn を読み取ります。その後
MODE_SBI_MASTER_FINISHED へ進みます。

```
case MODE_SBI_I2C_TX:
     LED_On(LED1);
     if (gI2CWCnt > gI2CTxData[0]) {
             gSBIMode = MODE_SBI_MASTER_FINISHED;
     } else {
             /* Do nothing */
     }
     break;
```

データ転送あるいは受信は INTI2C にて処理されます。
INTSBI2 ハンドラ内で、I2C バス状態を取得し、その値で I2C マスタ送信プロセスを決定します。
I2C マスタ送信中は、SBI_SetSendData() で次のデータを送信し、I2C プロセス終了時には、SBI_GenerateI2CStop() で I2C を停止します。

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

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

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

7-16 TMRB

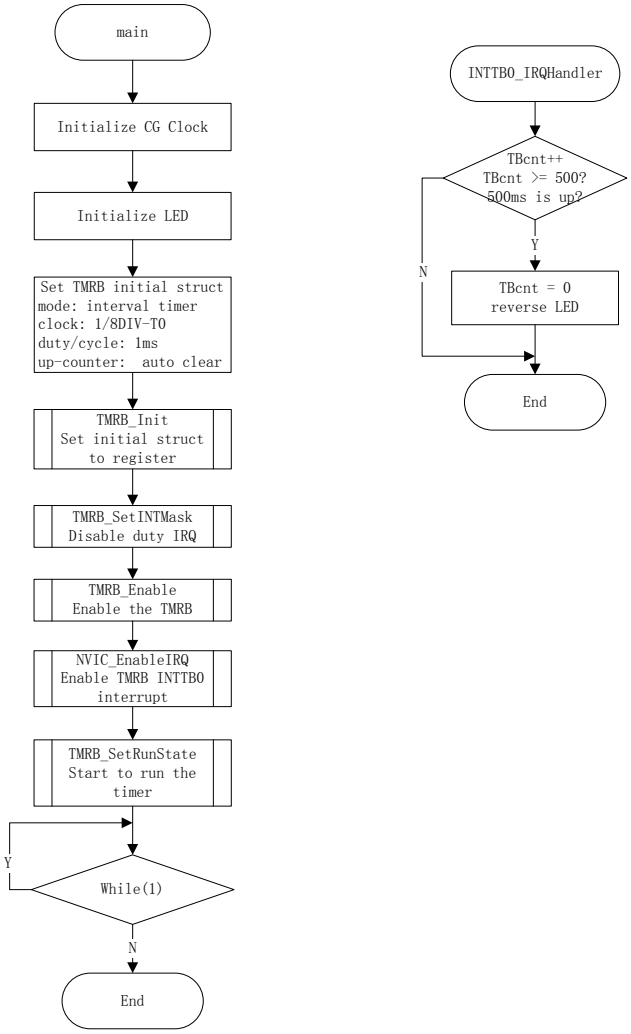
7-16-1 例: 汎用タイマ

TX04 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、LED0 から LED3 を OFF します。

```
LED_Init();                          /* LED initialize */
Led_off(LED0 | LED1 | LED2 | LED3);
```

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。このサンプルでは、1ms の周期とデューティを設定します。TMRB_1MS マクロは、0x30D4 です。(φT0=fsys=10MHz * PLL* = 100MHz, ftmrb = 1/8φT0 = 12MHz, Ttmrb = 0.08us, 1ms/0.08us = 12500 = 0x30D4 (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.Cycle = TMRB_1MS;                /* periodic time is 1ms */
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;     /* up-counter auto clear */
m_tmrb.Duty = TMRB_1MS;                 /* periodic time is 1ms */
```

TMRB モジュールを有効にした後、初期化構造体を指定のレジスタに設定します。INTTB00 割り込み（1ms ごとにトリガ）を有効にします。最後に TMRB を動作させます。

```
TMRB_SetINTMask(TSB_TB0,TMRB_MASK_MATCH_DUTY_INT);/*duty is not used in TMRB_INTERVAL_TIMER mode */
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) {
        Led_Off(LED0);
    } else {
        Led_On(LED0);
    }
} else {
    /* Do nothing */
}
```

7-16-2 例: PPG 出力

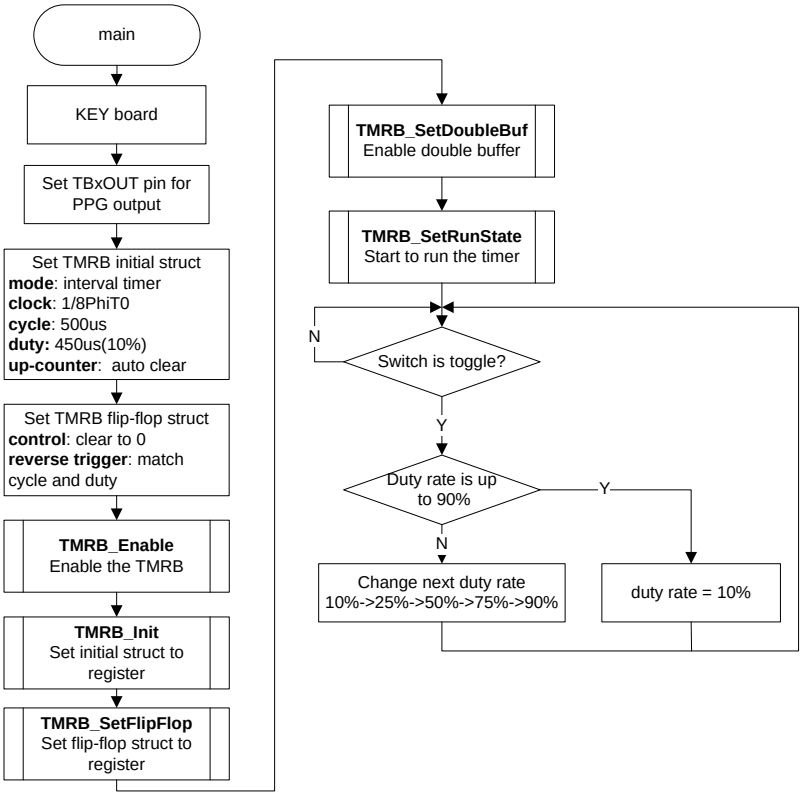
TMRB と GPIO のドライバを使用したサンプルプログラムです。

この例では以下を行います。

1. TMRB12 の初期化

2. PPG 機能の設定と開始
3. PPG デューティの調整

- **フローチャート**



- **サンプルプログラムのコードと説明**

最初に PPG 出力用に PW4 を TB12OUT に設定します。

```
    TSB_PW->CR |= 0x10;
    TSB_PW->FR1 |= 0X10;
    TSB_PW->IE &= 0;                    /* set KEY port to input */
```

TMRB の初期化用構造体を準備し、TMRB モード、クロック、アップカウンタクリア設定、サイクル、デューティを設定します。この例では 500us のサイクルを設定するため、TMRB4TIME マクロを定義してあります。このマクロは 0x186A です。(φT0 = fsys = fc = 10MHz * PLL * = 100MHz, ftmrb = 1/8 φT0 = 12MHz, Ttmrb = 0.08us, 500us/0.08us = 6250 = 0x186A (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.Cycle = TMRB4TIME;                 /* cycle is 500us */
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
m_tmrb.Duty = Duty[Rate];                 /* duty, 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、デューティー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_CYCLE |
                               TMRB_FLIPFLOP_MATCH_DUTY;
```

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_TB12);
TMRB_Init(TSB_TB12, &m_tmrb);
TMRB_SetFlipFlop(TSB_TB12, &PPGFFInital);
TMRB_SetDoubleBuf(TSB_TB12, ENABLE);     /* enable double buffer */
TMRB_SetRunState(TSB_TB12, TMRB_RUN);
```

スイッチ状態の変化を待ちます。

```
keyvalue = SW_Get(SW1);
if (keyvalue == 0) {
    delay(0xFFFU);          /* noise cancel */
    keyvalue = SW_Get(SW1);
    if (keyvalue == 1) {
        changestatus = 1;
    } else {
        /* Do nothing */
    }
} else if (keyvalue == 1) {
    delay(0xFFFU);          /* noise cancel */
    keyvalue = SW_Get(SW1);
    if (keyvalue == 0) {
        changestatus = 1;
    } else {
        /* Do nothing */
    }
}
```

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

```
if (changestatus == 1) {
    Rate++;                     /* change duty rate */
    if (Rate >= DUTYMAX) {
        Rate = DUTYINIT;
    } else {
        /* Do nothing */
    }
    TMRB_ChangeDuty(TSB_TB12, Duty[Rate]);
    changestatus = 0;
} else {
```

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

デューティの算出方法:
Cycle = 500us, ftmrb = 1/8 fphiT0 = 12.5MHz, Ttmrb = 0.08us (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)

Duty = 10%: High 幅は 500*10% = 50us, Low 幅は 500-50 = 450us, カウンタ値 = 450us/Ttmrb = 0x15F9U
Duty = 25%: High 幅は 500*25% = 125us, Low 幅は 500-125 = 375us, カウンタ値= 375us/Ttmrb = 0x124FU

Duty = 50%: High 幅は 500*50% = 250us, Low 幅は 500-250 = 250us, カウンタ値= 250us/Ttmrb = 0xC35U

Duty = 75%: High 幅は 500*75% = 375us, Low 幅は 500-375 = 125us, カウンタ値 = 125us/Ttmrb = 0x61AU

Duty = 90%: High 幅は 500*90% = 450us, Low 幅は 500-450 = 50us, カウンタ値 = 50us/Ttmrb = 0x271U

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 Duty[5] = { 0x15F9U, 0x124FU, 0Xc35U, 0x61AU, 0x271U};
    /* duty: 10%, 25%, 50%, 75%, 90% */
```

7-17 TMRC

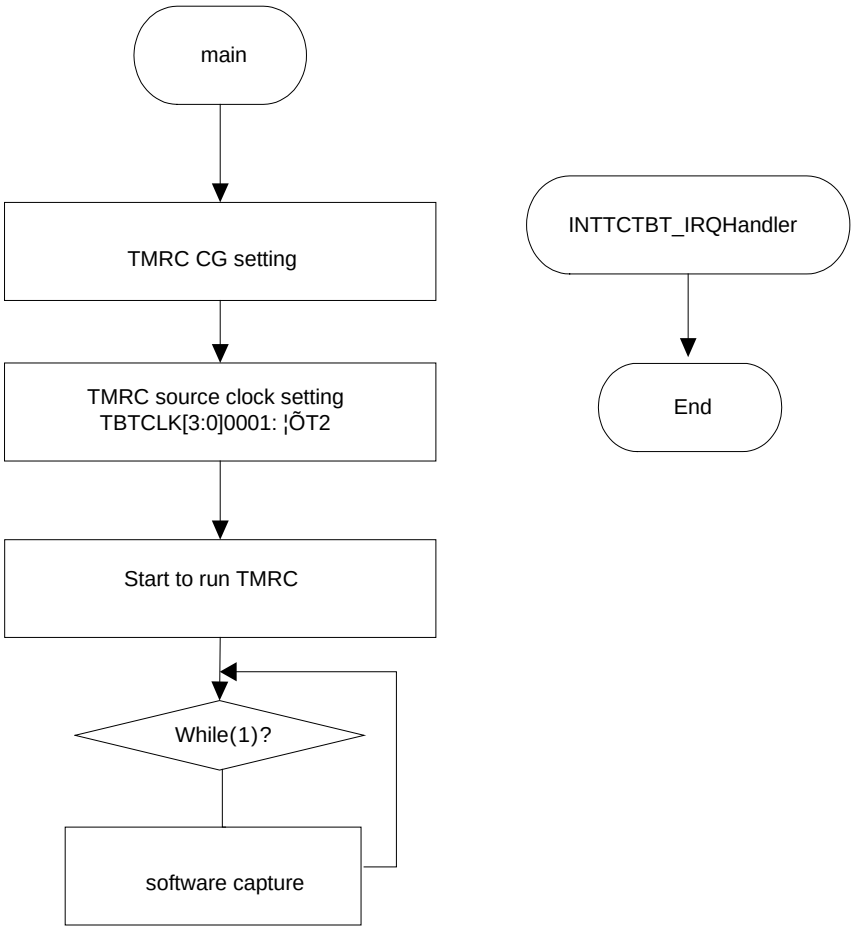
7-17-1 例: TMRC カウンタ

TX04 ペリフェラルドライバ (TMRC) 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. TMRC の初期化
2. ソフトウェアキャプチャー

- **フローチャート**



- **サンプルプログラムのコードと説明**

最初に TMRC CG を設定します。

fosc = 10MHz(外部高速クロック)

fpll = 10 * fosc = 100MHz (このサンプルプログラムでは PLL 逓倍値は 10 逓倍で設定されます。)

fperiph= fc = fpll = 100MHz

φT0(プリスケーラクロック) = fperiph(プリスケーラクロック選択)=100MHz

φT2(ソースクロック) = φT0/4 = 25MHz

```
CG_SetPhiT0Src(CG_PHIT0_SRC_FC);
CG_SetPhiT0Level(CG_DIVIDE_1);
CG_SetFcPeriphB(FC_B_ALL, ENABLE);
```

TMRB モジュールをイネーブルにします。ソースクロック設定を TBTCLK[3:0]0001: φT2*にします。割り込みをイネーブルにします。最後に TMRC の実行を開始します。

```
TMRC_Enable(TSB_TC);
TMRC_SetIdleMode(TSB_TC, ENABLE);
TMRC_SetSrcClk(TSB_TC, TMRC_CLK_DIV_4);
NVIC_EnableIRQ(INTTCTBT_IRQn);
TMRC_SetRunState(TSB_TC, TMRC_RUN);
```

その後メインルーチンは、カウンター値がオーバーフローしていると、“While(1)”へ移行します。オーバーフロー割り込み INTTCTBT が発生し、カウンター値は"0"にクリアされ、カウントを開始します。

```
void INTTCTBT_IRQHandler(void)
{
    /* overflow interrupt INTTCTBT occurred and the counter value will be
    cleared to "0" then start up counting*/
}
```

7-18 TMRD

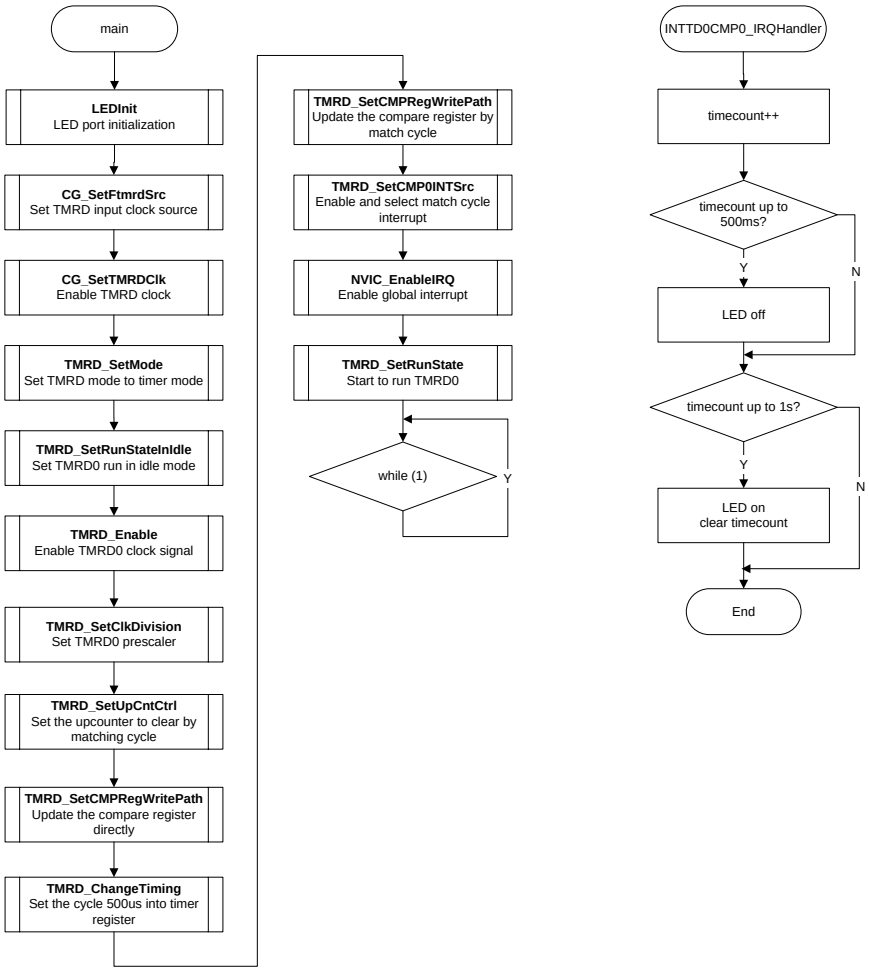
7-18-1 例: 汎用タイマ

TX04 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。

```
LED_Init();
```

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_TMRD_UNIT_A,
                    CG_FTMRD_SRC_HALF_FC);     /* clock is set to fc/2 */
CG_SetTMRDClk(CG_TMRD_UNIT_A, ENABLE);      /* enable the TMRDACLK */
```

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(TSB_TD, TMRD_MODE_BOTH_TMR);
```

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

```
/* TMRD IDLE mode */
TMRD_SetRunStateInIdle(TSB_TD, TMRD_UNIT_CH_0, 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_TD, TMRD_UNIT_CH_0);

/* Set the TMRD prescaler */
TMRD_SetClkDivision(TSB_TD, TMRD_UNIT_CH_0, TMRD_CLK_DIV_1);
```

サイクルと一致すると、TMRD0A 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_TDA, TMRD_UNIT_CH_0, 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 = 10 * fosc = 100MHz(サンプルプログラムでは、PLL 多重化の値は、10 逓倍です。)
ftmrd = fc/2 = 50MHz(サンプルプログラムでは、ftmrd = fc/2 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 = 25000 = 0x61A8

```
/* Update the compare register directly */
TMRD_SetCMPRegWritePath(TSB_TD,
                                    TMRD_UNIT_CH_0,
                                    TMRD_CMP_WRITE_DIRECT);

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

次に、API TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_INDIRECT)使用し、アップカウンター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_TD,
                                    TMRD_UNIT_CH_0,
```

```
                                        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_TD,
                       TMRD_UNIT_CH_0,
                       TMRD_INT_MATCH_CYCLE);

    NVIC_EnableIRQ(INTTD0CMP0_IRQn);
```

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

```
    /* Start to run TMRD */
    TMRD_SetRunState(TSB_TD, TMRD_UNIT_CH_0, 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 INTTDA0CMP0_IRQHandler(void)
{
    static uint16_t timecount = 0U;

    timecount++;
    if (timecount == 1000U) {     /* 500ms */
        LED_Off(LED_ALL);
    } else if (timecount == 2000U) {      /* 1s */
        timecount = 0U;_
        LED_On(LED_ALL);
    } else {
        /* Do nothing */
    }
}
```

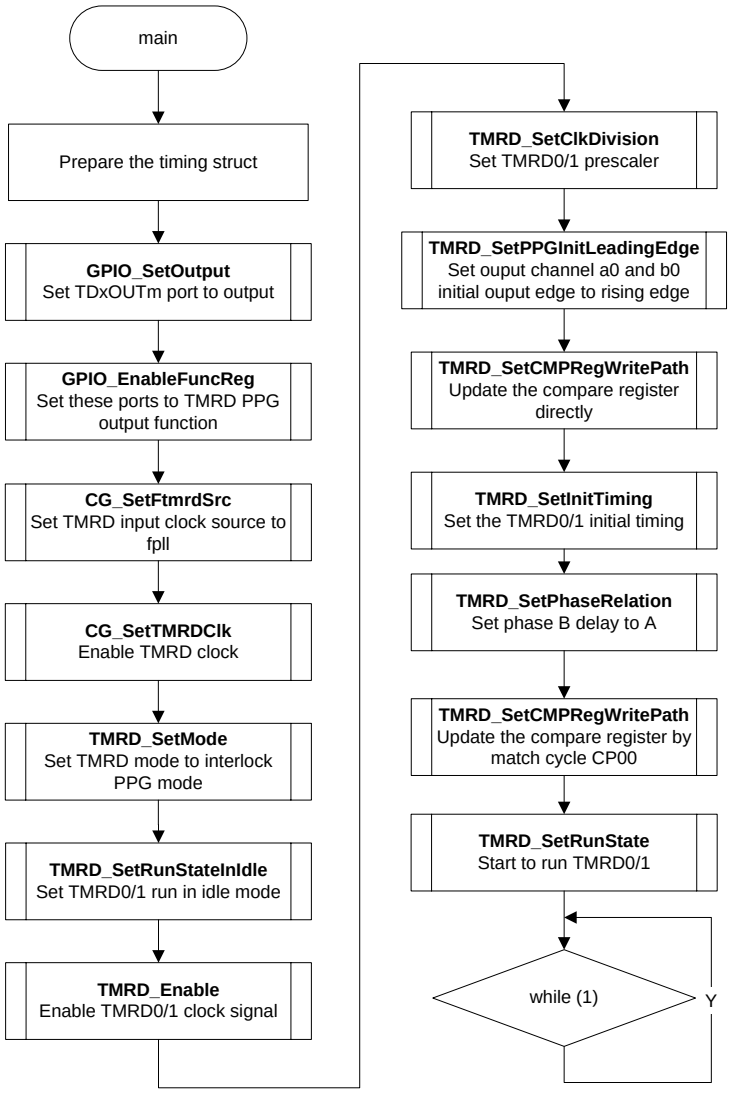
7-18-2 例: 連動 PPG 出力

TX04 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。

この例では以下を行います。

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**


main
Prepare the timing struct
GPIO_SetOutput
Set TDxOUTm port to output
GPIO_EnableFuncReg
Set these ports to TMRD PPG output function
CG_SetFtmrdSrc
Set TMRD input clock source to fpll
CG_SetTMRDClk
Enable TMRD clock
TMRD_SetMode
Set TMRD mode to interlock PPG mode
TMRD_SetRunStateInIdle
Set TMRD0/1 run in idle mode
TMRD_Enable
Enable TMRD0/1 clock signal
TMRD_SetClkDivision
Set TMRD0/1 prescaler
TMRD_SetPPGInitLeadingEdge
Set ouput channel a0 and b0 initial ouput edge to rising edge
TMRD_SetCMPRegWritePath
Update the compare register directly
TMRD_SetInitTiming
Set the TMRD0/1 initial timing
TMRD_SetPhaseRelation
Set phase B delay to A
TMRD_SetCMPRegWritePath
Update the compare register by match cycle CP00
TMRD_SetRunState
Start to run TMRD0/1
while (1)
Y

- **サンプルプログラムのコードと説明**

まず、TMRD パラメーター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 = 10 * fosc = 100MHz(このサンプルプログラムでは PLL の多重化の値は 10 逓倍です。)
ftmrd = fpll / 2 = 50MHz(この例では、ftmrd = fpll / 2 です。詳細は CG_SetFtmrdSrc()を参照してください)
設定値は 500us*ftmrd = 25000 = 0x61A8U です。

50%のデューティ:
LeadingTiming0 は 0x0000 に、TrailingTiming は Cycle/2=0x304D(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 = 0x208D(167us)に設定します。

```
#define TIME_CYCLE              ((uint16_t)0x61A8U)     /* 500us */
#define TIME_DUTY               ((uint16_t)0x30D4U)     /* 250us */
#define TIME_PHASE_SHIFT        ((uint16_t)0x208DU)     /* 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 Y 4,5,6,7 に設定します。

```
/* TD0OUT0, TD0OUT1, TD1OUT0, TD1OUT1 port setting */
GPIO_SetOutput(GPIO_PY,
                GPIO_BIT_4 | GPIO_BIT_5 | GPIO_BIT_6 | GPIO_BIT_7);
GPIO_EnableFuncReg(GPIO_PA,
                   GPIO_FUNC_REG_1,
                   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_TMRD_UNIT_A,
                CG_FTMRD_SRC_HALF_FC);      /* clock is set to fc/2 */
CG_SetTMRDClk(CG_TMRD_UNIT_A, ENABLE);      /* enable the TMRDACLK */
```

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(TSB_TD, TMRD_MODE_INTERLOCK_PPG_3CH);
```

TMRD_SetRunStateInIdle()を使用して、TMRD0A/1A を IDLE モードにします。

```
/* TMRD IDLE mode */
TMRD_SetRunStateInIdle(TSB_TD, TMRD_UNIT_CH_0, TMRD_RUN);
TMRD_SetRunStateInIdle(TSB_TD, TMRD_UNIT_CH_1, 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、プリスケーラーを設定します。連動 PPG モードにおいて、TMRD1A のプリスケーラー

は、TMRD0A の設定値と同じです。TMRD0 プリスケーラー設定のみ行ってください。

```
/* TMRD enable clock signal */
TMRD_Enable(TSB_TDA, TMRD_UNIT_CH_0);
TMRD_Enable(TSB_TDA, TMRD_UNIT_CH_1);
/* Set the TMRD prescaler */
TMRD_SetClkDivision(TSB_TDA, TMRD_UNIT_CH_0, 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(TSB_TD,
                           TMRD_PPG_CHANNEL_A0,
                           TMRD_WAVE_EDGE_RISING);
TMRD_SetPPGInitLeadingEdge(TSB_TD,
                           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_TD,
                        TMRD_UNIT_CH_0,
                        TMRD_CMP_WRITE_DIRECT);
TMRD_SetCMPRegWritePath(TSB_TD,
                        TMRD_UNIT_CH_1,
                        TMRD_CMP_WRITE_DIRECT);

/* Set the value to timer register */
TMRD_SetInitTiming(TSB_TD, TMRD_UNIT_CH_0, &timestruct);
TMRD_SetInitTiming(TSB_TD, TMRD_UNIT_CH_1, &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(TSB_TD, 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_TD,
                        TMRD_UNIT_CH_0,
                        TMRD_CMP_WRITE_INDIRECT);
TMRD_SetCMPRegWritePath(TSB_TD,
                        TMRD_UNIT_CH_1,
                        TMRD_CMP_WRITE_INDIRECT);
```

最後に、TMRD_SetRunState()を使用して、TMRD0A/1A 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、TMRD1A は TMRD0 の COUNTER0 と連動して動作を開始します。TMRD_SetRunState()により TMRD1A 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 TMRD1A を設定する必要はありません。

```
/* Start to run TMRD */
TMRD_SetRunState(TSB_TD, TMRD_UNIT_CH_0, TMRD_RUN);
```

7-19 SIO/UART

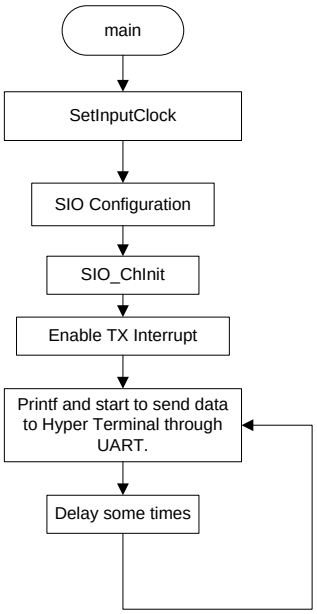
7-19-1 例: UART リターゲット

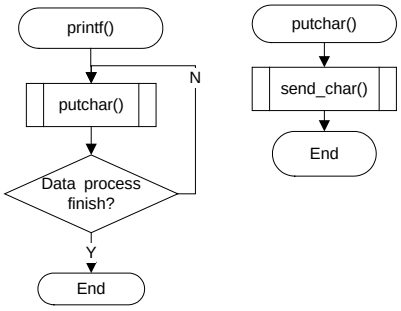
TX04 ペリフェラルドライバ(UART, GPIO)を使用したサンプルプログラムです。

この例では以下を行います。

1. UART 設定と初期化
2. UART 送信制御
3. データ送信に UART 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**





- **サンプルプログラムのコードと説明**

最初に入力クロックを設定します。

```
    UART_SetInputClock(UART0,0x01);
```

次に GPIO を設定し、UART を初期化します。

```
    TSB_RETG->FR1 |= RXD_BIT | TXD_BIT;
    TSB_RETG->CR |= TXD_BIT;
    TSB_RETG->IE |= RXD_BIT;
    TSB_RETG->PUP |= RXD_BIT | TXD_BIT;
```

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

```
    UART_InitTypeDef myUART;

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

上記設定を行い、その後、UART の送信割り込みを有効にします。

```
    NVIC_EnableIRQ(RETARGET_INT)
```

その後データ送信を開始します。ここでの TxBuffer は文字列です。

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

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

```
void INTTX0_IRQHandler(void)
{
    if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
        UART_SetTxData(UART_RETARGET,   gSIOTxBuffer[gSIORdIndex++]);          /*
send data */
        fSIO_INT = SET;              /* SIO0 INT is enable */
    } else {
        /* disable SIO0 INT */
        fSIO_INT = CLEAR;
```

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

printf()関数は IAR コンパイラの putchar()を、RealView コンパイラの fputc()をコールしてデータ出力を行います。

```
#if defined ( __CC_ARM      )          /* RealView Compiler */
struct __FILE {
        int handle;                     /* Add whatever you need here */
};
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE * f)
#elif defined ( __ICCARM__ )        /*IAR Compiler */
int putchar(int ch)
#endif
{
        return (send_char(ch));
}

uint8_t send_char(uint8_t ch)
{
        while (gSIORdIndex != gSIOWrIndex) {            /* wait for finishing sending */
            /* do nothing */
        }
        gSIOTxBuffer[gSIOWrIndex++] = ch;      /* fill TxBuffer */
        if (fSIO_INT == CLEAR) {        /* if SIO INT disable, enable it */
            fSIO_INT = SET;             /* set SIO INT flag */
            UART_SetTxData(UART_RETARGET, gSIOTxBuffer[gSIORdIndex++]);
            NVIC_EnableIRQ(RETARGET_INT);
        }

        return ch;
}
```

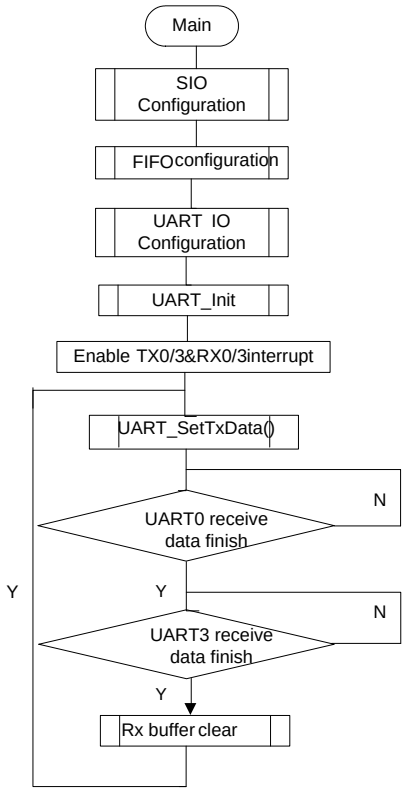
7-19-2 例: UART の FIFO サンプル

TX04 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 UART3 に設定します。

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
    if (SCx == TSB_SC0) {
        TSB_PH->CR |= GPIO_BIT_4;
        TSB_PH->FR1 |= GPIO_BIT_4;
        TSB_PH->FR1 |= GPIO_BIT_5;
        TSB_PH->IE |= GPIO_BIT_5;
    } else if (SCx == TSB_SC1) {
        TSB_PK->CR |= GPIO_BIT_4;
        TSB_PK->FR1 |= GPIO_BIT_4;
        TSB_PK->FR1 |= GPIO_BIT_5;
        TSB_PK->IE |= GPIO_BIT_5;
    } else if (SCx == TSB_SC2) {
        TSB_PE->CR |= GPIO_BIT_0;
        TSB_PE->FR2 |= GPIO_BIT_0;
        TSB_PE->FR2 |= GPIO_BIT_1;
        TSB_PE->IE |= GPIO_BIT_1;
    } else if (SCx == TSB_SC3) {
        TSB_PM->CR |= GPIO_BIT_4;
```

```
            TSB_PM->FR1 |= GPIO_BIT_4;
            TSB_PM->FR1 |= GPIO_BIT_5;
            TSB_PM->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、UART0/3 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(UART3);
     UART_Init(UART3, &myUART);
```

FIFO の設定を行います。

```
    UART_RxFIFOByteSel(UART0,UART_RXFIFO_RXFLEVEL);
    UART_RxFIFOByteSel(UART3,UART_RXFIFO_RXFLEVEL);

    UART_TxFIFOINTCtrl(UART0,ENABLE);
    UART_TxFIFOINTCtrl(UART3,ENABLE);

    UART_RxFIFOINTCtrl(UART0,ENABLE);
    UART_RxFIFOINTCtrl(UART3,ENABLE);

    UART_TRxAutoDisable(UART0,UART_RXTXCNT_AUTODISABLE);
    UART_TRxAutoDisable(UART3,UART_RXTXCNT_AUTODISABLE);

    UART_FIFOConfig(UART0,ENABLE);
    UART_FIFOConfig(UART3,ENABLE);

    UART_RxFIFOFillLevel(UART0,UART_FIFO_4B,UART_RXFIFO4B_FLEVLE_4_2B);
    UART_RxFIFOFillLevel(UART3,UART_FIFO_4B,UART_RXFIFO4B_FLEVLE_4_2B);

    UART_RxFIFOINTSel(UART0,UART_RFIS_REACH_EXCEED_FLEVEL);
    UART_RxFIFOINTSel(UART3,UART_RFIS_REACH_EXCEED_FLEVEL);

    UART_RxFIFOClear(UART0);
    UART_RxFIFOClear(UART3);

    UART_TxBufferClear(UART0);
    UART_TxBufferClear(UART3);

    UART_TxFIFOFillLevel(UART0,UART_FIFO_4B,UART_TXFIFO4B_FLEVLE_0_0B);
    UART_TxFIFOFillLevel(UART3,UART_FIFO_4B,UART_TXFIFO4B_FLEVLE_0_0B);
```

```
    UART_TxFIFOINTSel(UART0,UART_TFIS_REACH_NOREACH_FLEVEL);
    UART_TxFIFOINTSel(UART3,UART_TFIS_REACH_NOREACH_FLEVEL);

    UART_TxFIFOClear(UART0);
    UART_TxFIFOClear(UART3);
```

上記設定を行い、その後 UART0/3 の各割り込みを許可します。

```
    NVIC_EnableIRQ(INTTX0_IRQn);
    NVIC_EnableIRQ(INTRX3_IRQn);

    NVIC_EnableIRQ(INTTX3_IRQn);
    NVIC_EnableIRQ(INTRX0_IRQn);
```

最後に UART0/3 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 UART3 の送信割り込み処理ルーチンです。

```
void INTTX3_IRQHandler(void)
{
    volatile UART_Err err;

    if (TxCounter1 < NumToBeTx1) {
        UART_SetTxData(UART3, TxBuffer1[TxCounter1++]);
    } else {
        err = UART_GetErrState(UART3);
    }
}
```

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 UART3 の受信割り込み処理ルーチンです。

```
void INTRX3_IRQHandler(void)
{
    volatile UART_Err err;

    err = UART_GetErrState(UART3);
    if (UART_NO_ERR == err) {
        RxBuffer1[RxCounter1++] = (uint8_t) UART_GetRxData(UART3);
    }
}
```

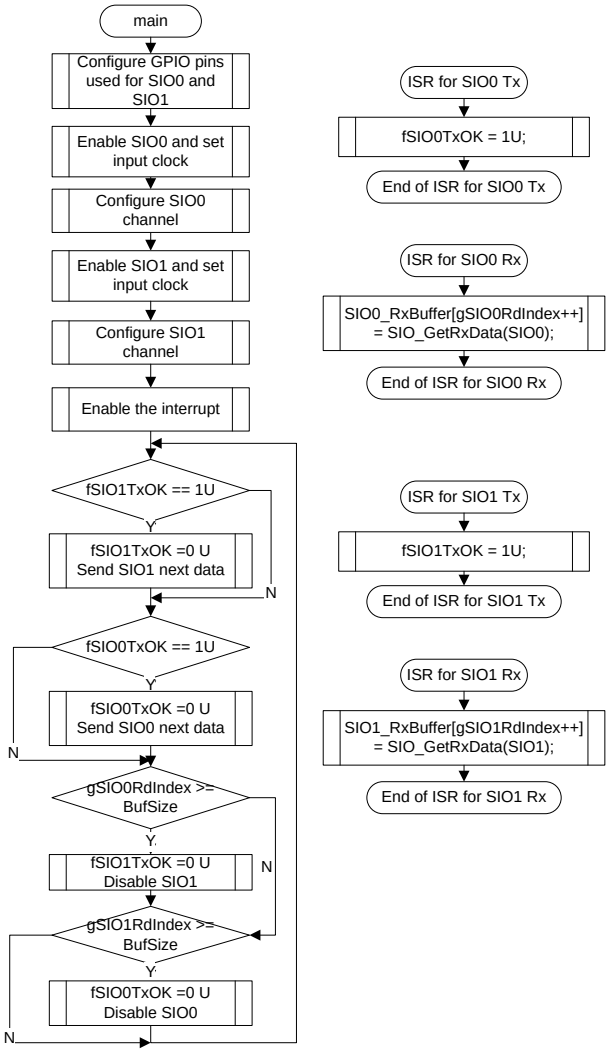
7-19-3 例: SIO サンプル

TX04 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);
/* Selects input clock for prescaler as PhiT0. */
SIO_SetInputClock(SIO0, SIO_CLOCK_T0);

/*initialize the SIO0 struct */
SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO0_Init.TIDLE = SIO_TIDLE_HIGH;
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_TS2;
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);
/* Selects input clock for prescaler as PhiT0. */
SIO_SetInputClock(SIO1, SIO_CLOCK_T0);

/*initialize the SIO1 struct */
SIO1_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO1_Init.TIDLE = SIO_TIDLE_HIGH;
SIO1_Init.TXDEMP = SIO_TXDEMP_HIGH;
SIO1_Init.EHOLDTime = SIO_EHOLD_FC_64;
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);
}
```

7-20 WDT

7-20-1 例: WDT サンプル

TX04 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();
```

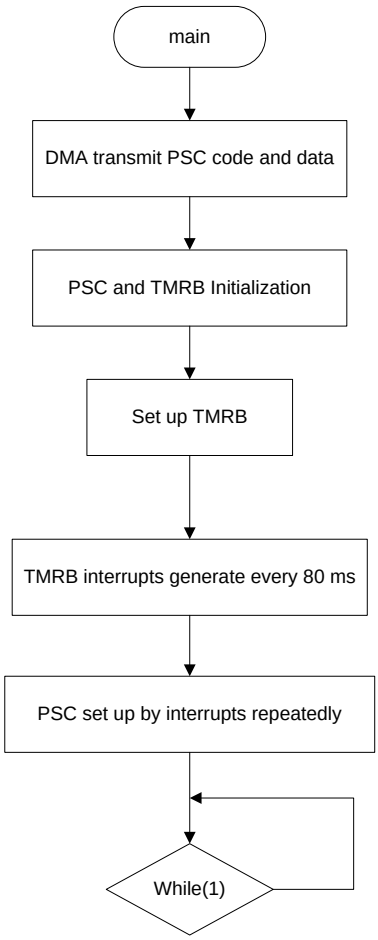
7-21 PSC

7-21-1 例: PSC Repeat Proc

TX04 ペリフェラルドライバ(PSC, DMAC, TMRB)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. リンクファイルを使用して PSC のコードとデータを指定された ROM 領域にリンクします。
2. DMAC 転送を使用して、PSC のコードとデータを PSC 用コード RAM とデータ RAM に転送します。
3. 80ms 毎に TMRB 割り込みが発生させます。
4. TMRB 割り込みが発生する毎に PSC が繰り返し起動します。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

以下はリンクファイルの定義内容です。

```
      /* PSC settings */
      define symbol __region_PSC_Code_ROM_start__     = 0x000BC000;
      define symbol __region_PSC_Code_ROM_end__       = 0x000BDFFF;
      define symbol __region_PSC_Data_ROM_start__     = 0x000BE000;
      define symbol __region_PSC_Data_ROM_end__       = 0x000BFFFF;

      define symbol __region_PSC_Code_RAM_start__     = 0x40010000;
      define symbol __region_PSC_Code_RAM_end__       = 0x40011FFF;
      define symbol __region_PSC_Data_RAM_start__     = 0x40014000;
      define symbol __region_PSC_Data_RAM_end__       = 0x40015FFF;

      define region ROM_PSC_Code_region = mem:[from __region_PSC_Code_ROM_start__ to __region_PSC_Code_ROM_end__];
      define region RAM_PSC_Code_region = mem:[from __region_PSC_Code_RAM_start__ to __region_PSC_Code_RAM_end__];
      define region ROM_PSC_Data_region = mem:[from __region_PSC_Data_ROM_start__ to __region_PSC_Data_ROM_end__];
      define region RAM_PSC_Data_region = mem:[from __region_PSC_Data_RAM_start__ to __region_PSC_Data_RAM_end__];

      initialize manually { section PSC_CodeBinary };
      place in ROM_PSC_Code_region   { section PSC_CodeBinary_init };
      place in RAM_PSC_Code_region   { section PSC_CodeBinary };

      initialize manually { section PSC_DataBinary };
      place in ROM_PSC_Data_region   { section PSC_DataBinary_init };
      place in RAM_PSC_Data_region   { section PSC_DataBinary };
```

DMAC の設定(メモリ→メモリ)
Flash ROM に配置する PSC 用コードを 0x4001_0000 番地に転送します。
同時にチェックサム用の変数 gPSC_codeChksum を初期化します。

```
      DMAC_InitTypeDef DMAC_InitStruct;
      DMAC_Channel myChannel = DMAC_CHANNEL_0;
      uint8_t      *pData;

      /* DMA channel3 configuration */
      DMAC_InitStruct.TxINT = ENABLE;
      DMAC_InitStruct.SrcAddr = (uint32_t) PSC_CODEROM;
      DMAC_InitStruct.SrcIncrementState = ENABLE;
      DMAC_InitStruct.SrcBitWidth = DMAC_WORD;
      DMAC_InitStruct.SrcBurstSize = DMAC_1_BEAT;
      DMAC_InitStruct.DstAddr = (uint32_t)PSC_CODEADDRSTA;
      DMAC_InitStruct.DstIncrementState = ENABLE;
      DMAC_InitStruct.DstBitWidth = DMAC_WORD;
      DMAC_InitStruct.DstBurstSize = DMAC_1_BEAT;
      DMAC_InitStruct.TxSize = PSC_CODETRANSSIZE;
      DMAC_InitStruct.TxDirection = DMAC_MEMORY_TO_MEMORY;

      /*Enable DMA circuit */
      DMAC_Enable(DMAC_UNIT_C);
      /* Set DMA endian */
      DMAC_SetTransferEndian(DMAC_UNIT_C, DMAC_LITTLE_ENDIAN);
      /*Initialize DMA channel */
      DMAC_Init(DMAC_UNIT_C, myChannel, &DMAC_InitStruct);
```

```
    DMAC_SetTxINTConfig(DMAC_UNIT_C,    myChannel,    DMAC_INT_TX_END,
ENABLE);

    pData = (uint8_t*)PSC_CODEADDRSTA;
    gPSC_codeChksum = 0x0000;                                   /* clear checksum */

    DMAC_SetDMAChannel(DMAC_UNIT_C, myChannel, ENABLE);


    do {
        GetPSC_DestAddr = TSB_DMACC->C0DESTADDR + 0x03;
        if (pData < (uint8_t*)GetPSC_DestAddr) {
            gPSC_codeChksum += *pData;
            pData++;
        }
    }     while     ((DMAC_GetTxINTReq(DMAC_UNIT_C,     myChannel)     !=
DMAC_TX_END_REQ)  ||  (pData  <  (uint8_t*)PSC_CODEADDREND));      /*  Was
calculation of the checksum completed? */

    DMAC_ClearTxINTReq(DMAC_UNIT_C, myChannel, DMAC_INT_TX_END);
```

Flash ROM に配置する PSC 用データを 0x4001_4000 番地にリンクします。
同時にチェックサム用の変数 gPSC _dataChksum を初期化します。

```
    / /* DMA channel3 configuration */
    DMAC_InitStruct.TxINT = ENABLE;
    DMAC_InitStruct.SrcAddr = (uint32_t) PSC_DATAROM;
    DMAC_InitStruct.SrcIncrementState = ENABLE;
    DMAC_InitStruct.SrcBitWidth = DMAC_WORD;
    DMAC_InitStruct.SrcBurstSize = DMAC_1_BEAT;
    DMAC_InitStruct.DstAddr = (uint32_t)PSC_DATAADDRSTA;
    DMAC_InitStruct.DstIncrementState = ENABLE;
    DMAC_InitStruct.DstBitWidth = DMAC_WORD;
    DMAC_InitStruct.DstBurstSize = DMAC_1_BEAT;
    DMAC_InitStruct.TxSize = PSC_DATATRANSSIZE;
    DMAC_InitStruct.TxDirection = DMAC_MEMORY_TO_MEMORY;

    /*Enable DMA circuit */
    DMAC_Enable(DMAC_UNIT_C);
    /*Initialize DMA channel */
    DMAC_Init(DMAC_UNIT_C, myChannel, &DMAC_InitStruct);
    DMAC_SetTxINTConfig(DMAC_UNIT_C,    myChannel,    DMAC_INT_TX_END,
ENABLE);

    pData = (uint8_t*)PSC_DATAADDRSTA;
    gPSC_dataChksum = 0x0000;                                   /* clear checksum */

    DMAC_SetDMAChannel(DMAC_UNIT_C, myChannel, ENABLE);

    do {
        GetPSC_DestAddr = TSB_DMACC->C0DESTADDR + 0x03;
        if (pData < (uint8_t*)GetPSC_DestAddr) {
            gPSC_dataChksum += *pData;
            pData++;
        }
    }     while     ((DMAC_GetTxINTReq(DMAC_UNIT_C,     myChannel)     !=
DMAC_TX_END_REQ)  ||  (pData  <  (uint8_t*)PSC_DATAADDREND));      /*  Was
```

```
calculation of the checksum completed? */

    DMAC_ClearTxINTReq(DMAC_UNIT_C, myChannel, DMAC_INT_TX_END);
```

TMRB の設定
割り込み間隔を 80ms に設定します。

```
    TMRB_InitTypeDef m_tmrb;

    m_tmrb.Mode = TMRB_INTERVAL_TIMER;
    m_tmrb.ClkDiv = TMRB_CLK_DIV_128;
     /* periodic time is 80 ms */
     m_tmrb.TrailingTiming = SAMPLINGTIME;
     m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;
     /* periodic time is 80 ms */
     m_tmrb.LeadingTiming = SAMPLINGTIME;

    TMRB_Enable(TSB_TB9);
    TMRB_Init(TSB_TB9, &m_tmrb);
    NVIC_EnableIRQ(INTTB09_IRQn);
```

TMRB 割り込みハンドラ
TMRB9 割り込みのイベントフラグをクリアします。

```
    void INTTB09_IRQHandler(void)
    {
        PSC_ClearEventOverrunFlag(PSC_TRIGGER_EVENT_0);
    }
```

PSC の設定
リピートモードに設定します。

```
    PSC_SetRepetProcVectPointer(PSC_STARTADDR);
    PSC_SetTriggerEventFUNC(PSC_TRIGGER_EVENT_0, ENABLE);
```

TMRB 動作の開始

```
    TMRB_SetRunState(TSB_TB9, TMRB_RUN);
```

PSC の処理
データの定義を行います。

```
    dsec1 section romdata abs=0x40114000
    data1    dd    0x12340000
    ;
    dsec2 section romdata abs=0x40114010
    data2    dd    0x00005678
```

A0 レジスタに 0x1234_0000 を、R0 レジスタに 0x00005678 をそれぞれ設定します。A0 と R0 を足し合わせた結果を M0 に格納します。

```
    csec1 section code abs=0x40110000
        mvi      AP0,data1
        ld       A0,(AP0)
        mvi      AP0,data2
        ld       R0,(AP0)
        add      M0
        jmp      lab_csec2nop
```

A0 レジスタに 0x1234_0000 を、R0 レジスタに 0x00005678 をそれぞれ設定します。A0 か

ら R0 を差し引きし、結果を M1 へ格納して終了します。

```
csec2 section code abs=0x40110100
lab_csec2:
        mvi     AP0,data1
        ld      A0,(AP0)
        mvi     AP0,data2
        ld      R0,(AP0)
        sub     M1
;
        stop
```