TOSHIBA

TX04 ペリフェラルドライバ使用例 (TMPM46B)

第 1.000 版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M46BUE-01xJ

TOSHIBA

本製品取り扱い上のお願い

- ソフトウエア使用権許諾契約書の同意無しに使用しないで下さい。

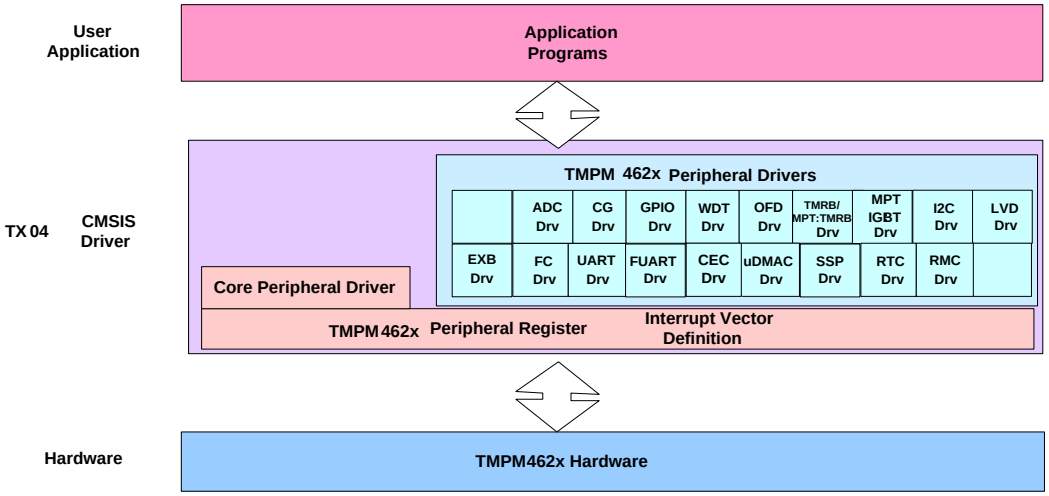
目次

1 はしがき

本サンプルプログラムは、東芝製マイコンTMPM46B用です。各サンプルプログラムは、主なMCU内蔵機能を単独で実行するように出来ています。サンプルプログラム内の一部を取り出して再利用することで、必要な機能を動作させることができます。

2 概要

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	チャネル	使用/未使用
CG	クロックギア	使用
	PLL	PLL6 逓倍
低消費電力モード	-	使用 (IDLE / STOP1 / STOP2 モード)
SysTick	-	未使用
ウォッチドッグタイマ (WDT)	-	使用
外部割込み (INT)	INT0	未使用
	INT1	未使用
	INT2	未使用
	INT3	未使用
	INT4	未使用
	INT5	未使用
	INT6	未使用

機能	チャネル	使用/未使用
	INT7	未使用
	INT8	未使用
	INT9	未使用
	INTA	未使用
	INTB	未使用
	INTC	未使用
	INTD	未使用
	INTE	未使用
	INTF	使用(CG サンプルにおける STOP モードからの復帰)
	INTRTC	使用(RTC)
	INTRX0	使用(SIO/UART)
	INTTX0	使用(SIO/UART)
	INTRX1	使用(SIO/UART)
	INTTX1	使用(SIO/UART)
シリアルチャネル(SIO/UART)	SIO0	使用(リターゲット、SIO 通信制御、UART FIFO)
	SIO1	使用(SIO 通信制御)
	SIO2	未使用
	SIO3	使用(UART FIFO)
16 ビットタイマ/イベントカウンタ(TMRB)	TMRB0	使用(汎用タイマ)
	TMRB1	未使用
	TMRB2	未使用
	TMRB3	未使用
	TMRB4	未使用
	TMRB5	未使用
	TMRB6	使用(PPG 出力)
	TMRB7	未使用
電圧検知回路(LVD)	-	使用(LVD)
12ビットA/Dコンバータ	AIN0	使用(ADC)
	AIN1	未使用
	AIN2	未使用
	AIN3	未使用
	AIN4	未使用
	AIN5	未使用
	AIN6	未使用
	AIN7	未使用
uDMA コントローラ	-	使用(DMAC)
リアルタイムクロック(RTC)	-	使用(RTC)
I2C バス	I2C0	使用(I2C 受信)

機能	チャネル	使用/未使用
	I2C1	未使用
	I2C2	使用(I2C 送信)
同期式シリアルインタフェース(SSP)	SSP0	使用(SSP0)
	SSP1	未使用
	SSP2	未使用
非同期式シリアルインタフェース(UART)	FUART0	使用(Full UART)
	FUART1	未使用
16 ビット多目的タイマ(MPT)	MPT0	使用(IGBT: PPG 出力)
	MPT1	未使用
	MPT2	未使用
	MPT3	未使用
外部バスインタフェース(EBIF)	-	使用(EXB)
AES	-	使用(AES サンプル)
SHA	-	使用(SHA サンプル)
MLA	-	使用(MLA サンプル)
ESG	-	使用(ESG サンプル)

4 端子用途

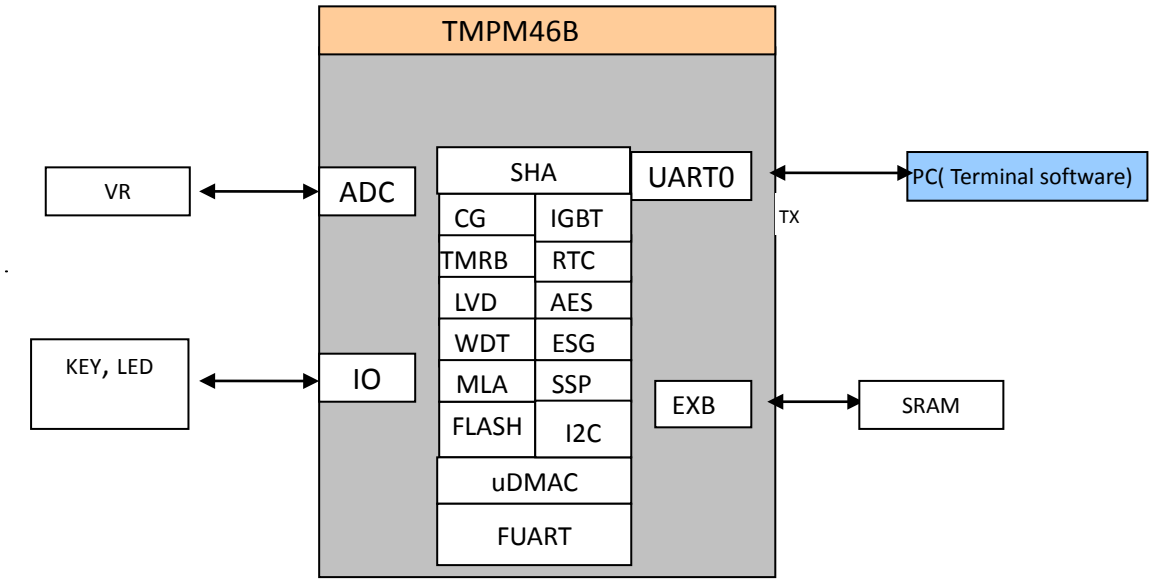
本サンプルプログラムは、開発環境に東芝製TMPM46B用評価ボードを用いてテストされています。
以下に、端子用途を説明します。

Pin No.	Name	Usage
23	PF0	EBIF AD0 / FUART UT0CTS
24	PF1	EBIF AD1 / FUART UT0TXD
25	PF2	EBIF AD2 / FUART UT0RXD
26	PF3	EBIF AD3 / FUART UT0RTS
27	PF4	EBIF AD4
28	PF5	EBIF AD5
29	PF6	EBIF AD6
30	PF7	EBIF AD7
31	PG0	EBIF AD8
32	PG1	EBIF AD9
33	PG2	EBIF AD10
34	PG3	EBIF AD11
35	PG4	EBIF AD12
36	PG5	EBIF AD13
37	PG6	EBIF AD14
38	PG7	EBIF AD15
88	PB0	LED0 / SC3TXD

87	PB1	LED1 / SC3RXD
43	PB2	LED2 / EBIF WR
44	PB3	LED3 / EBIF RD
1	PJ0	SW0/SW4 / AIN0
2	PJ1	SW1/SW5
3	PJ2	SW2/SW6
4	PJ3	SW3/SW7
13	PE0	EBIF A16
14	PE1	EBIF A17 / SC0RXD
15	PE2	EBIF A18 / SC0TXD
16	PE3	EBIF A19 / SC0SCK
17	PE4	EBIF A20 / SC1SCK
18	PE5	EBIF A21 / SC1TXD
19	PE6	EBIF A22 / SC1RXD
20	PE7	EBIF A23
46	PB5	EBIF ALE
51	PH1	EBIF CS1 / I2C2SCL
47	PB6	EBIF BELL
52	PH0	EBIF BELH / I2C2SDA
31	PG0	IGBT MT0IN
34	PG3	IGBT MT0OUT0
33	PG2	IGBT MT0OUT1
32	PG1	IGBT GEMG0
80	PK2	I2C0SDA
79	PK3	I2C0SCL
81	PK1	TB6OUT
59	X1	High frequency resonator connection pin
61	X2	High frequency resonator connection pin
69	XT1	Low frequency resonator connection pin
70	XT2	Low frequency resonator connection pin
62	MODE	MODE pin
58	RESET	Reset signal input pin
47	BOOT	BOOT mode control pin

5 開発環境

下記に開発環境の構成を示します。



1. 開発ツール:
   - IAR:
     1) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
     2) IDE:IAR Embedded workbench 7.30.4 version
   - KEIL:
     1) IDE: KEIL uVision 5.14

6 機能

6-1 動作モード選択

本デバイスは 4 つの動作モードがあります: NORMAL, IDLE, STOP1, STOP2

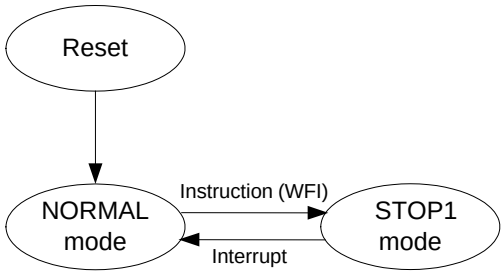
- **NORMAL モード:**

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

IDLE, STOP1, STOP2 の各モードは低消費電力モードです。
低消費電力モードへ移行するには、システ制御レジスタ CGSTBYCR<STBY[2:0]>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。
補足: STOP1 モードのみサンプルプログラムに含まれます。

- **STOP1 モード:**

STOP1モードでは、すべての内蔵回路が停止します。STOP1モードが解除されると、STOP1モードへ移行する直前の動作モードへ復帰し、動作を開始します。
補足: STOP1 モードのサンプルプログラムは CG の例に含まれています



6-2 ADC

ADC 入力チャネルに接続された *VR2* の値を変換します。この電圧を ADC で測定し、ボード上の 4 LED を点滅させます。
電圧レベルが高いと点滅間隔が短くなります。

動作シーケンス

1. AD コンバータをソフトウェアリセットします。
2. AD コンバータ用クロック供給を許可します。
3. AD コンバータ用プリスケーラークロックを設定し、ホールド時間をサンプリングします。
4. AD コンバータ入力チャネル: AIN0 を選択します。
5. AD コンバータリピートモードを許可します。
6. 割り込みモードを設定します。

7. VREF を ON にセットします。
8. AD 変換開始します。
9. AD 変換が完了したら、結果を取得し、LED ループ点滅スピードの調整に使用します。

補足: LED0~LED3 は PB0~PB3 と接続します。

6-3 AES

6-3-1 ECB モード

ECB モードによる暗号化と複合化を行うサンプルプログラムです。
AES 回路のアルゴリズムを ECB モードに設定します。平文を 384 ビットで暗号化し、その後、この暗号化データを復号化します。復号化されたテキストと平文が同じ場合は正常、等しくなければ失敗の判定を行います。
AES 回路は 128 ビットのブロック単位で、平文を暗号化、または暗号化されたテキストを復号化します。
FIFO へのデータ書き込み、または FIFO からのデータ読み出しは、DMA コントローラで行います。

6-3-2 CBC モード

CBC モードによる暗号化と復号化を行うサンプルプログラムです。
AES 回路のアルゴリズムを CBC モードに設定します。平文を 384 ビットで暗号化し、その後、この暗号化データを復号化します。復号化されたテキストと平文が同じ場合は正常、等しくなければ失敗の判定を行います。
AES 回路は 128 ビットのブロック単位で、平文を暗号化、または暗号化されたテキストを復号化します。
FIFO へのデータ書き込み、または FIFO からのデータ読み出しは、CPU が行います。

6-3-3 CTR モード

CTR モードによる暗号化と復号化を行うサンプルプログラムです。
AES 回路のアルゴリズムを CTR モードに設定します。平文を 384 ビットで暗号化し、その後、この暗号化データを復号化します。復号化されたテキストと平文が同じ場合は正常、等しくなければ失敗の判定を行います。
AES 回路は 128 ビットのブロック単位で、平文を暗号化、または暗号化されたテキストを復号化します。
FIFO へのデータ書き込み、または FIFO からのデータ読み出しは、CPU が行います。

6-4 CG

6-4-1 STOP1

CPU の動作モードを変更します。NORMAL と STOP1 の 2 モードを使用します。スイッチによって動作モードを切り替えます。

現在のモード	アクション(スイッチ)	動作	LED 表示
NORMAL	SW1 を ON	NORMAL→ STOP1	UART 出力 “NORMAL MODE” →”STOP MODE” LED 0,1,2,3 を消灯します。
STOP1	まず SW1 を OFF し、次に SW0 を ON します。	STOP1 →NORMAL	UART 出力 “STOP MODE” →” NORMAL MODE” LED 0,1,2,3 を点灯します。

6-4-2 IDLE

CPU の動作モードを変更します。NORMAL と IDLE の 2 モードを使用します。スイッチによって動作モードを切り替えます。

現在のモード	アクション(スイッチ)	動作	LED 表示
NORMAL	SW1 を OFF	NORMAL→ IDLE	LED 3 消灯 LED 2 点灯
IDLE	SW1 を ON し、次に SW0 を ON します。	IDLE →NORMAL	LED 3 点灯 LED 2 消灯

6-4-3 STOP2

CPU の動作モードを変更します。NORMAL と STOP2 の 2 モードを使用します。スイッチによって動作モードを切り替えます。

現在のモード	アクション(スイッチ)	動作	LED 表示
NORMAL	SW1 を ON	NORMAL→ STOP2	LED 0, 3 消灯
STOP2	SW1 を OFF し、次に SW0 を ON します。	STOP2 →NORMAL	LED 0, 3 点灯

補足:
LED0~LED3 を PB0~PB3 と接続します。
SW0 と PC1 (Pin No.76)を接続し、外部割り込み(INTF)を設定します。
SW1 と PJ1 (Pin No.2)を接続します。

6-5 ESG

512 ビットの乱数シードを生成するサンプルプログラムです。
乱数シードのラッチタイミングと乱数シードの出力タイミングを設定し、ESG コアを動作させます。
ESG コアが動作を終了すると、512 ビットの乱数シードを読み出します。

6-6 EXB

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 124K バイトの IS61LV6416 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. TMPM46B 評価ボードの AD[15:0]と IS61LV6416 SRAM の AD[15:0]を接続します。
2. TMPM46B 評価ボードの A16 と IS61LV6416 SRAM の A15 を接続します。
3. TMPM46B 評価ボードの CS1、WE、RD、ALE、BELL と IS61LV6416 SRAM の CE、WE、OE、ALE、LB、UB を接続します。

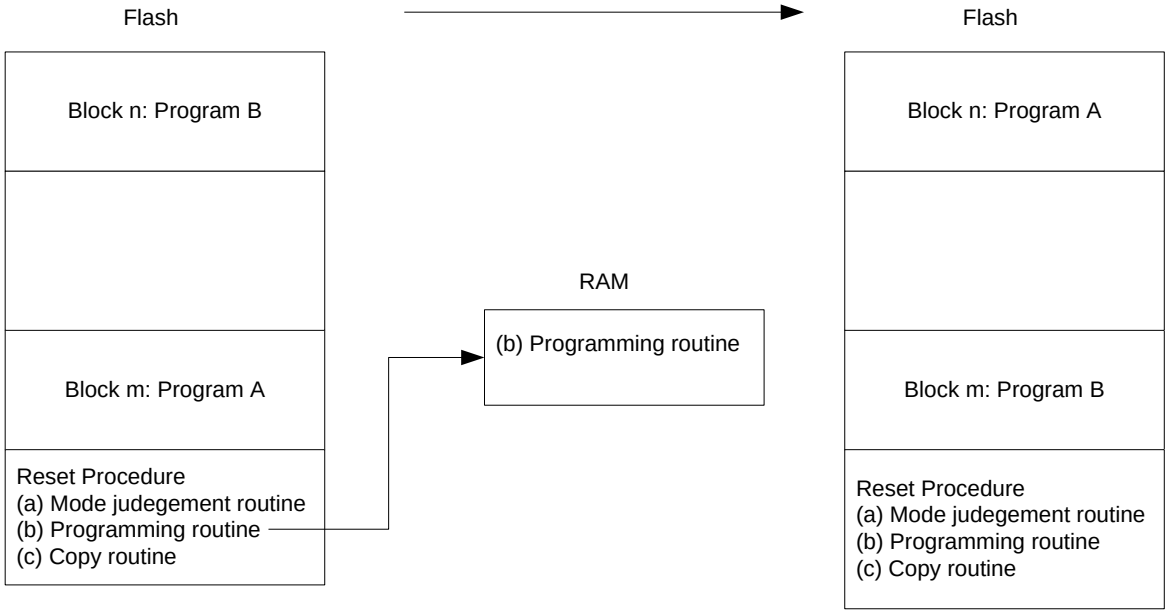
6-7 FLASH

6-7-1 FLASH_UserBoot

このアプリケーションは、内蔵フラッシュメモリの消去及び再書き込み操作を行います。
このプログラムは、シングルチップモードのノーマルモードとユーザーブートモードで実行します。
ーリセット動作: モード判定、書き込みルーチン(フラッシュ API)、転送ルーチンで構成されます
・モード判定ルーチン: ユーザブートモードまたはノーマルモードの判定を行います。
・転送ルーチン: 書き込みルーチンを Flash から RAM へ転送します。
・書き込みルーチン: RAM 上で動作し、フラッシュ上のプログラム A とプログラム B のコードを入れ替えます。
ープログラム A/B (リセット動作)は事前にフラッシュメモリに書き込まれています。
ープログラム A/B(A: LED0 点滅、LED2 点灯
B: LED1 点滅、LED3 点灯)
初期状態は、プログラム A が最初に動作します。
ーSW4 はリセット動作のモード判定に使います。

SW4 が押された場合 → ユーザーブートモードです。

SW4 が押されていない場合 → Normal モードです。



動作シーケンス

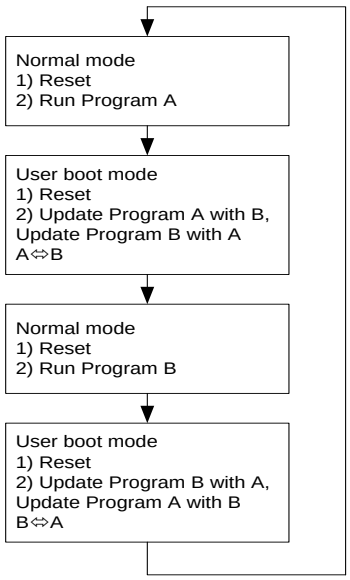
(1) 電源投入
LED0 blinks and LED2 Always show.
評価ボードのリセット動作が行われます。
初期プログラム A がフラッシュ ROM へ格納され動作します。
LED0 が点滅し、LED2 は点灯します。

(2) SW4 を押している間にリセットを押してください。その後、SW4 を離してください。
評価ボードはリセット処理を行います: 書き込みプログラムが転送処理によって RAM に転送されます。
その後、フラッシュ内のプログラム A と B が入れ替わります。

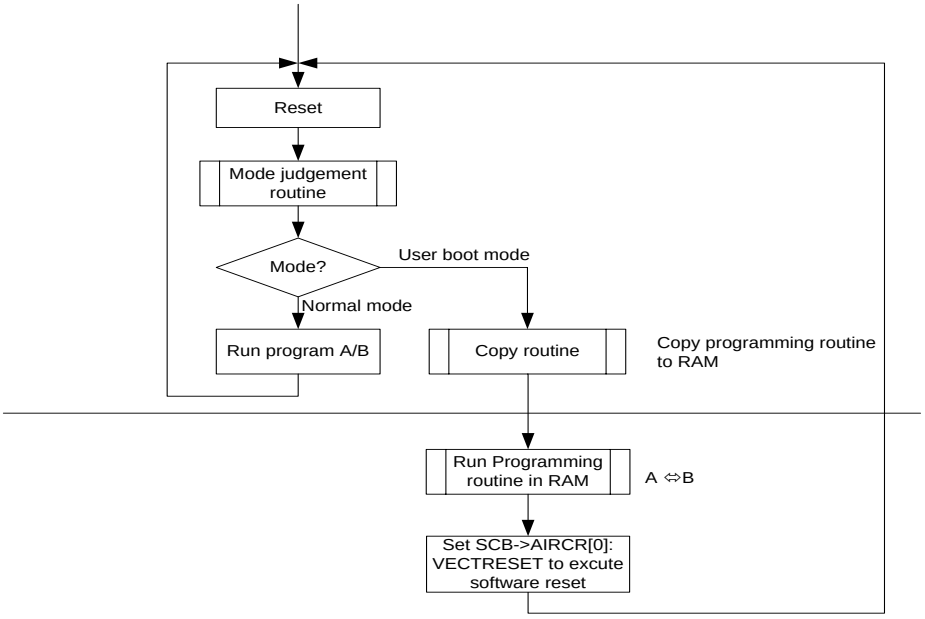
(3) 評価ボードは自動的にソフトウェアによってリセットします。
その後、プログラム B がフラッシュ ROM へ格納され動作します。
LED1 が点滅し、LED3 が点灯します。

(4) SW4 を押しながらリセットします。その後 SW4 を離すと、手順(2)を行います。

(5) 評価ボードはソフトウェアによって自動的にリセットします。
その後、フラッシュ ROM へ格納されたプログラム A が動作します。
LED0 が点滅し、LED2 が点灯します。



フローチャート



6-7-2 FLASH_Swap

このサンプルプログラムはフラッシュメモリ操作用のドライバを使ってスワップ動作を自動的に行います。

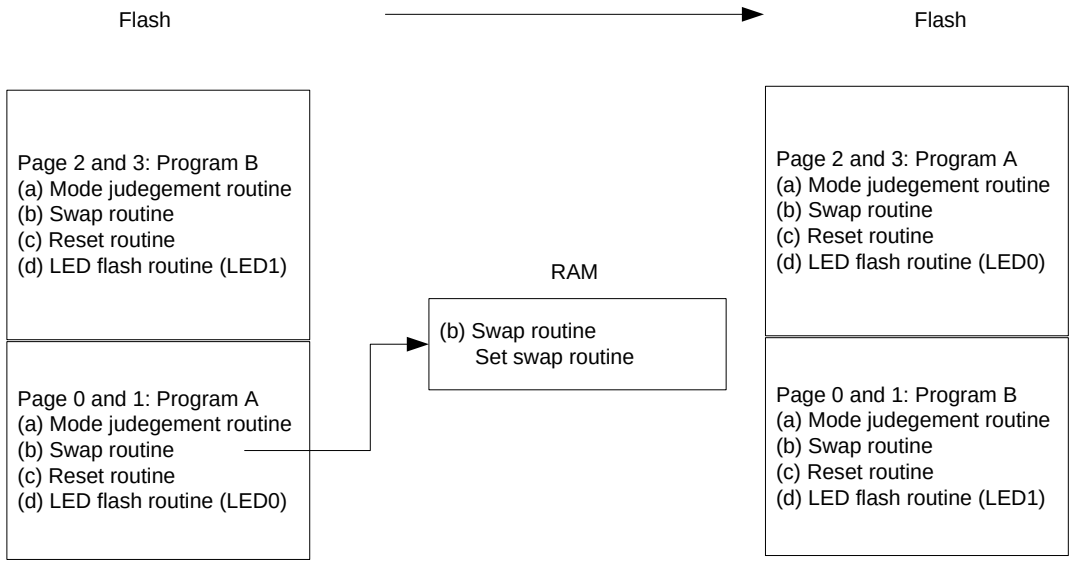
このプログラムは、シングルチップモードのノーマルモードとユーザーブートモードで実行します。
・モード判定ルーチン: ユーザブートモードまたはノーマルモードの判定を行います。
・スワップルーチン: スワップコマンドとスワップ解除コマンドを実装したプログラムを RAM 実行します。
・LED フラッシュルーチン: LED フラッシュプログラムはノーマルモードで実行します。

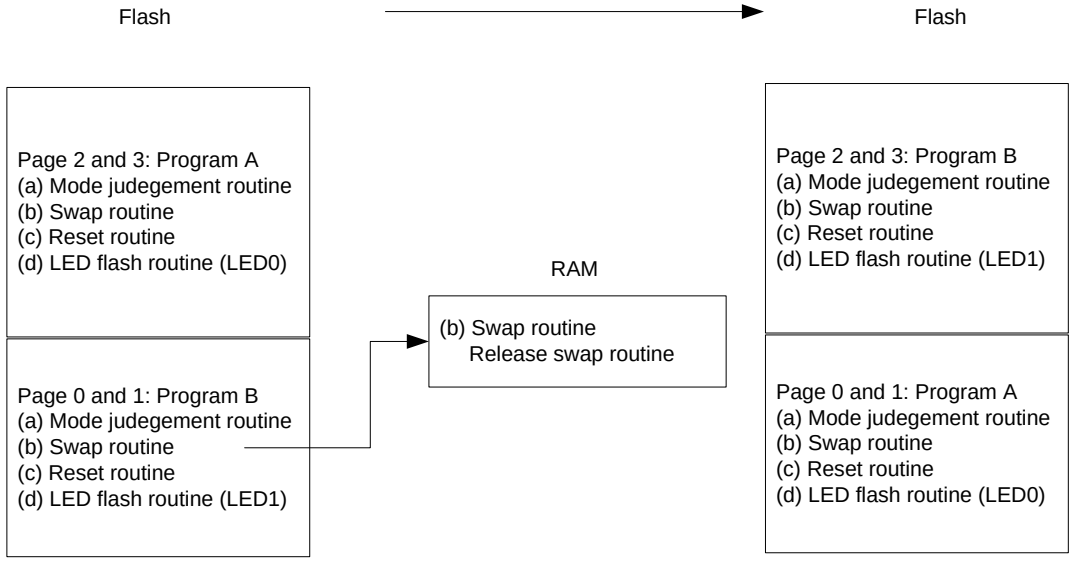
—プログラム A/B は事前にフラッシュメモリの任意の Block に書き込まれています。
—プログラム A/B は LED フラッシュルーチン以外同じです。
—プログラム A は LED0 を点灯し、プログラム B は LED1 を点灯します。
初期状態は、プログラム A が最初に動作します。
－SW4ーはリセット動作のモード判定に使います。

SW4 が押された場合 → ユーザーブートモードです。

SW4 が押されていない場合 → Normal モードです。

－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。
－プログラム A/B (A: LED0 点滅、LED2 点灯
B: LED1 点滅、LED3 点灯)





動作シーケンス:

1) IAR 社の EWARM を例に説明します。
   EWARM を立ち上げ、PC と J-Link、J-Link と評価ボードを接続します。その後、
   "Flash_Swp_A"ワークスペースを開きます。
2) メインメニュー[Project]から[Options]を選択し、オプションダイアログを開きます。
   [Debugger]から[Download]メニューをクリックし、"Override default .board file"チェックボックスをオンします。
   スボタンをクリックして、"Flash_Swp_A\IAR\res\Flash_Swp_A_for_M46B.board"を選択し、OK ボタンをクリックしてダイアログを閉じます。
3) メインメニュー[Project]から[Download]-[Download file]を選択し、その後
   "Flash_Swp_A\IAR\res\Flash_Swp_A.out"を指定し、"open"をクリックします。フラッシュローダーは Flash_Swp_A を指定されたブロックに書き込みます。
4) 2)と 3)を繰り返します。board ファイルは
   "Flash_Swp_A\IAR\res\Flash_Swp_B_for_M46B.board"を指定します。
   "Flash_Swp_A\IAR\res\Flash_Swp_B.out"は、Flash_Swp_B を指定されたブロックへダウンロードします。
5) 評価ボードのリセットボタンをオン->オフします。LED0 が点滅します。
6) SW4 をオンしたまま、リセットボタンをオン->オフします。スワップ処理が終了すると LED2 が点灯します。
7) SW4 をオフします。LED1 が点滅します。
8) SW4 をオンしたまま、リセットボタンをオン->オフします。スワップ解除処理が終了すると、LED2 が点灯します。
9) SW4 をオフします。LED0 が点滅します。

補足:
<1> 動作を再確認する場合は手順 6)から手順 9)を繰り返します。
<2> Flash_Swp_A.out と Flash_Swp_B.out はアプリケーションにあわせて修正してお使いください。
<3> LED0~LED2 は PB0~PB2 と接続します。
<4> SW4 は PJ0 と接続します。

6-8 FUART

本プログラムにおいて、フル UART 送信と FIFO 受信が可能です。
本プログラムは 64 種の異なるデータをフル UART チャネル 0 の UT0TXD 端子から送信し、UT0RXD 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 UT0RXD 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、UART に"SAME"を出力します。
受信データが送信データと異なる場合、UART に“DIFFERENT”を出力します。
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である場合、UT0RTS と UT0CTS ハードウェアフロー制御をイネーブルにしてください。

2 回目:

受信データが送信データと異なる場合、UT0RTS と UT0CTS ハードウェアフロー制御をディセーブルにしてください。

補足:

評価ボードの UT0TXD 端子と UT0RXD 端子、UT0RTS 端子と UT0CTS 端子を接続してください。

SW0 は PJ0 と接続します。

LED0~3 は PB0~3 と接続します。

6-9 GPIO

このサンプルプログラムはペリフェラルドライバの GPIO を使用し、LED の設定、LED の点灯/消灯を行います。

補足:

LED0~LED3 は PB0~PB3 と接続します。

SW0~SW3 は PJ0~PJ3 と接続します。

6-10 I2C

このサンプルプログラムは、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います。

評価ボードの 2 本の I2C バス (I2C0 & I2C2) 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 バスのスレーブ側

I2C スレーブ (I2C0) は I2C マスタ (I2C2)から “TOSHIBA”を受信します。

受信結果は UART 出力にて確認できます。

I2C2 (マスタ) と I2C0 (スレーブ) サンプルの開始:

```
K3:I2C2 to I2C0
```

SW7 をオンすると、文字列 “TOSHIBA” を I2C2 (マスタ) から I2C0 (スレーブ) へ送信します。

```
Write Over
K1: Show I2C0
```

SW4 をオンすると文字列 “TOSHIBA” を I2C0 (スレーブ) が受信します。

```
TOSHIBA
I2C2 to I2C0 OK
```

補足:
SW4 は PJ0 と接続します。
SW7 は PJ3 と接続します。
PK2 は I2CxSDA として使用し、PH0 と接続します。
PK3 は I2CxSCL として使用し、PH1 と接続します。

6-11 IGBT

6-11-1 1 相 PPG 出力

このサンプルプログラムは、IGBT ドライバと EMG 機能により PPG 出力を制御するため、外部トリガをどのように使用するかを表します。MPT チャネル 0 (IGBT0) を使用し、PPG 波形が MT0OUT0 から生成されます。

ピン配置:

機能	端子名	端子番号	入出力ポート
Trigger input	MT0IN	31	PG0
PPG output	MT0OUT0	34	PG3
EMG input	GEMG0	32	PG1

動作シーケンス:

1. ポート GEMG0 を”High”にするため、DVDD と接続します。
2. MT0OUT0 をオシロスコープに接続してください。
3. 電源投入し、LED0 の点灯、MT0OUT0 から波形が無いことを確認します。
4. MT0IN を GND に接続すると、立下りエッジが生成されますので“Low”に保ちます。
5. IGBT タイマは、立下りエッジのため動作を開始します。同時に PPG 波形を MTOUT00 から出力します。PPG サイクルは 50us で、デューティ 50%です。
6. GEMG0 を GND に接続します。その後すぐ PPG 出力が止まり、LED1 が点灯します。
7. GEMG0 を DVDD に再度接続します。LED1 をオフして EMG 状態をキャンセルし、その後手順 4 を繰り返すと、再度波形を出力します。

6-11-2　2 相 PPG 出力

このサンプルプログラムは、IGBT ドライバと EMG 機能により、2 相 PPG 出力を制御するコマンドの開始方法を表しています。MPT チャネル 0 (IGBT0)を使用し、2 相 PPG 波形が MTOUT00 と MTOUT10 から生成されます。

ピン配置:

機能	端子名	端子番号	入出力ポート
PPG output 0	MT0OUT0	34	PG3
PPG output 1	MT0OUT1	33	PG2
EMG input	GEMG0	32	PG1

動作シーケンス:

1　GEMG0 端子と DVDD を接続し、プルアップします。
2　MT0OUT0 端子と MT0OUT1 端子をオシロスコープに接続します。
3　SW7 を OFF します。
4　電源を投入し、LED0 が点灯することを確認します。その後、MT0OUT0 端子や MT0OUT1 端子から何も波形が出力されていないことを確認します。
5　SW7 を ON します。
6　IGBT タイマが実行を開始すると同時に PPG 波形が MT0OUT0 と MT0OUT1 から観察できます。2 相の PPG 波形サイクルは、50us でデューティ 40 %です。(high レベル：20us)。MT0OUT1 は MT0OUT0 より 25us 遅れます。
7　SW7 を OFF すると、PPG 出力を停止します。
8　GEMG0 を GND に接続すると、PPG 出力は停止し、LED1 が点灯します。
9　GEMG0 を DVDD と接続すると、LED1 を OFF して EMG 状態をキャンセルします。その後、手順 5 を行うと波形が表示されます。

補足:
LED0 は PB0 と接続します。
LED1 は PB1 と接続します。

6-12　LVD

このサンプルプログラムは、LVDによって電圧状態を検出します。
電圧が検出電圧より低い場合は UART に“LOWER”を出力します。
電圧が検出電圧より高い場合は UART に“UPPER”を出力します。

6-13　MLA

6-13-1　モンゴメリ乗算モード

モンゴメリ乗算の演算を行うサンプルプログラムです。
MLA 回路はアルゴリズムをモンゴメリ乗算モードに設定します。演算入力データを設定し、自動処理にて演算結果を取得します。また、このモードでは除数とモンゴメリパラメータを設定する必要があります。
3 つの演算入力データ“OriginaAData, OriginaBData”、“TrueAData, TrueBData”、“FalseAData, FalseBData”をそれぞれ “OriginaData”、“TrueData”、“FalseData”に設定します。同じ演算入力データからは同じ演算結果が得られ、異なる演算入力データからは異なる演算結果が得られます。このため、正しい結果を “OriginaData” = “TrueData” ≠ “FalseData”とします。

6-13-2　多倍長加算モード

多倍長加算の演算を行うサンプルプログラムです。
MLA 回路はアルゴリズムを多倍長加算モードに設定します。演算入力データを設定し、自動処理にて演算結果を取得します。
3 つの演算入力データ“OriginaAData, OriginaBData”、“TrueAData, TrueBData”、“FalseAData, FalseBData”をそれぞれ“OriginaData”、“TrueData”、“FalseData”に設定します。同じ演算入力データからは同じ演算結果が得られ、異なる演算入力データからは異なる演算結果が得られます。このため、正しい結果を “OriginaData” = “TrueData” ≠ “FalseData”とします。

6-13-3　多倍長減算モード

多倍長減算の演算を行うサンプルプログラムです。
MLA 回路はアルゴリズムを多倍長減算モードに設定します。演算入力データを設定し、自動処理にて演算結果を取得します。
3 つの演算入力データ“OriginaAData, OriginaBData”、“TrueAData, TrueBData”、“FalseAData, FalseBData”をそれぞれ“OriginaData”、“TrueData”、“FalseData”に設定します。同じ演算入力データからは同じ演算結果が得られ、異なる演算入力データからは異なる演算結果が得られます。このため、正しい結果を “OriginaData” = “TrueData” ≠ “FalseData”とします。

6-14　RTC

内蔵 RTC を利用し、日付・時計を UART 出力します。
初期設定の日付: **2010/10/22 12:50:55** (24 時間表示)
更新間隔: 1 秒

UART 出力:

```
2010/10/22
12:50:55
```

6-15 SHA

メッセージデータから 256 ビットのハッシュ値を演算するサンプルプログラムです。
3 つの 1024 ビットメッセージ“OriginalMessage”、“TrueMessage”、“FalseMessage”をそれぞれハッシュ値 “OHashValue”、“THashValue”、“FHashValue”に設定します。同じメッセージからは同じハッシュ値が得られ、異なるメッセージからは異なるハッシュ値が得られます。このため正しい結果を“OHashValue” = “THashValue” ≠ “FHashValue”とします。
演算は 512 ビット単位で行われるため、1024 ビットメッセージは 2 ユニットに分割されます。メッセージブロックは、CPU によって転送します。このため、SHA コアは SHASTA<START>に"1"を設定することで演算を開始します。

6-16 SSP

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の場合、LED2 と LED3 を点灯します。同一でない場合、LED0 と LED1を点灯します。受信データは UART 出力にて確認可能です。
UART 出力:

```
SSP RX DATA:
xxxxxx
```

補足: LED0~LED3 は PB0~PB3 と接続します。

6-17 TMRB

6-17-1 汎用タイマ

本サンプルは、MCU のタイマを使って、汎用タイマを実現します。
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。

6-17-2 PPG 出力

SW0 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、波形は UART 出力にて確認できます。
電源投入すると PPG モードに入り、PPG 出力を開始します。
SW0 を押すことでデューティを変更します。
10% → 25% → 50% → 75% → 90% → 10%

補足:
LED0~LED3 は PB0~PB3 と接続します。
SW4 は PJ0 と接続します。

6-18 SIO/UART

6-18-1 UART

この例は、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 UART 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。
補足:
この UART チャネルは本例では UART0 を使用します。
SW_PORT は SW0, SW0 で、PJ0 と接続します。

6-18-2 UART FIFO

この例は、UART0 から"TMPM46B1"というデータを FIFO を使用した UART3 へ送信します。また同時に UART0 は"TMPM46B2"というデータを FIFO を使用した UART3 から受信します。
ResetIdx()関数の前にブレークポイントを設定してください。RxBuffer=”TMPM46B2”となり、RxBuffer1="TMPM46B1"となると設定したブレークポイントで停止します。
補足:
UART0 の TX(pin15)と UART3 の RX(pin87)を接続し、UART0 の RX(pin14)と UART3 の TX(pin88)を接続します。

6-18-3 SIO

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

6-19 µDMAC

本サンプルプログラムでは、各 uDMAC ユニットのデータ制御用に 1K の RAM 領域をどのように確保するかを示します。また本サンプルプログラムでは、RAM 領域の"src"から"dst"へデータを転送するために DMA のソフトウェアトリガを使用します。

動作シーケンス:

1. uDMAC ユニット A を次のように初期化。転送タイプをバーストタイプ、チャネルをイネーブル、マスク設定のチャネルをイネーブル、初期データの使用、優先度通常を使用します。
2. 確保した 1K の RAM 領域先頭に初期ベースアドレスを設定します。
3. ソフトウェアトリガ用設定データを指定、制御データ領域に設定データを書き込みます。
4. 全設定完了後、DMA ユニット A を許可します。
5. DMAC_BASIC モードが選択されている場合、転送が完了するまで再度トリガをかけ続けます。DMAC_AUTOMATIC モードが選択されている場合、転送が完了する前に一度トリガをかけます。
6. 転送完了を確認した後、送信元と送信先のデータを比較する。

6-20 WDT

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

動作シーケンス:

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。

補足:
LED0 は PB0 と接続します。
LED1 は PB1 と接続します。

7 ソフトウェア

本ソフトウェアは、TMPM46B MCU の主要機能を評価ボード上で動作確認するためのサンプルプログラムです。

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

ワークスペース構造とプロジェクト名は、以下の通りです:

IAR EWARM:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm46b_wdt_int.c
        │   tmpm46b_wdt_int.h
        │
        ├─TX04_CMSIS
        │ │ system_TMPM46B.c
        │ │ system_TMPM46B.h
        │ │ TMPM46B.h
        │ │
        │ └─startup
        │       startup_TMPM46B.s
        │
        ├─TX04_Periph_Driver
        │   ├─inc
        │   │   tmpm46b_gpio.h
        │   │   tmpm46b_wdt.h
        │   │   TX04_common.h
        │   │
        │   └─src
        │       tmpm46b_gpio.c
        │       tmpm46b_wdt.c
        │
        └─TMPM46B-EVAL
                led.c
                led.h
                sw.c
                sw.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm46b_wdt_int.c
        │   tmpm46b_wdt_int.h
        │
        ├─TX04_CMSIS
```

```
│    system_TMPM46B.c
│    startup_TMPM46B.s
│
├─TX04_Periph_Driver
│    tmpm46b_gpio.c
│    tmpm46b_wdt.c
│
└─TMPM46B-EVAL
      led.c
      sw.c
```

7-1 ADC

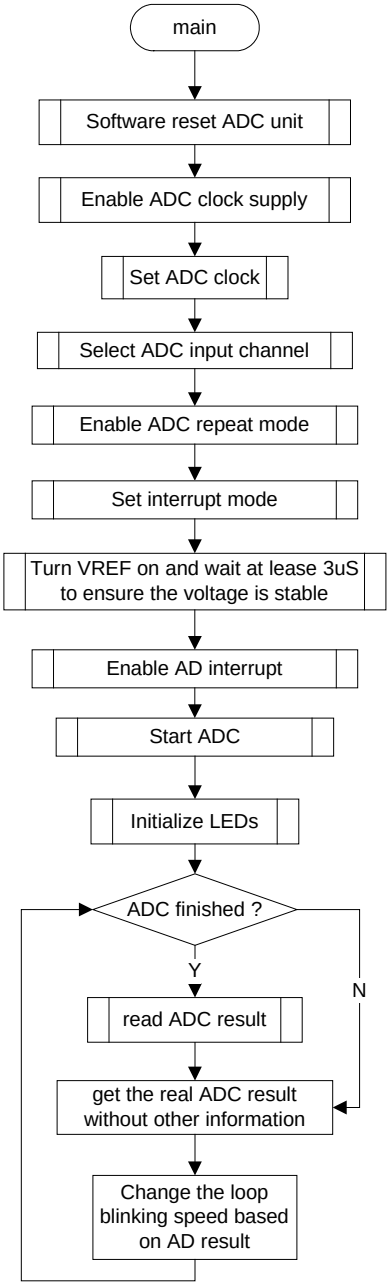
7-1-1 例: ADC データリード

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

このでは以下を行います。

1. ADC 設定と初期化を行います。
2. AD 変換の開始と AD 変換結果の読み出しを行います。
3. ループ点滅スピードの調整に AD 変換結果を使用します。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に AD コンバータユニットをリセットし、クロック供給を許可します。AD コンバータクロックを選択します。チャネルを選択し、リピートモードをイネーブルにします。割り込みモードを設定します。

```
/* Software reset ADC */
ADC_SWReset(TSB_AD);

/* Enable ADC clock supply */
CG_SetADCClkSupply(ENABLE);

/* Set ADC clock */
ADC_SetClk(TSB_AD,    ADC_CONVERSION_CLK_80,
                  ADC_FC_DIVIDE_LEVEL_8);
```

```
/* Select ADC input channel : Channel 0 */
ADC_SetInputChannel(TSB_AD, ADC_AN_00);

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

/* Set interrupt mode */
ADC_SetINTMode(TSB_AD, ADC_INT_CONVERSION_8);
```

VREF を ON し、電圧が安定するまで 3us 待ちます。

```
ADC_SetVref(TSB_AD, ENABLE);
cnt = 100U;
while(cnt){
    cnt--;
}
```

AD 割り込みを許可し、ADC 動作を許可します。

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

/* Start ADC */
ADC_Start(TSB_AD);
```

AD 変換を開始した後、AD 変換終了割り込みフラグを待ちます。その後 AD 変換結果を読み出し、LED ループ点滅スピードの調整に使用します。

```
while (1U) {

    if (fIntADC == 1U) {
        fIntADC = 0U;
        /* Read ADC result when it is finished */
        adResult = ADC_GetConvertResult(TSB_AD, ADC_REG00);

        /* Get the real ADC result without other information */
        /* "/256" is to limit the range of AD value */
        timeUp = 16U - adResult.Bit.ADResult / 256U;

    } else {
        /* Do nothing */
    }

    /* use 'timeUp' above to adjust the loop blinking speed */
    if (softT.flag_TimeUp) {
        softT.flag_TimeUp = 0U;
        cnt++;
        if (cnt >= timeUp) {
            cnt = 0U;
            idx++;
            if (idx == 1U) {
                LED_On(LED0);
            } else if (idx == 2U) {
                LED_On(LED1);
                LED_Off(LED0);
```

```
                } else if (idx == 3U) {
                    LED_On(LED2);
                     LED_Off(LED1);
                } else if (idx == 4U) {
                     LED_On(LED3);
                     LED_Off(LED2);
                } else if (idx == 5U) {
                     idx = 0U;
                     LED_Off(LED3);
                } else {
                     /* Do nothing */
                }
            } else {
                     /* Do nothing */
            }
        } else {
            /* Do nothing */
        }
    }
```

7-2 AES

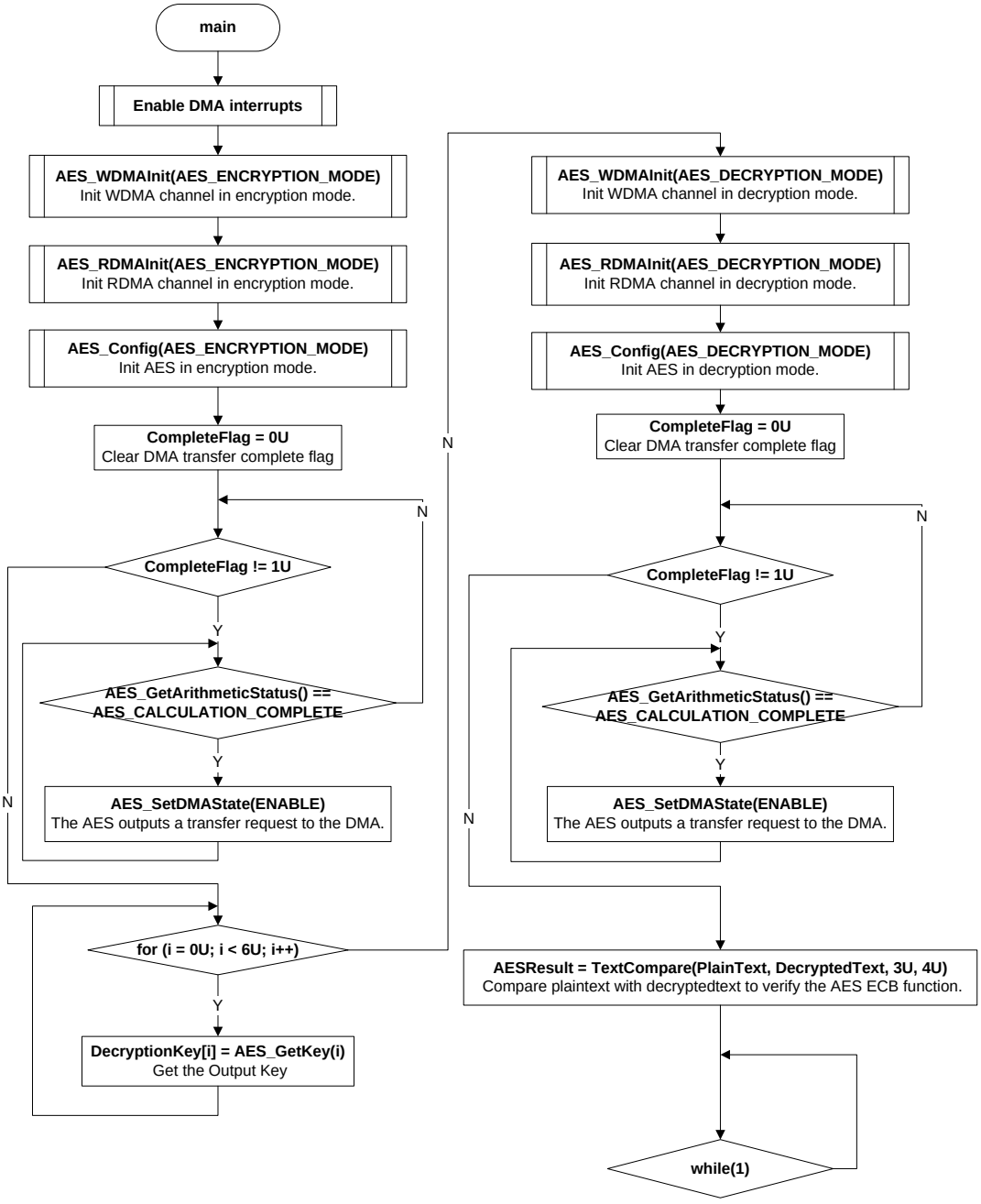
7-2-1 例: AES ECB モード

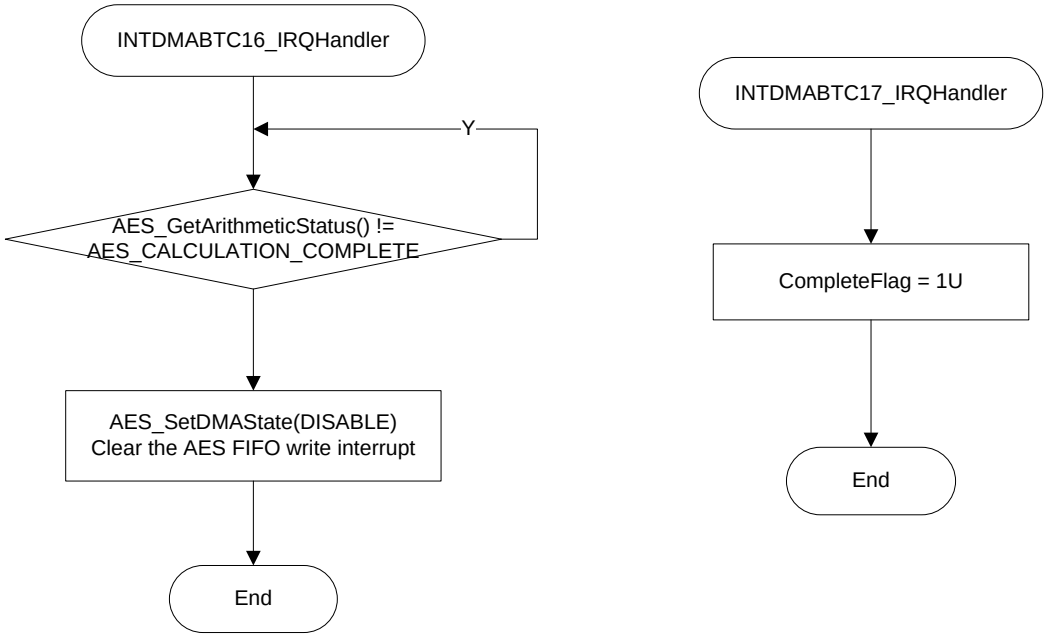
ペリフェラルドライバ(AES, uDMAC, UART)を使用したサンプルプログラムです。

このサンプルプログラムでは以下を行います。

1. uDMAC の設定と ECB モードの暗号化設定を行います。
2. AES コアは DMA 転送により平文を暗号化します。
3. uDMAC の設定と ECB モードの復号化設定を行います。
4. AES コアは DMA 転送により暗号化データを復号します。
5. ECB 処理を確認するため、復号データと暗号化前のデータを比較します。

- **フローチャート**


main
Enable DMA interrupts
AES_WDMAInit(AES_ENCRYPTION_MODE)
Init WDMA channel in encryption mode.
AES_RDMAInit(AES_ENCRYPTION_MODE)
Init RDMA channel in encryption mode.
AES_Config(AES_ENCRYPTION_MODE)
Init AES in encryption mode.
CompleteFlag = 0U
Clear DMA transfer complete flag
CompleteFlag != 1U
Y
N
AES_GetArithmeticStatus() == AES_CALCULATION_COMPLETE
Y
N
AES_SetDMAState(ENABLE)
The AES outputs a transfer request to the DMA.
for (i = 0U; i < 6U; i++)
Y
N
DecryptionKey[i] = AES_GetKey(i)
Get the Output Key
AES_WDMAInit(AES_DECRYPTION_MODE)
Init WDMA channel in decryption mode.
AES_RDMAInit(AES_DECRYPTION_MODE)
Init RDMA channel in decryption mode.
AES_Config(AES_DECRYPTION_MODE)
Init AES in decryption mode.
CompleteFlag = 0U
Clear DMA transfer complete flag
CompleteFlag != 1U
Y
N
AES_GetArithmeticStatus() == AES_CALCULATION_COMPLETE
Y
N
AES_SetDMAState(ENABLE)
The AES outputs a transfer request to the DMA.
AESResult = TextCompare(PlainText, DecryptedText, 3U, 4U)
Compare plaintext with decryptedtext to verify the AES ECB function.
while(1)



- **サンプルプログラムのコードと説明**

DMA 割り込みを設定します。

```
/* enable DMA interrupts */
NVIC_ClearPendingIRQ(INTDMABTC16_IRQn);
NVIC_EnableIRQ(INTDMABTC16_IRQn);
NVIC_ClearPendingIRQ(INTDMABTC15_IRQn);
NVIC_EnableIRQ(INTDMABTC15_IRQn);
NVIC_ClearPendingIRQ(INTDMABTC17_IRQn);
NVIC_EnableIRQ(INTDMABTC17_IRQn);
__enable_irq();
```

AES 回路へデータ書き込みを行うための WDMA チャネルの設定を行います。また AES 回路からのデータ読み出しを行うための RDAM チャネルの設定を行います。AES コアを ECB モードにし、暗号化の動作設定、192 ビットの鍵長、鍵データの設定を行います。

```
/* Initialze WDMA, WDMA will be triggered by DMA transfer interrupt. */
AES_WDMAInit(AES_ENCRYPTION_MODE);
/* Initialzie RDMA, RDMA will be triggered by AES calculation complete. */
AES_RDMAInit(AES_ENCRYPTION_MODE);
/* Initialize AES to encryption mode */
AES_Config(AES_ENCRYPTION_MODE);
```

AES コアは 128 ビット単位で平文を暗号化します。

```
CompleteFlag = 0U;
/* Encrypt 3 units plaintext. */
while (CompleteFlag != 1U) {
    while(AES_GetArithmeticStatus() == AES_CALCULATION_COMPLETE) {
        AES_SetDMAState(ENABLE);
    }
}
```

鍵データを読み出し、出力された鍵を復号化時の復号鍵として使用します。

```
for (i = 0U; i < 6U; i++) {
```

```
        DecryptionKey[i] = AES_GetKey(i);
    }
```

AES コアにデータを書き込むための WDMA の設定を行います。AES コアからデータを読み出すための RDMA の設定を行います。AES コアを ECB モードにし、復号化の動作設定、192 ビットの鍵長、鍵データの設定を行います。

```
    /* Initialze WDMA, WDMA will be triggered by DMA transfer interrupt. */
    AES_WDMAInit(AES_DECRYPTION_MODE);
    /* Initialzie RDMA, RDMA will be triggered by AES calculation complete. */
    AES_RDMAInit(AES_DECRYPTION_MODE);
    /* Initialize AES to encryption mode */
    AES_Config(AES_DECRYPTION_MODE);
```

AES コアは暗号化されたデータを 128 ビット単位で復号化します。

```
    CompleteFlag = 0U;
    /* Decrypt the 3 units encryted text. */
    while (CompleteFlag != 1U) {
        while(AES_GetArithmeticStatus() == AES_CALCULATION_COMPLETE )
{
            AES_SetDMAState(ENABLE);
        }
    }
```

ECB 処理を確認するため、復号データと暗号化前のデータを比較します。AESResult が SUCCESS の場合は比較結果に問題がないと判断します。

```
    /* Compare plaintext with decrypted text to verify the AES ECB function. */
    AESResult = TextCompare(PlainText, DecryptedText, 3U, 4U);
    if (AESResult == SUCCESS) {
        common_uart_disp("AES ECB mode processed successfully !\n");
    } else {
        common_uart_disp("AES ECB mode processed with error !\n");
    }
```

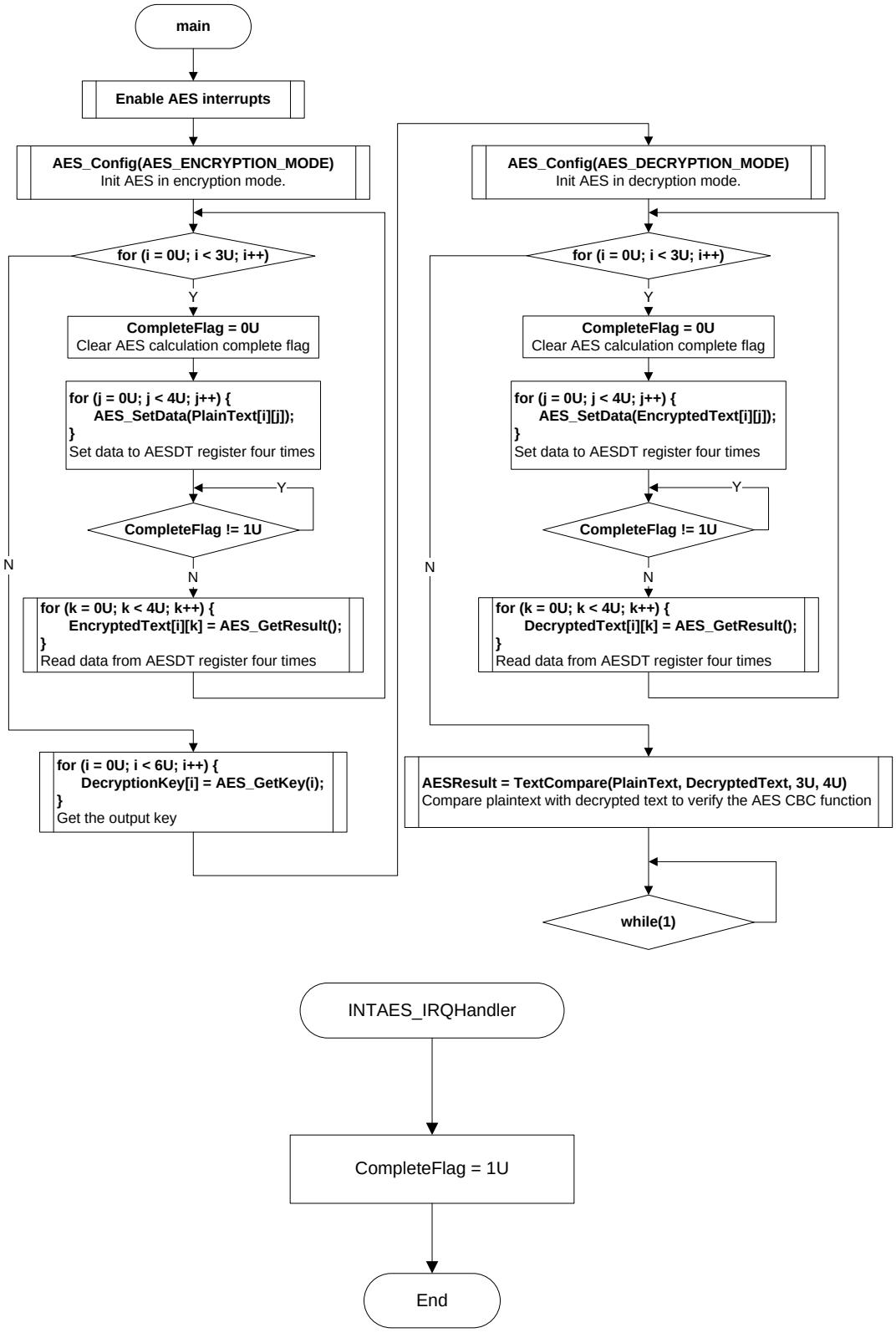
7-2-2 例: AES CBC モード

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

このサンプルプログラムでは以下を行います。

1. uDMAC の設定と CBC モードの暗号化設定を行います。
2. AES コアは CPU により平文を暗号化します。
3. ECB モードの復号化設定を行います。
4. AES コアは CPU により暗号化データを復号します。
5. CBC 処理を確認するため、復号データと暗号化前のデータを比較します。

- **フローチャート**



- **サンプルプログラムのコードと説明**

AES 割り込みの設定を行います。

```
/* enable AES calculation completion interrupt */
NVIC_ClearPendingIRQ(INTAES_IRQn);
```

```
        NVIC_EnableIRQ(INTAES_IRQn);
        __enable_irq();
```

AESコアをCBCモードにし、暗号化の動作設定、192ビットの鍵長、鍵データの設定を行います。

```
        /* Initialize AES to encryption mode */
        AES_Config(AES_ENCRYPTION_MODE);
```

AESコアは平文を128ビット単位で暗号化を行います。

```
    /* Encrypt 3 units plaintext. */
    for (i = 0U; i < 3U; i++) {
            CompleteFlag = 0U;
        /* If 128-bit data is stored in the AESDT register, calculation automatically
starts */
        for (j = 0U; j < 4U; j++) {
            AES_SetData(PlainText[i][j]);
        }
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /*The CPU reads the value from AESODT<ODT[31:0]>(four times) */
        for (k = 0U; k < 4U; k++) {
            EncryptedText[i][k] = AES_GetResult();
        }
    }
```

鍵データを読み出し、出力された鍵を復号化時の復号鍵として使用します。

```
    for (i = 0U; i < 6U; i++) {
        DecryptionKey[i] = AES_GetKey(i);
    }
```

AESコアをCBCモードにし、復号化の動作設定、192ビットの鍵長、鍵データの設定を行います。

```
    /* Initialize AES to encryption mode */
    AES_Config(AES_DECRYPTION_MODE);
```

AESコアは暗号化されたデータを128ビット単位で復号化します。

```
    /* Decrypt the 3 units encryted text. */
    for (i = 0U; i < 3U; i++) {
        CompleteFlag = 0U;
        /* If 128-bit data is stored in the AESDT register, calculation automatically
starts */
        for (j = 0U; j < 4U; j++) {
            AES_SetData(EncryptedText[i][j]);
        }
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /*The CPU reads the value from AESODT<ODT[31:0]>(four times) */
        for (k = 0U; k < 4U; k++) {
            DecryptedText[i][k] = AES_GetResult();
```

```
        }
    }
```

CBC 処理を確認するため、復号データと暗号化前のデータを比較します。AESResult が SUCCESS の場合は比較結果に問題がないと判断します。

```
    /* Compare plaintext with decrypted text to verify the AES CBC function. */
    AESResult = TextCompare(PlainText, DecryptedText, 3U, 4U);
    if (AESResult == SUCCESS) {
        common_uart_disp("AES CBC mode processed successfully !\n");
    } else {
        common_uart_disp("AES CBC mode processed with error !\n");
    }
```

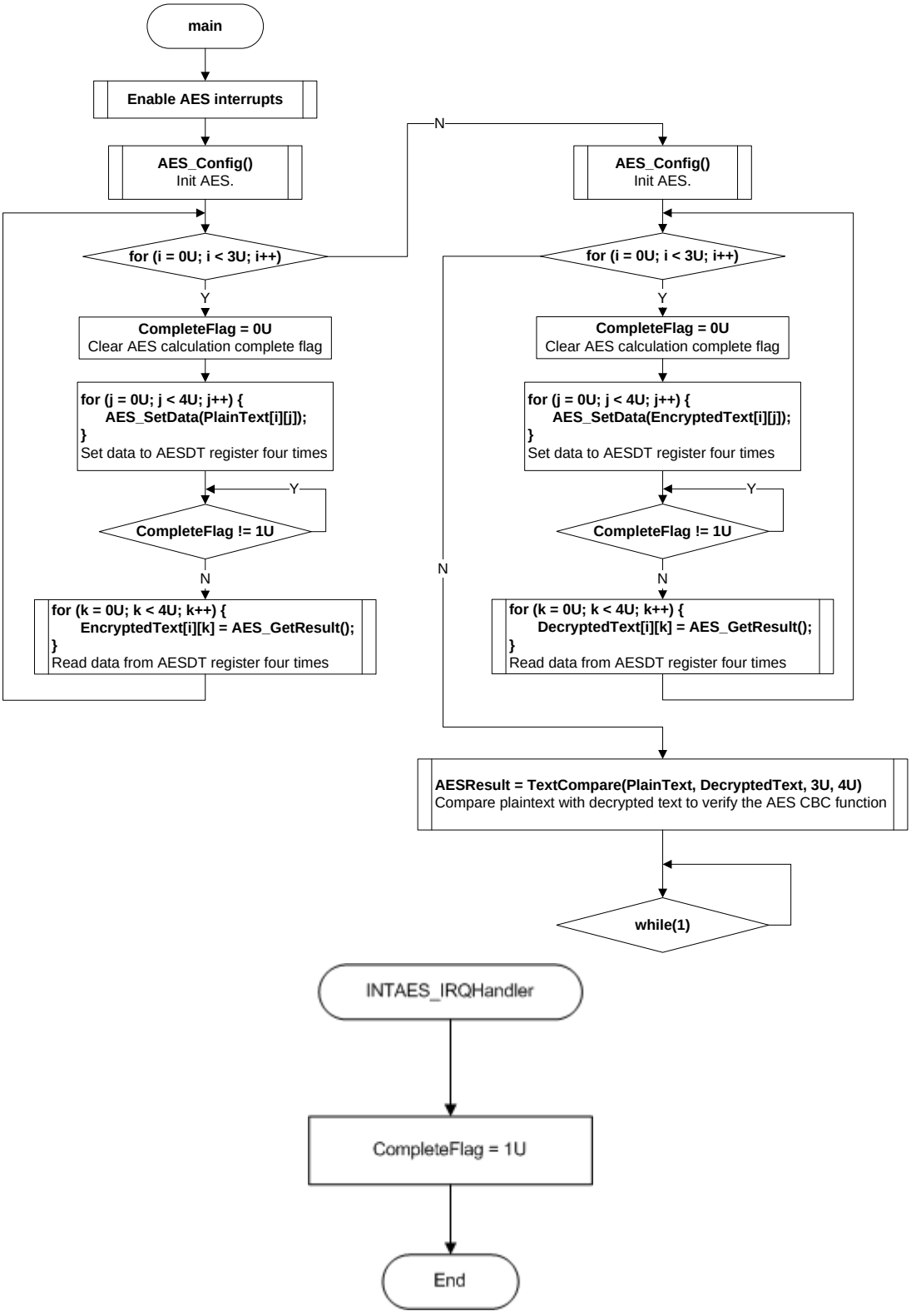
7-2-3 例: AES CTR モード

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

このサンプルプログラムでは以下を行います。

1. AES コアを CTR モードに設定します。
2. AES コアは CPU により平文を暗号化します。
3. AES コアを CTR モードに設定します。
4. AES コアは CPU により暗号化データを復号します。
5. CTR 処理を確認するため、復号データと暗号化前のデータを比較します。

- **フローチャート**



- **サンプルプログラムのコードと説明**

AES 割り込みの設定を行います。

```
/* enable AES calculation completion interrupt */
NVIC_ClearPendingIRQ(INTAES_IRQn);
NVIC_EnableIRQ(INTAES_IRQn);
```

```
        __enable_irq();
```

AESコアをCTRモードにし、192ビットの鍵長、鍵データの設定を行います。カウント値の入力を行い、AESOD<OP>を"0"クリアします。

```
        /* Initialize AES to encryption mode */
        AES_Config();
```

AES コアは平文を 128 ビット単位で暗号化します。

```
    /* Encrypt 3 units plaintext. */
    for (i = 0U; i < 3U; i++) {
        CompleteFlag = 0U;
        /* If 128-bit data is stored in the AESDT register, calculation automatically
starts */
        for (j = 0U; j < 4U; j++) {
            AES_SetData(PlainText[i][j]);
        }
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /*The CPU reads the value from AESODT<ODT[31:0]>(four times) */
        for (k = 0U; k < 4U; k++) {
            EncryptedText[i][k] = AES_GetResult();
        }
    }
```

AESコアをCTRモードにし、192ビットの鍵長、鍵データの設定を行います。カウント値の入力を行い、AESOD<OP>を"0"クリアします。

```
    /* Initialize AES to encryption mode */
    AES_Config();
```

AES コアは 128 ビット単位で暗号化されたデータを復号化します。

```
    /* Decrypt the 3 units encryted text. */
    for (i = 0U; i < 3U; i++) {
        CompleteFlag = 0U;
        /* If 128-bit data is stored in the AESDT register, calculation automatically
starts */
        for (j = 0U; j < 4U; j++) {
            AES_SetData(EncryptedText[i][j]);
        }
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /*The CPU reads the value from AESODT<ODT[31:0]>(four times) */
        for (k = 0U; k < 4U; k++) {
            DecryptedText[i][k] = AES_GetResult();
        }
    }
```

CTR 処理を確認するため、復号データと暗号化前のデータを比較します。AESResult が SUCCESS の場合は比較結果に問題がないと判断します。

```
    /* Compare plaintext with decryptedtext to verify the AES CTR function. */
```

```
AESResult = TextCompare(PlainText, DecryptedText, 3U, 4U);
if (AESResult == SUCCESS) {
      common_uart_disp("AES CTR mode processed successfully !\n");
} else {
      common_uart_disp("AES CTR mode processed with error !\n");
}
```

7-3 CG

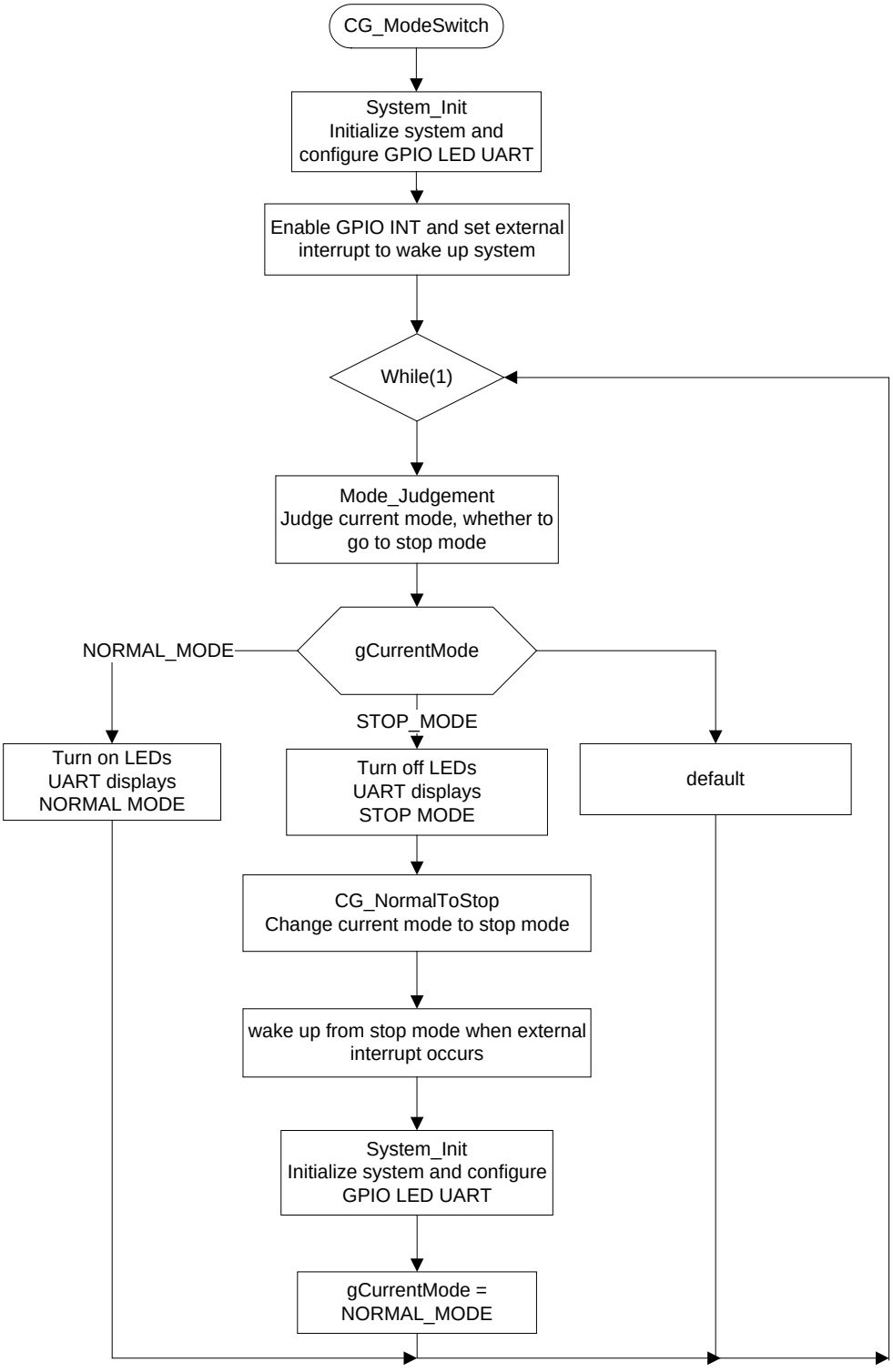
7-3-1 例: NORMAL <-> STOP1 モード変更

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

この例では以下を行います。

1. 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**

(リセット後)CG の通常設定:

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

```
    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);
    /* Set ΦT0 = fc/4 */
    CG_SetPhiT0Level(CG_DIVIDE_4);
    /* Set low power consumption mode stop1 */
    CG_SetSTBYMode(CG_STBY_MODE_STOP1);
```

低消費電力モードから復帰するための外部割込みの設定:

INTF を設定します。割り込み保留要求をクリアし、INTF を許可します。

```
    __disable_irq();
    CG_ClearINTReq(CG_INT_SRC_F);
    /* Set external interrupt to wake up system */
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_F,
CG_INT_ACTIVE_STATE_FALLING, ENABLE);
    NVIC_ClearPendingIRQ(INTF_IRQn);
    NVIC_EnableIRQ(INTF_IRQn);
    __enable_irq();
```

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 STOP モードに入ります。

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

マルチクロック回路の許可:

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

```
    WorkState st = BUSY;
    CG_SetPLL(DISABLE);
    CG_SetFPLLValue(CG_16M_MUL_6_FPLL);
    retval = CG_SetPLL(ENABLE);
    if (retval == SUCCESS) {
        /* Set warm up time */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
                                CG_WUODR_PLL);
        CG_StartWarmUp();
        do {
            st = CG_GetWarmUpState();
        } while (st != DONE);

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

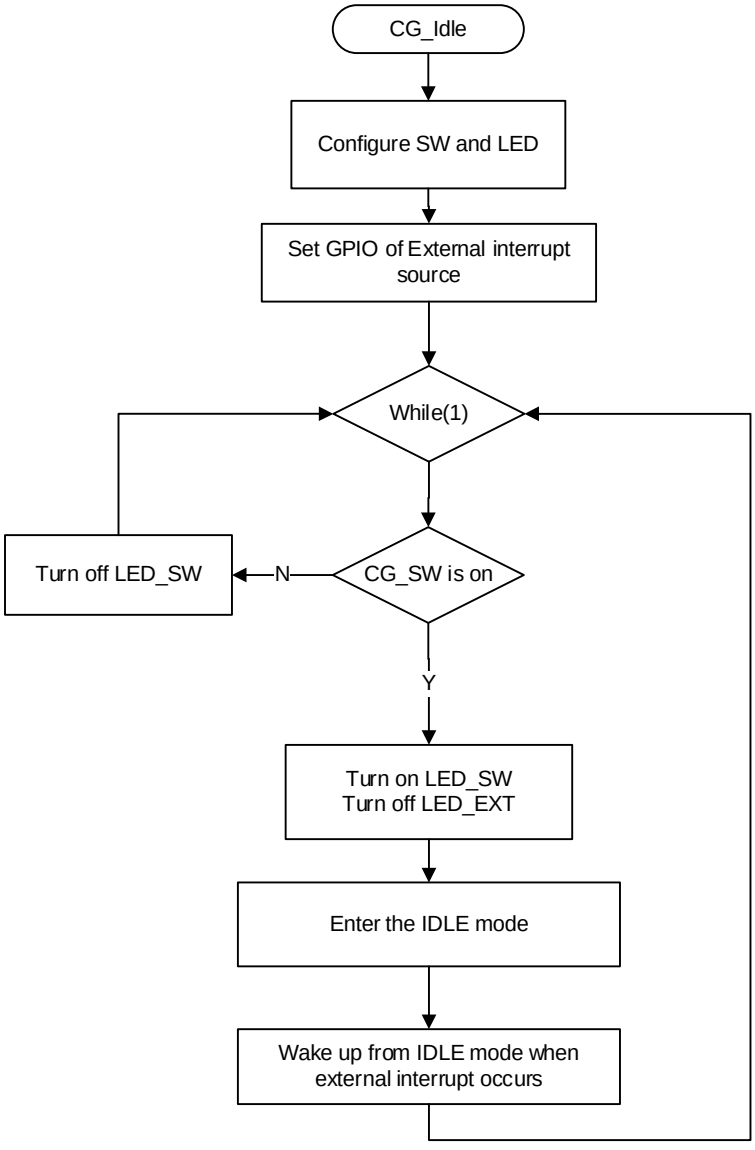
7-3-2 例: NORMAL <-> IDLE モード変更

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 IDLE 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**

低消費電力モードから復帰するための外部割込みの設定:
INTF を設定します。割り込み保留要求をクリアし、INTF を許可します。

```
    GPIO_ExtIntSrc();
    CG_ClearINTReq(CG_ExtINTSrc);
    NVIC_ClearPendingIRQ(ExtINTSrc_IRQn);
    NVIC_EnableIRQ(ExtINTSrc_IRQn);
```

IDLE モード設定:
IDLE モードに入る設定を行います。IDLE モードの解除を行うための解除割り込み(INTF)を設定し、__WFI() 命令を使用し IDLE モードに入ります。

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

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

    __DSB();
    __WFI();

    /* INT release */
    CG_SetSTBYReleaseINTSrc(CG_ExtINTSrc,
CG_INT_ACTIVE_STATE_RISING, DISABLE);
```

NORMAL モードと IDLE モードの切り替え処理
LED_EXT を点灯し、while()ループにて CG_SW の状態を取得します。
CG_SW が OFF であれば LED_SW を点灯し、ON であれば LED_SW を消灯し、IDLE モードへ移行します。

```
    LED_On(LED_EXT);
    while (1U) {

        if (SW_Get(CG_SW) == 0U) {
            LED_On(LED_SW);

            /* LED indicator is off before enter IDLE */
            LED_Off(LED_EXT);

            enter_IDLE();

        } else {
            LED_Off(LED_SW);
        }
```

7-3-3 例: NORMAL <-> STOP2 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 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、低消費モード解除用端子の設定を行います。

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

```
    GPIO_ExtIntSrc();
```

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。

```
    /* after release standby mode */
    if (is_StanbyRelease()) {
        LED_On(LED2);
        CG_SetSTBYReleaseINTSrc(CG_ExtINTSrc,
                CG_INT_ACTIVE_STATE_RISING, DISABLE);

        NVIC_EnableIRQ(ExtINTSrc_IRQn);

        CG_SetPortKeepInStop2Mode(DISABLE);
```

リセット要因がSTOP2モードの解除でなければ、低消費電力モード解除用割り込みの設定を行います。

```
    CG_ClearINTReq(CG_ExtINTSrc);
    NVIC_ClearPendingIRQ(ExtINTSrc_IRQn);
    NVIC_EnableIRQ(ExtINTSrc_IRQn);
```

LEDを点灯し、while()ループにてCG_SWの状態を取得します。
CG_SWがOFFであればLED_SWを点灯し、ONであればLEDを消灯し、STOP2モードへ移行します。

```
    LED_On(LED_EXT);
    while (1U) {
        if (SW_Get(CG_SW) == 1) {

            LED_Off(LED_ALL);    /* LED   is off before enter stop2 */
            enter_STOP2();

        } else {
            LED_On(LED_SW);
        }
    }
}
```

STOP2モードの解除要因として外部割り込みを許可します。

```
    CG_SetSTBYReleaseINTSrc(CG_ExtINTSrc,
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。

```
void config_STOP2(void)
{
    volatile WorkState st = BUSY;
    volatile uint32_t wuef = 0U;

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

    TSB_CG->PLLSEL &= PLLON_CLEAR;
    TSB_CG->PLLSEL &= PLLSEL_CLEAR;
    while (CG_GetFcSrc() != CG_FC_SRC_FOSC) {
```

```
    };                                  /* Confirm */

    /* When IHOSC is disable, enable IHOSC */
    if (CG_GetFoscState(CG_FOSC_OSC_INT) == DISABLE) {
        /* Enable IHOSC */
        CG_SetFosc(CG_FOSC_OSC_INT, ENABLE);

        /* Wait until IHOSC become stable */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
OSCCR_WUPT_EXT);
        CG_StartWarmUp();
        wuef = TSB_CG->OSCCR & 0x00008000U;
        while (wuef) {              /* Warm-up */
            wuef = TSB_CG->OSCCR & 0x00008000U;
        }

        /* Set IHOSC as fosc */
        CG_SetFoscSrc(CG_FOSC_OSC_INT);

        /* Wait until fosc become IHOSC */
        while (CG_GetFoscSrc() != CG_FOSC_OSC_INT) {
        };
    }

    CG_SetPortKeepInStop2Mode(ENABLE);
}
```

最後に__WFI()命令を使用して STOP2 モードに入ります。

```
    __DSB();
    __WFI();
```

7-4 ESG

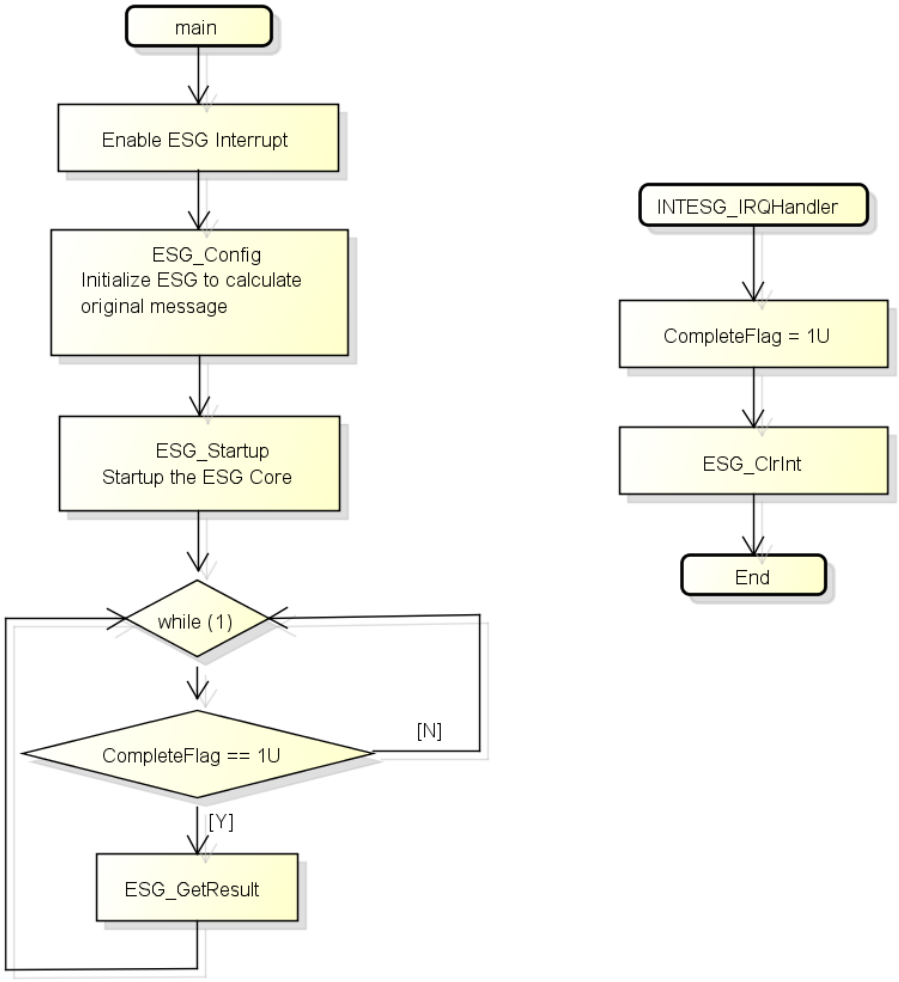
7-4-1 例: ESG

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

このサンプルプログラムでは以下を行います。

1. オリジナルメッセージの演算を行うために ESG の設定を行います。
2. 演算を開始し、演算結果を取得します。

- **フローチャート**



- **サンプルプログラムのコードと説明**

ESG 割り込みの設定を行います。

```
/* enable ESG calculation completion interrupt */
NVIC_ClearPendingIRQ(INTESG_IRQn);
NVIC_EnableIRQ(INTESG_IRQn);
__enable_irq();
```

ESG の初期設定を行います。

```
void ESG_Config(ESG_LatchTiming Value, uint16_t Fintming)
{
    /* Check the parameters */
    assert_param(IS_ESG_LATCH_TIMING(Value));

    /* Confirm the ESG core stops */
    if(ESG_GetCalculationStatus() == ESG_CALCULATION_COMPLETE) {
      /* Confirm no interrupt generation */
      if(ESG_GetIntStatus() == DISABLE) {
        /* Set the latch timing. */
        ESG_SetLatchTiming(Value);
        latchtiming = ESG_GetLatchTiming();
        if (Fintming >= 512U * (latchtiming + 1U) + 3U) {
        /* Set the output timing. */
        ESG_SetFintiming(Fintming);
```

```
            } else {
                common_uart_disp("Fintming value input error !\n");
            }
        } else {
            /* Do nothing */
        }
    } else {
        /* Do nothing */
    }
}
```

オリジナルメッセージの演算を行うために ESG の設定を行い、その後、ESG 演算を開始します。

```
ESG_Config(ESG_LATCH_TIMING_1, 560);
ESG_Startup();
```

ESG 演算が終了すると、割り込みが発生し、乱数シードを取得することができます。

```
if ((CompleteFlag == 1U)) {
        CompleteFlag = 0;
        /* Get the calculation result */
        ESG_GetResult(Data);
        for(i = 0U; i < 16U; i++) {
#ifdef DEBUG
            printf("0x%08x \r\n", Data[i]);
#endif
            sprintf((char *)buffer, "%lu", Data[i]);
            common_uart_disp(buffer);
            common_uart_disp("\n");
        }
}
```

7-5 EXB

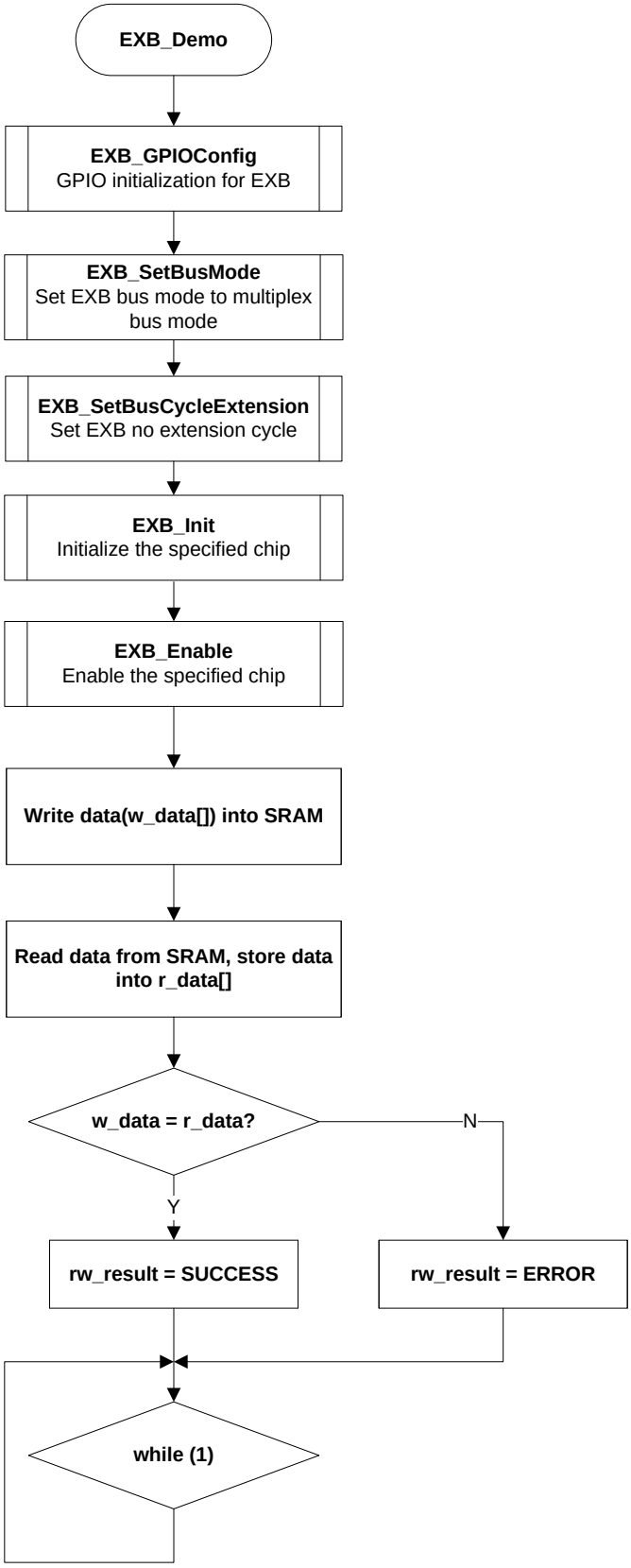
7-5-1 例: SRAM のリード/ライト

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

この例では以下を行います。

1. EXB の初期化
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 の初期設定を行います。

```
	uint8_t chip = EXB_CS1;
	uint8_t BusMode = EXB_BUS_MULTIPLEX;
	uint8_t Cycle = EXB_CYCLE_QUADRUPLE
	rw_result = SUCCESS;

#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 };

	/* Configure UART */
	hardware_init(UART_RETARGET);

	InitStruct.AddrSpaceSize = EXB_128K_BYTE;
	InitStruct.StartAddr = 0x00U;
	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_8;
	InitStruct.Cycles.ReadSetupCycle = EXB_CYCLE_2;
	InitStruct.Cycles.WriteSetupCycle = EXB_CYCLE_2;
	InitStruct.Cycles.ALEWaitCycle = EXB_CYCLE_2;
	InitStruct.Cycles.ReadRecoveryCycle = EXB_CYCLE_2;
	InitStruct.Cycles.WriteRecoveryCycle = EXB_CYCLE_2;
	InitStruct.Cycles.ChipSelectRecoveryCycle = EXB_CYCLE_2;
```

GPIO の EXB 設定を行い、外部 SRAM アクセスを行えるようにします。外部 SRAM へ w_data[]を書き込みます。その後、外部 SRAM からリードしたデータを r_data[]へ保存します。外部 SRAM へのライト/リードが成功したかを rw_result にて確認します。

```
#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;
	}

	/* write data from w_data[] to SRAM */
	for (i = 0; i < TEST_DATA_LEN ; i++) {
	   addr[i] = w_data[i];
```

```
    }

    /* read data from SRAM, store into r_data[] */
    for (i = 0; i < TEST_DATA_LEN ; i++) {
       r_data[i] = addr[i];
    }

    /* check rw_result to see if SRAM write/read is successful or not */
    for (i = 0; i < TEST_DATA_LEN ; i++) {
        if (w_data[i] != r_data[i]) {
            rw_result = ERROR;
            break;
        }
    }

    if (rw_result == SUCCESS) {
        common_uart_disp("SRAM read/write successful \n");
    } else {
        common_uart_disp("SRAM read/write failed \n");
    }

#endif
    while (1) {
       /* Do nothing */
    }
```

7-6 FLASH

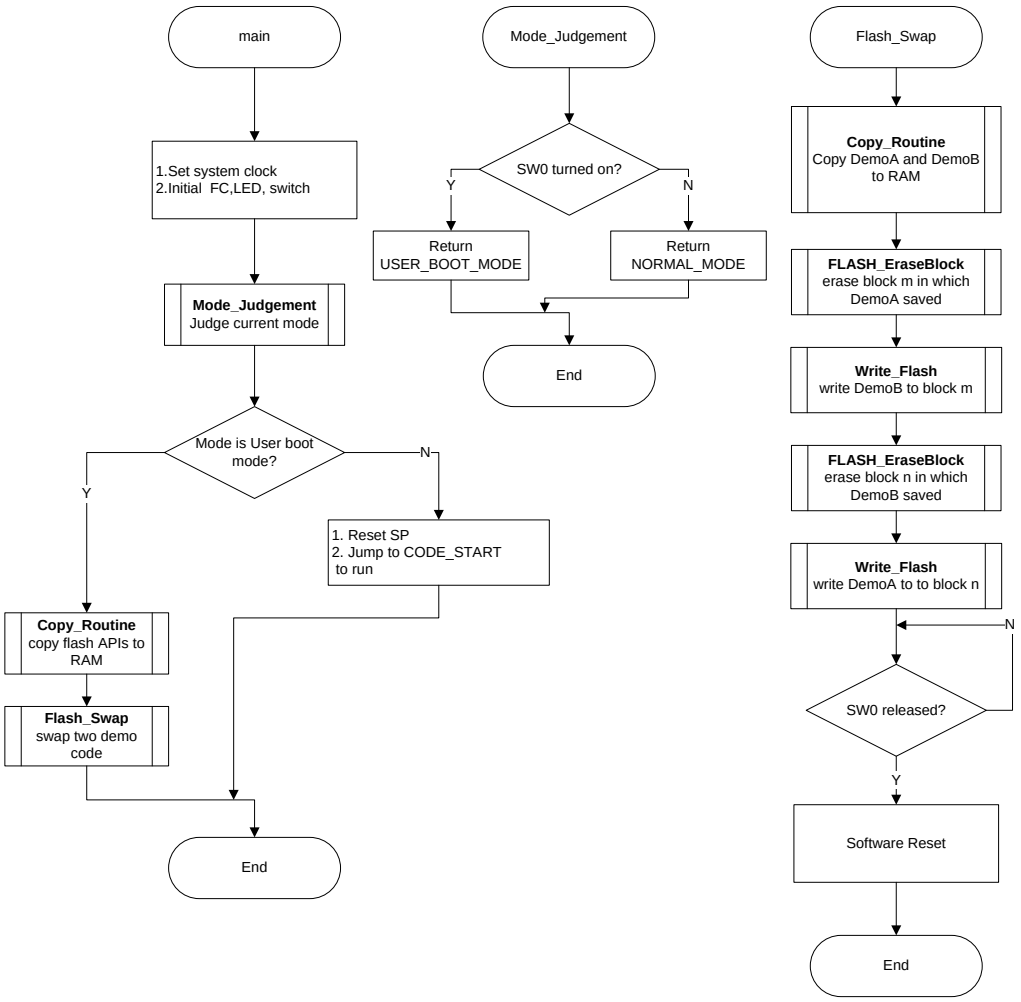
7-6-1 例: Flash_UserBoot

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

この例では以下を行います。

1. FLASH メモリのオンボードプログラミング (書き込み/消去)
2. 動作モード:シングルチップモード (ノーマルモード、ユーザーブートモード)
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 SW4 と LED を初期化します。リセット時に現在のモード判定を SW4(GPIO) で行います。

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

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

```
uint8_t Mode_Judgement(void)
{
    return (SW_Get(SW4) == 1U) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

リセット時に SW4 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(LED0 が点滅、LED2 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
          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 SW4 が押されている場合、ユーザブートモードになります。Flash メモリは自分自身で消去/書き込みを実行できないため、Flash 動作用 API を、Flash メモリ内の アドレス “FLASH_API_ROM” から RAM の “FLASH_API_RAM”にコピーします。

```
          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,まずプログラム A、プログラム 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。プログラム A とプログラム B のプログラムは Flash メモリ内にて差し替えられます。

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

```
    FC_SelectArea(FC_AREA_NONE);
```

SW4 が離されると、NVIC_SystemReset()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。プログラム A、プログラム B が差し替えられているため、アドレス “CODE_START” はプログラム B のスタートアドレスになり、プログラム B が動作します (LED1 が点滅、LED3 が点灯)。

```
    while (SW_Get(SW4) == 1U) {
    }
    /* software reset */
    NVIC_SystemReset();
```

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;
                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 < PROGRAM_UNIT; i++) {
            *PA = *source;
            source++;
        }
```

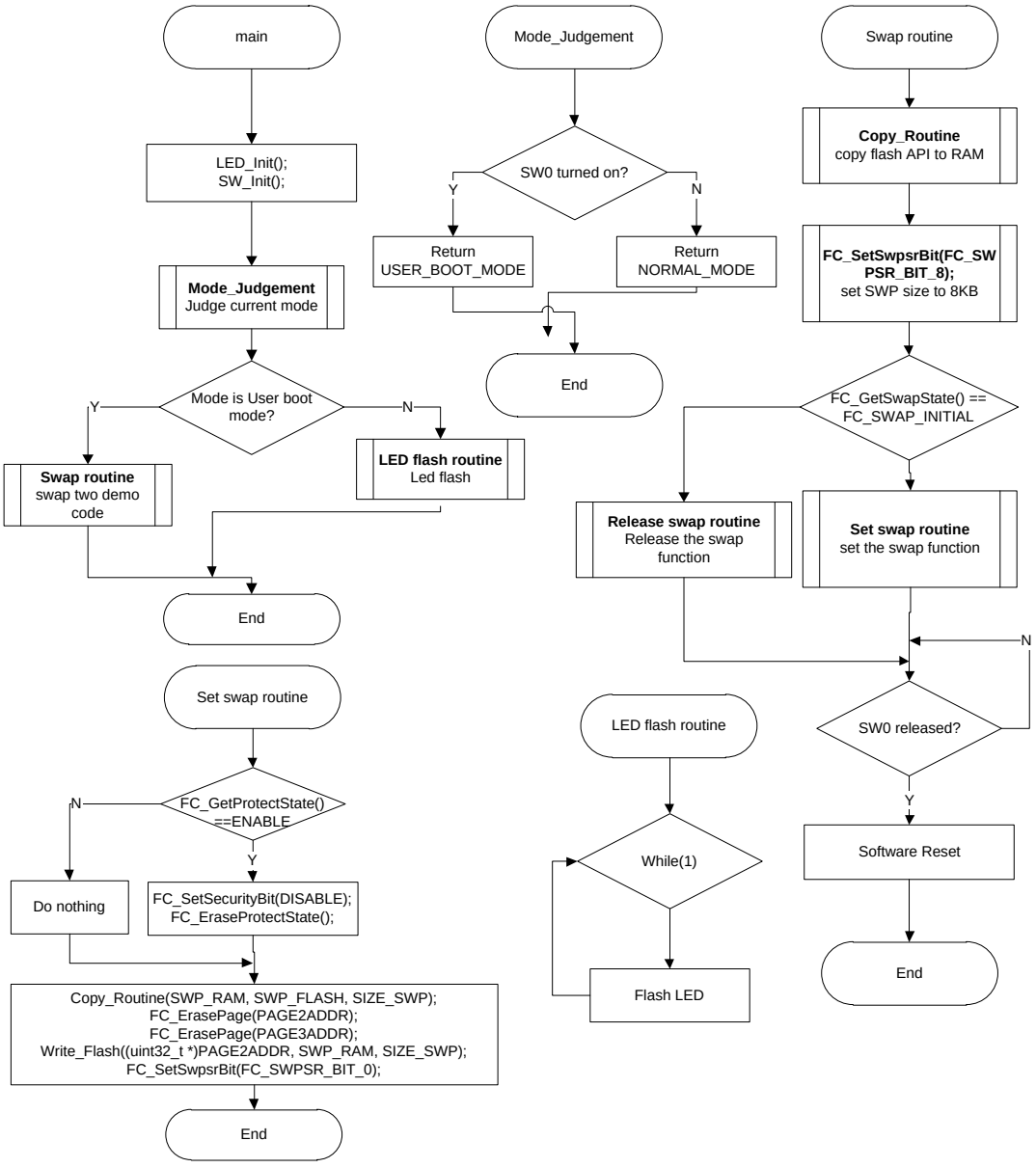
7-6-2 例: Flash_Swap

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。

この例では以下を行います。

1. Flash ROM のオンボードプログラミング (リード/ページ書き込み、メモリスワップ)

- **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 SW と LED を初期化します。

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

その後モード判定を行います。

```
        if (Mode_Judgement() == SWP_BOOT_MODE) {    /* if SW4 is turned on,
enter SWP boot mode */                  }
```

現在のモードがSWP_BOOT_MODEモードの場合flash APIをRAMへコピーし、スワップサイズを8KBに設定します。

```
        Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
/* copy flash API to RAM */
        FC_SetSwpsrBit(FC_SWPSR_BIT_8);                 /* set SWP size to
8KB*/                }
```

次にスワップ状態を確認します。

```
        if (FC_GetSwapState() == FC_SWAP_INITIAL) {
```

スワップ状態がFC_SWAP_INITIALの場合、Flash ROMのプロテクトを解除し、スワップ状態を有効にします。

```
        if(FC_GetProtectState() == ENABLE){
                FC_EraseProtectState();
            } else {
                /* Do nothing */
            }
            Copy_Routine(SWP_RAM,  SWP_FLASH,  SIZE_SWP);    /*  copy
program to RAM */
            FC_ErasePage(PAGE2ADDR);
            FC_ErasePage(PAGE3ADDR);
            Write_Flash((uint32_t   *)PAGE2ADDR,   SWP_RAM,   SIZE_SWP);
/* write program to PAGE2ADDR */
            FC_SetSwpsrBit(FC_SWPSR_BIT_0);      /* enable swp function */
```

スワップ状態がFC_SWAP_INITIALではない場合、スワップ状態を解除します。

```
        } else {
            FC_EraseProtectState();                 /* release swp function */
        }
```

スワップ状態の有効または解除を行った後、LED2を点灯します。SW4が離されるのを待ち、ソフトウェアリセットを行います。

```
        delay(4000000U);
        LED_On(LED2);
        /* wait for Key SW4 to release */
        while(Mode_Judgement() == SWP_BOOT_MODE){
            /* Do nothing */
        }
        /* software reset */
        NVIC_SystemReset();
```

現在のモードがNORMAL_MODEの場合、LEDを点滅します (サンプルAの場合はLED0、サンプルBの場合はLED1です)

```
        while(1){
            LED_On(LED0);
            delay(4000000U);
            LED_Off(LED0);
            delay(4000000U);
        }
```

7-7 FUART

7-7-1 例: ループバック

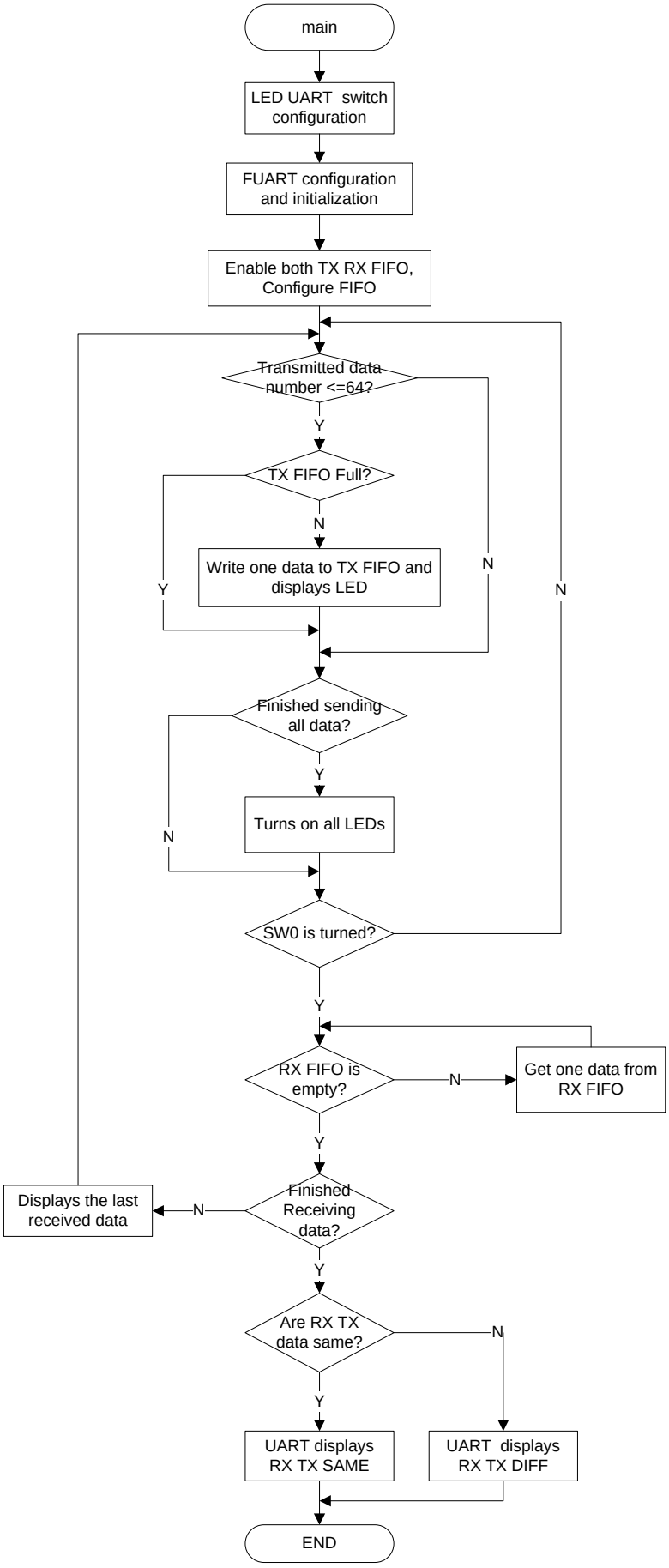
ペリフェラルドライバ(FUART, GPIO)を用いたサンプルプログラムです。

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 回実行することができます。
1 回目:UT0RTS と UT0CTS ハードウェアフロー制御をイネーブルにします。
2 回目:UT0RTS と UT0CTS ハードウェアフロー制御ををディセーブルにします。

この例は以下を行います。

1. LED、UART、スイッチの初期化、フル UART の設定と初期化
2. フル 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 UART switch configuration
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
N
SW0 is turned?
Y
RX FIFO is empty?
N
Get one data from RX FIFO
Y
Finished Receiving data?
N
Displays the last received data
Y
Are RX TX data same?
N
Y
UART displays RX TX SAME
UART displays RX TX DIFF
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、UT0RTS あるいは UT0CTS 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 が未定義の場合、UT0RTS あるいは UT0CTS フロー制御はプログラム内でイネーブルにします。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、プログラムは LED と UART を初期化します。
LED/UART 用に GPIO を設定します。

```
    LED_Init();
    SW_Init();
    hardware_init(UART_RETARGET);
```

FUART0 用 GPIO を設定します。

```
    /* Configure port PF1 to be UT0TXD */
    GPIO_SetOutput(GPIO_PF, GPIO_BIT_1);
    GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_3, GPIO_BIT_1);

    /* Configure port PF2 to be UT0RXD */
    GPIO_SetInput(GPIO_PF, GPIO_BIT_2);
    GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_3, GPIO_BIT_2);

    /* Configure port PF0 to be UT0CTS */
    GPIO_SetInput(GPIO_PF, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_3, GPIO_BIT_0);

    /* Configure port PF3 to be UT0RTS */
    GPIO_SetOutput(GPIO_PF, GPIO_BIT_3);
    GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_3, 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);
```

FUART 周辺ドライバを使用して FUART0 の許可、と FIFO の設定を行います。

```
    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。フル UART は 64 種のデータ値のみ送信したのち、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(64U==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。このとき UART に最終受信データを出力します。SW0 が On であり、データが存在しない場合には、UART に“RX FINISH” プログラムを出力し、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、UART には“RX TX SAME”と出力します。受信データが送信データと異なる場合、UART には “RX TX DIFF”と出力します。

```
    SW0_this = SW_Get(SW0);
    rxlast = rxthis;
    rxthis = cntRx;

    if (rxlast != rxthis) {        /* there are some data that has been received    */
        common_uart_disp("LAST RX DATA:");
        rxnum[0] = ('0' + receive/10U);
        rxnum[1] = ('0' + receive%10U);
        common_uart_disp(rxnum);     /* dispaly the last received data */
    } else {                    /* receiving data is finished */
        common_uart_disp("RX FINISH");
        result = Buffercompare(Tx_Buf, Rx_Buf, MAX_BUFSIZE);
        if (result == SAME) {
            /* received data are same with trnsmitted data */
            /* UT0RTS and UT0CTS flow control has worked normally */
                common_uart_disp("RX TX SAME");
                while(1){}
        } else {
            /* received data are different with trnsmitted data */
            /* UT0RTS and UT0CTS flow control doesn't work */
                common_uart_disp("RX TX DIFF");
                while(1){}
        }
    }
```

補足:

LED0~LED3 は PB0~PB3 と接続します。

SW0 は PJ0 と接続します。

7-8 GPIO

7-8-1 例: GPIO データリード

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

- **サンプルプログラムのコードと説明**

まず、LED 用 GPIO の設定を GPIO_SetOutput()関数を用いて行い、スイッチ用 GPIO の設定を GPIO_SetInput()関数を用いて行います。

```
    GPIO_SetOutput(GPIO_PB, GPIO_BIT_0 | GPIO_BIT_1 | GPIO_BIT_2 |
GPIO_BIT_3);

    GPIO_SetInput(GPIO_PJ,   GPIO_BIT_0 | GPIO_BIT_1 | GPIO_BIT_2 |
GPIO_BIT_3);
```

for(;;)内において、スイッチ状態に応じて LED 点灯/消灯を切り替えます。

GPIO_ReadDataBit()関数を使用してスイッチ状態を判断します。

```
    if (GPIO_ReadDataBit(GPIO_PJ, GPIO_BIT_0) == 1U) {
                    tmp = 1U;
    } else {
                /*Do nothing */
    }
```

GPIO_WriteData()関数を使用して LED を点灯します。

```
    uint8_t tmp;
    tmp = GPIO_ReadData(GPIO_PB);
    tmp |= led;
    GPIO_WriteData(GPIO_PB, tmp);
```

GPIO_WriteData()関数を使用して LED 消灯します。

```
    uint8_t tmp;
    tmp = GPIO_ReadData(GPIO_PB);
    tmp &= ~led;
    GPIO_WriteData(GPIO_PB, tmp);
```

7-9 I2C

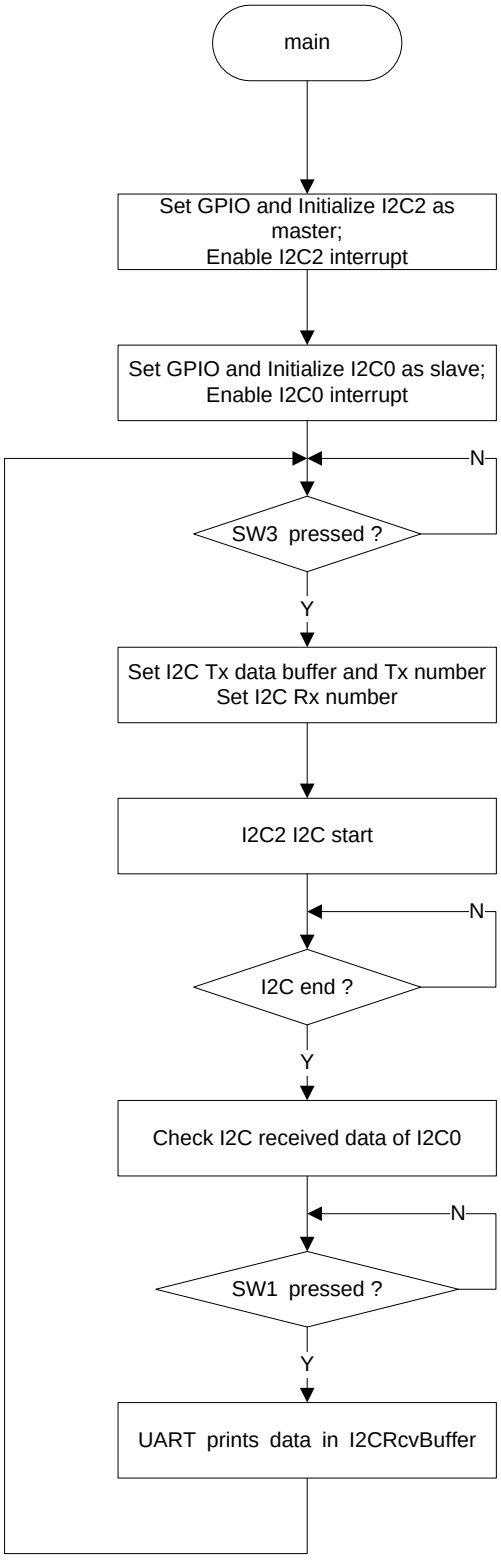
7-9-1 例: I2C Slave

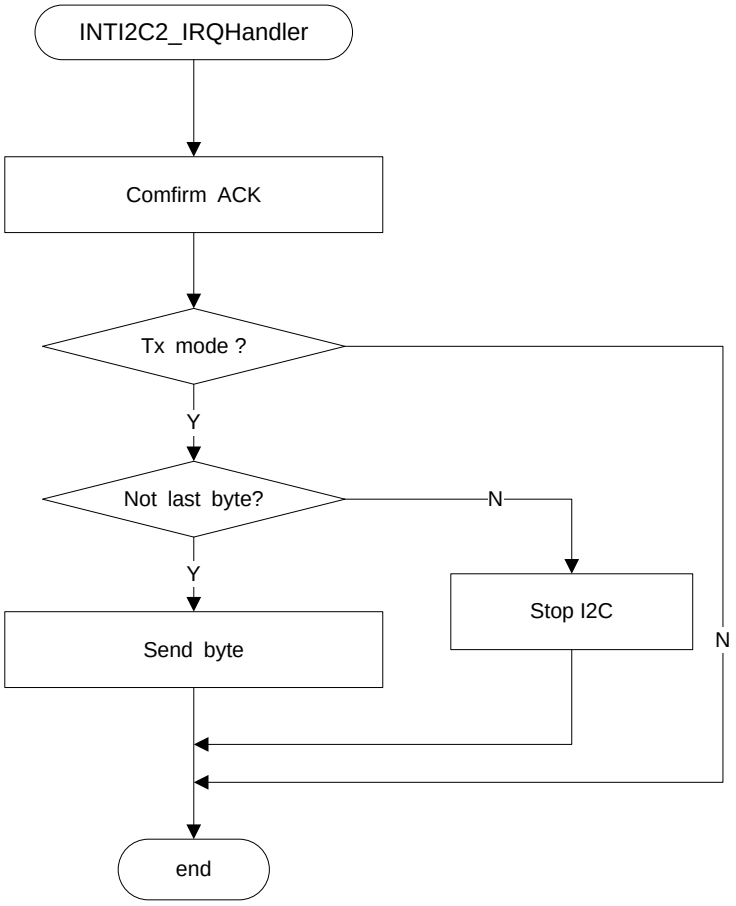
ペリフェラルドライバ (I2C, GPIO) のプログラムサンプルです。

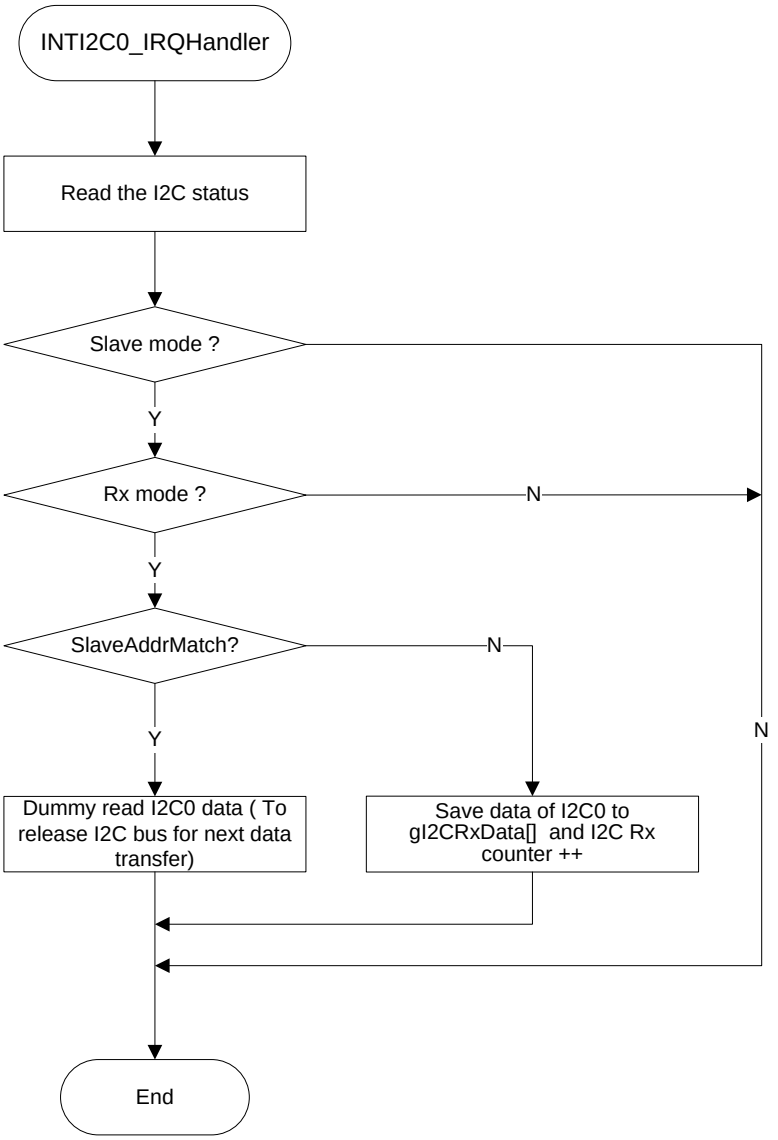
この例では以下を行います。

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

- **フローチャート:**


main
Set GPIO and Initialize I2C2 as master;
Enable I2C2 interrupt
Set GPIO and Initialize I2C0 as slave;
Enable I2C0 interrupt
N
SW3 pressed ?
Y
Set I2C Tx data buffer and Tx number
Set I2C Rx number
I2C2 I2C start
N
I2C end ?
Y
Check I2C received data of I2C0
N
SW1 pressed ?
Y
UART prints data in I2CRcvBuffer


INTI2C2_IRQHandler
Comfirm ACK
Tx mode ?
Y
Not last byte?
N
Y
Stop I2C
Send byte
N
end



- **サンプルプログラムのコードと説明**

まず、GPIO(I2C2)を I2C に設定します。

```
GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_4, GPIO_BIT_0);
GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_4, GPIO_BIT_1);
GPIO_SetOutputEnableReg(GPIO_PH, GPIO_BIT_0, ENABLE);
GPIO_SetOutputEnableReg(GPIO_PH, GPIO_BIT_1, ENABLE);
GPIO_SetInputEnableReg(GPIO_PH, GPIO_BIT_0, ENABLE);
GPIO_SetInputEnableReg(GPIO_PH, GPIO_BIT_1, ENABLE);
GPIO_SetOpenDrain(GPIO_PH, GPIO_BIT_0, ENABLE);
GPIO_SetOpenDrain(GPIO_PH, GPIO_BIT_1, ENABLE);
```

次に GPIO(I2C0)を I2C に設定します。

```
GPIO_EnableFuncReg(GPIO_PK, GPIO_FUNC_REG_3, GPIO_BIT_2);
GPIO_EnableFuncReg(GPIO_PK, GPIO_FUNC_REG_3, GPIO_BIT_3);
```

```
    GPIO_SetOutputEnableReg(GPIO_PK, GPIO_BIT_2, ENABLE);
    GPIO_SetOutputEnableReg(GPIO_PK, GPIO_BIT_3, ENABLE);
    GPIO_SetInputEnableReg(GPIO_PK, GPIO_BIT_2, ENABLE);
    GPIO_SetInputEnableReg(GPIO_PK, GPIO_BIT_3, ENABLE);
    GPIO_SetOpenDrain(GPIO_PK, GPIO_BIT_2, ENABLE);
    GPIO_SetOpenDrain(GPIO_PK, GPIO_BIT_3, ENABLE);
```

I2C2 の許可と初期設定を行い、その後 INTI2C2 割り込みを許可します。

```
  myI2C.I2CSelfAddr = SELF_ADDR;
  myI2C.I2CDataLen = I2C_DATA_LEN_8;
  myI2C.I2CACKState = ENABLE;
  myI2C.I2CClkDiv = I2C_SCK_CLK_DIV_32;
  myI2C.PrescalerClkDiv = I2C_PRESCALER_DIV_12;
  I2C_SWReset(TSB_I2C2);
  I2C_Init(TSB_I2C2, &myI2C);
  NVIC_EnableIRQ(INTI2C2_IRQn);
  I2C_SetINTReq(TSB_I2C2,ENABLE);
```

I2C0 の許可と初期設定を行い、その後 INTI2C0 割り込みを許可します。

```
  myI2C.I2CSelfAddr = SLAVE_ADDR;
  myI2C.I2CDataLen = I2C_DATA_LEN_8;
  myI2C.I2CACKState = ENABLE;
  myI2C.I2CClkDiv = I2C_SCK_CLK_DIV_32;
  myI2C.PrescalerClkDiv = I2C_PRESCALER_DIV_12;
  I2C_SWReset(TSB_I2C0);
  I2C_Init(TSB_I2C0, &myI2C);
  NVIC_EnableIRQ(INTI2C0_IRQn);
  I2C_SetINTReq(TSB_I2C0,ENABLE);
```

上記設定を行った後、I2C 送信を開始します。
I2C 受信バッファのクリア、I2C 送信バッファとバッファ長の初期設定を行います。

```
        /* Initialize TRx buffer and Tx length */
    case MODE_I2C_INITIAL:
        gI2CTxDataLen = 8U;
        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;
        }
        gI2CMode = MODE_I2C_START;
        break;
```

I2C バスが空いているかどうか、“SLAVE_ADDR” データを I2C_SetSendData()に設定します。そして、送信方向を“I2C_SEND”から I2C データバッファへ設定します。その後、I2C_GenerateStart(TSB_I2C2) を使用して、I2C 動作を開始します。

```
        /* Check I2C bus state and start TRx */
    case MODE_I2C_START:
```

```
        i2c_state = I2C_GetState(TSB_I2C2);
        if (!i2c_state.Bit.BusState) {
            I2C_SetSendData(TSB_I2C2, SLAVE_ADDR | I2C_SEND);
            I2C_GenerateStart(TSB_I2C2);
            gI2CMode = MODE_I2C_TRX;
        } else {
            /* Do nothing */
        }
        break;
```

INTI2C2 でデータ転送を行います。

INTI2C0 でデータ受信を行います。

INTI2C2 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 INTI2C2_IRQHandler(void)
{
    TSB_I2C_TypeDef *I2Cx;
    I2C_State i2c_sr;

    I2Cx = TSB_I2C2;
    I2c_sr = I2C_GetState(I2Cx);

    if (i2c_sr.Bit.MasterSlave) {           /* Master mode */
        if (i2c_sr.Bit.TRx) {     /* Tx mode */
            if (i2c_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further data. */
                I2C_GenerateStop(I2Cx);
            } else {                /* LRB=0: the receiver requires further data. */
                if (gI2CWCnt <= gI2CTxDataLen) {
                    I2C_SetSendData(I2Cx, gI2CTxData[gI2CWCnt]);          /* Send next data */
                    gI2CWCnt++;
                } else {            /* I2C data send finished. */
                    I2C_GenerateStop(I2Cx);    /* Stop I2C */
                }
            }
        } else {                    /* Rx Mode */
            /* Do nothing */
        }
    } else {                        /* Slave mode */
        /* Do nothing */
    }
}
```

INTI2C0 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 INTI2C0_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_I2C_TypeDef *I2Cy;
    I2C_State I2c0_sr;
```

```
    I2Cy = TSB_I2C0;
    i2c0_sr = I2C_GetState(I2Cy);

    if (!i2c0_sr.Bit.MasterSlave) {        /* Slave mode */
        if (!i2c0_sr.Bit.TRx) { /* Rx Mode */
            if (i2c0_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = I2C_GetReceiveData(I2Cy);
                gI2CRCnt = 0U;
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = I2C_GetReceiveData(I2Cy);
                gI2CRxData[gI2CRCnt] = tmp;
                gI2CRCnt++;
            }
        } else {                        /* Tx Mode */
            /* Do nothing */
        }
    } else {                            /* Master mode */
        /* Do nothing */
    }
}
```

7-10 IGBT

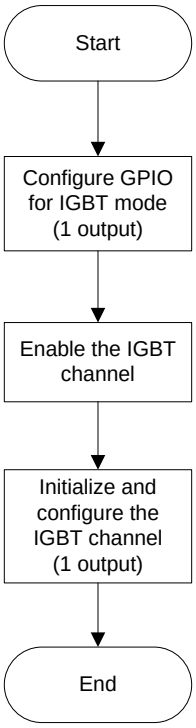
7-10-1 例: 1 相 PPG 出力

ペリフェラルドライバ(IGBT , 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. IGBT タイマ設定と初期化 (1 相 PPG 出力).
2. IGBT タイマ EMG 保護機能
3. IGBT タイマ状態と EMG 割り込みの使用方法

- **フローチャート:**



- **サンプルプログラムのコードと説明**

LED 設定後、IGBT_InitTypeDef 構成を生成し、全データフィールドを入力します。

```
    IGBT_InitTypeDef myIGBT;
    /* IGBT trigger start: falling edge start and active level is "Low" */
    myIGBT.StartMode = IGBT_FALLING_TRG_START;
    /* IGBT operation: continuous operation */
    myIGBT.OperationMode = IGBT_CONTINUOUS_OUTPUT;
    /* IGBT stopping status: initial output satus and counter */
    myIGBT.CntStopState = IGBT_OUTPUT_INACTIVE;
    /* Trigger edge accept mode: Don't accept trigger during active level */
    myIGBT.ActiveAcceptTrg = DISABLE;
    /* Interrupt cycle: Every one cycle */
    myIGBT.INTPeriod = IGBT_INT_PERIOD_1;
    /* For M46B: fperiph = fc = 16MHz*4 = 64MHz, T0 = fperiph = 64MHz, figbt = T0/2 =
32MHz */
    myIGBT.ClkDiv = IGBT_CLK_DIV_2;
    /* MT0OUT0 initial state is Low, and active level is High */
    myIGBT.Output0Init = IGBT_OUTPUT_HIGH_ACTIVE;
    /* Disable MT0OUT1 output */
    myIGBT.Output1Init = IGBT_OUTPUT_DISABLE;
    /* Trigger input noise elimination time: 240/fsys */
    myIGBT.TrgDenoiseDiv = IGBT_DENOISE_DIV_240;
    myIGBT.Output0ActiveTiming = 1U;
    myIGBT.Output0InactiveTiming = 1U + IGBT_PPG_PERIOD_50US / 2;
```

```
    /* Period: 50us */
    myIGBT.Period = IGBT_ PPG _PERIOD_50US;
    /* The polarity of MTOUT0x at EMG protection: High-impedance */
    myIGBT.EMGFunction = IGBT_EMG_OUTPUT_HIZ;
    /* EMG input noise elimination time: 240/fsys */
    myIGBT.EMGDenoiseDiv = IGBT_DENOISE_DIV_240;
```

IGBT チャネルと EMG 保護割り込みをイネーブルにし、初期化と構成設定前に EMG 状態をキャンセルします。

```
    /* Enable IGBT and EMG interrupt */
    IGBT_Enable(IGBT0);
    NVIC_EnableIRQ(INTMTEMG0_IRQn);

    /* If the timer is still running, wait until it stops */
    do {
        counter_state = IGBT_GetCntState(IGBT0);
    } while (counter_state == BUSY);
    /* Cancel the EMG protection state */
    do {
        cancel_result = IGBT_CancelEMGState(IGBT0);
    } while (cancel_result == ERROR);
```

IGBT_CancelEMGState() の戻り値が SUCCESS の場合、IGBT_Init()を呼び出して IBGT の初期化を行います。

```
  IGBT_Init(IGBT0, &myIGBT);
```

スタートコマンドを送信して IGBT タイマを開始します。

```
  IGBT_SetSWRunState(IGBT0, IGBT_RUN);
```

その後、対応するポートは開始トリガを検出するごとに PPG 波形を出力します。

GEMG の EMG 保護レベルがアクティブの場合、EMG 割り込みが発生します。LED1 を点灯し、タイマを停止します。

```
  void INTMTEMG0_IRQHandler(void)
  {
      /* If EMG protection, turn on the LED and stop the timer */
      LED_On(LED1);
      IGBT_SetSWRunState(IGBT0, IGBT_STOP);
  }
```

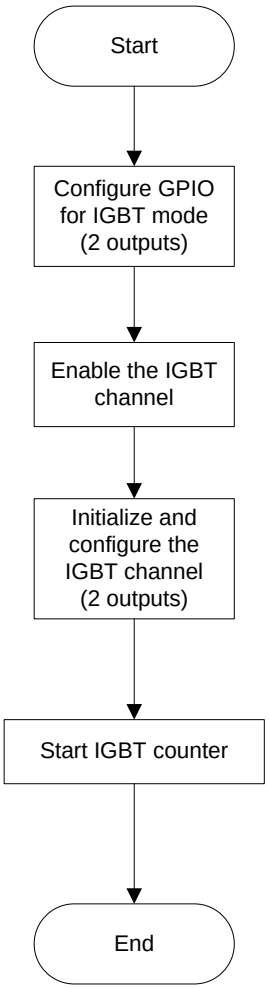
7-10-2　例: 2 相 PPG 出力

ペリフェラルドライバ(IGBT , 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. IGBT タイマ設定と初期化 (2 相 PPG 出力)
2. IGBT タイマ EMG 保護機能
3. IGBT タイマ状態と EMG 割り込みの使用方法

- **フローチャート:**



- **サンプルプログラムのコードと説明**

LED およびスイッチポートの設定後、IGBT_InitTypeDef 構造を生成し、全データフィールドを入力します。

```
IGBT_InitTypeDef myIGBT;
/* IGBT trigger start: command start */
```

```
    myIGBT.StartMode = IGBT_CMD_START;
    /* IGBT operation: continuous operation */
    myIGBT.OperationMode = IGBT_CONTINUOUS_OUTPUT;
    /* IGBT stopping status: initial output satus and counter */
    myIGBT.CntStopState = IGBT_OUTPUT_INACTIVE;
    /* Trigger edge accept mode: Don't accept trigger during active level */
    myIGBT.ActiveAcceptTrg = DISABLE;
    /* Interrupt cycle: Every one cycle */
    myIGBT.INTPeriod = IGBT_INT_PERIOD_1;
    /* For M46B: fperiph = fc = 16MHz*4 = 64MHz, T0 = fperiph = 64MHz, figbt = T0/2 =
32MHz */
    myIGBT.ClkDiv = IGBT_CLK_DIV_2;
    /* MT0OUT0 initial state is Low, and active level is High */
    myIGBT.Output0Init = IGBT_OUTPUT_HIGH_ACTIVE;
    /* MT0OUT1 initial state is Low, and active level is High */
    myIGBT.Output1Init = IGBT_OUTPUT_HIGH_ACTIVE;
    /* Trigger input noise elimination time: no use */
    myIGBT.TrgDenoiseDiv = IGBT_NO_DENOISE;
    myIGBT.Output0ActiveTiming = 1U;
    myIGBT.Output0InactiveTiming = 1U + IGBT_ACTIVE_PERIOD_20US;
    myIGBT.Output1ActiveTiming = myIGBT.Output0InactiveTiming +
                                 IGBT_DEAD_TIME_5US;
    myIGBT.Output1InactiveTiming = myIGBT.Output1ActiveTiming +
                                   IGBT_ACTIVE_PERIOD_20US;
    /* Period: 50us */
    myIGBT.Period = IGBT_PPG_PERIOD_50US;
    /* The polarity of MTOUT0x at EMG protection: High-impedance */
    myIGBT.EMGFunction = IGBT_EMG_OUTPUT_HIZ;
    /* EMG input noise elimination time: 240/fsys */
    myIGBT.EMGDenoiseDiv = IGBT_DENOISE_DIV_240;
```

IGBT チャネルと EMG 保護割り込みを有効にします。
初期設定を行う前に EMG 状態をキャンセルします。

```
    /* Enable IGBT and EMG interrupt */
    IGBT_Enable(IGBT0);
    NVIC_EnableIRQ(INTMTEMG0_IRQn);

    /* If the timer is still running, wait until it stops */
    do {
        counter_state = IGBT_GetCntState(IGBT0);
    } while (counter_state == BUSY);
    /* Cancel the EMG protection state */
    do {
        cancel_result = IGBT_CancelEMGState(IGBT0);
```

```
    } while (cancel_result == ERROR);
```

IGBT_CancelEMGState() の戻り値が SUCCESS の場合、IGBT_Init()を呼び出して IBGT の初期化を行います。

```
  IGBT_Init(IGBT0, &myIGBT);
```

スイッチが ON になると、スタートコマンドを送信して IGBT タイマを開始します。

```
  /* If switch is ON, start to run IGBT timer */
  if (SWITCH_ON) {
      IGBT_SetSWRunState(IGBT0, IGBT_RUN);
  } else {
    /* Do nothing */
  }
```

GEMG の EMG 保護レベルがアクティブの場合、EMG 割り込みが発生します。LED1 を点灯し、タイマを停止します。

```
  void INTMTEMG0_IRQHandler(void)
  {
        /* If EMG protection, turn on the LED and stop the timer */
        LED_On(LED1);
        IGBT_SetSWRunState(IGBT0, IGBT_STOP);
  }
```

7-11　LVD

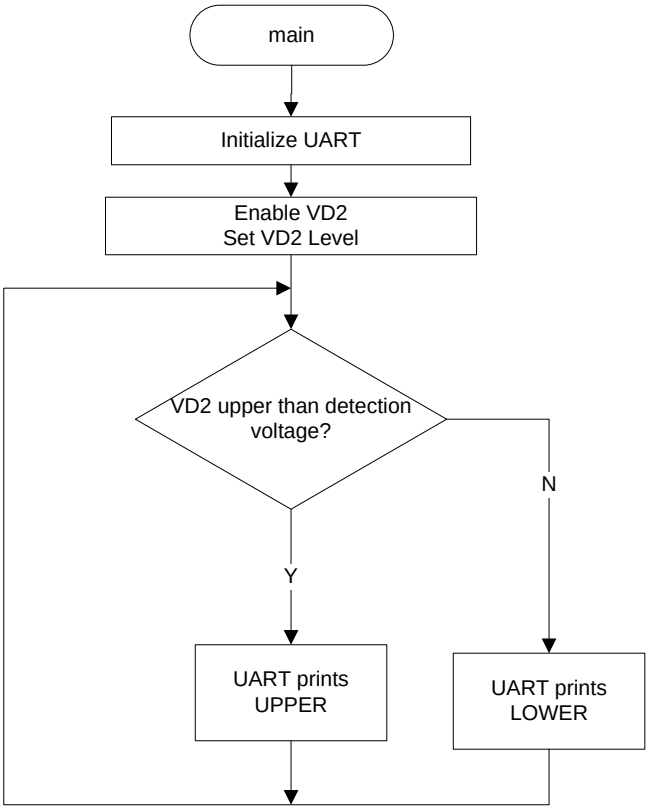
7-11-1　例: LVD

ペリフェラルドライバ(LVD, 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. LVD の初期化。
2. LVD 状態の監視。

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初にシステムを初期化します。不用意な WDT 割り込みを避けるため WDT を無効にします。

```
hardware_init(UART_RETARGET)                    /* Initialize UART */;
```

電圧検出の設定モードと検出レベルをイネーブルにします。

```
LVD_EnableVD();
LVD_SetVDLevel(LVD_VDLVL_315);
```

その後 while(1)内にて VD の状態を監視します。
VD が検出電圧より高い場合、UART に“UPPER”を出力し、VD が検出電圧より低い場合、UART 上には“LOWER”と出力します。

```
while (1) {
    if (LVD_GetVDStatus() == LVD_VD_UPPER) {
        common_uart_disp(“UPPER\n”);
    } else {
        common_uart_disp(“LOWER\n”);
    }
}
```

7-12 MLA

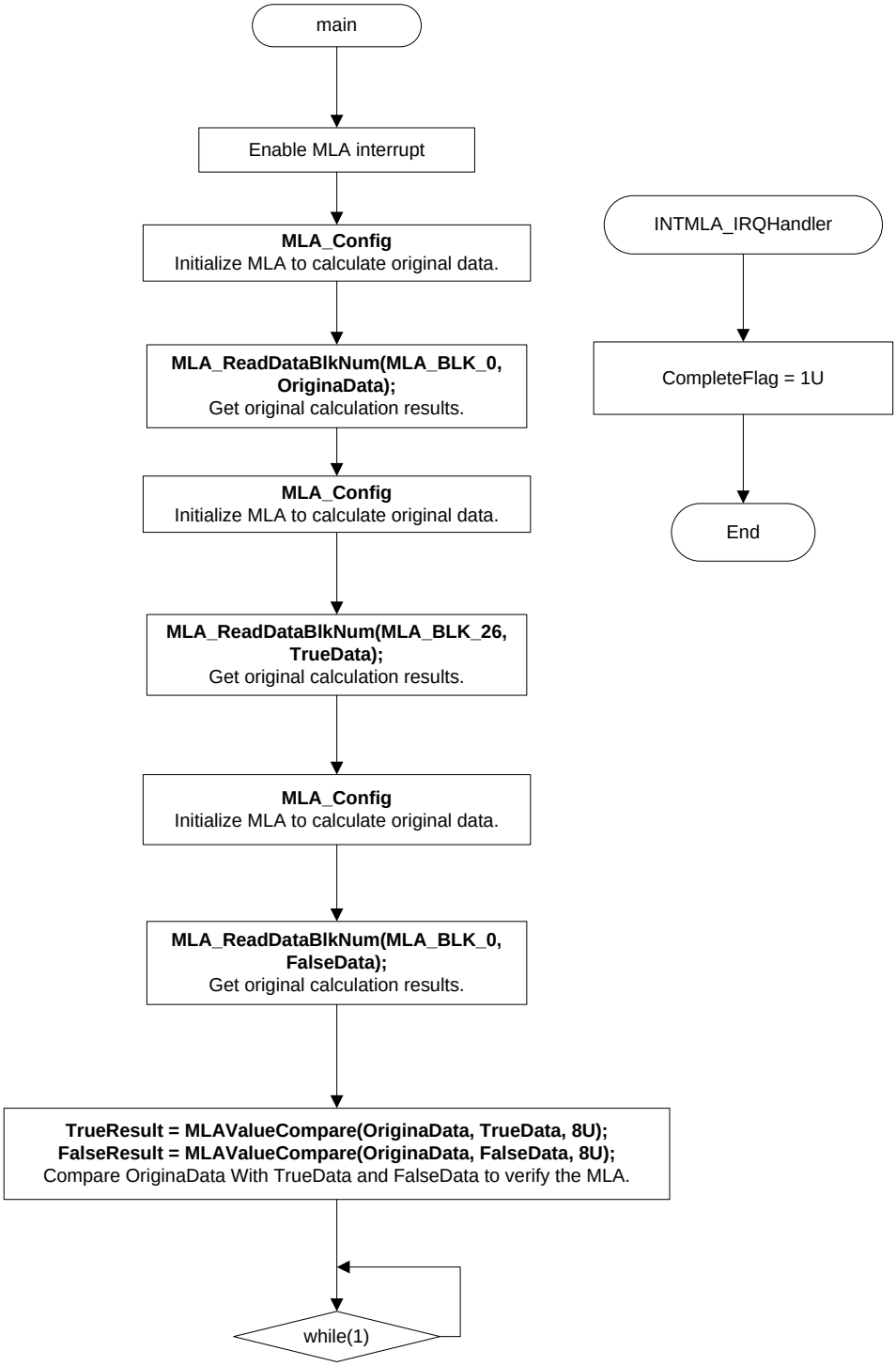
7-12-1 例: MLA モンゴメリ乗算モード

ペリフェラルドライバ(MLA)を使用したサンプルプログラムです。

このサンプルプログラムでは以下を行います。

1. モンゴメリ乗算モードの設定
2. 演算結果を取得します。

- **フローチャート**


main
Enable MLA interrupt
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_0, OriginaData);
Get original calculation results.
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_26, TrueData);
Get original calculation results.
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_0, FalseData);
Get original calculation results.
TrueResult = MLAValueCompare(OriginaData, TrueData, 8U);
FalseResult = MLAValueCompare(OriginaData, FalseData, 8U);
Compare OriginaData With TrueData and FalseData to verify the MLA.
while(1)
INTMLA_IRQHandler
CompleteFlag = 1U
End

- **サンプルプログラムのコードと説明**

MLA 割り込みの設定を行います。

```
/* enable MLA calculation completion interrupt */
NVIC_ClearPendingIRQ(INTMLA_IRQn);
NVIC_EnableIRQ(INTMLA_IRQn);
 __enable_irq();
```

演算入力データを指定されたデータブロックへ設定します。次にモンゴメリ乗算データ除数要因とモンゴメリパラメータの設定を行います。モンゴメリ演算終了後、除数を格納するレジスタとして MLABLK01 を使用します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_2, OriginaAData);
MLA_WriteDataBlkNum(MLA_BLK_5, OriginaBData);
/* Sets the divisor used to Montgomery calculation */
MLA_WriteDataBlkNum(MLA_BLK_1, DivisorData) ;
/* Sets montgomery parameter */
MLA_SetMontgomeryParameter(0x12345678);
MLA_Config(MLA_BLK_2, MLA_BLK_5, MLA_BLK_0);
```

MLA 演算モードの設定を行い、MLA コアをモンゴメリ乗算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();

    /* Set the calculation mode */
    MLA_SetCalculationMode(MLA_COM_MODE_MUL);
}
```

演算結果を取得し、データ出力を行います。

```
/* Wait for calculation completion */
while (CompleteFlag != 1U) {
    /* Do nothing */
}
/* Get the calculation result */
Delay();
MLA_ReadDataBlkNum(MLA_BLK_0, OriginaData);
```

演算入力データを指定されたデータブロックへ設定します。次にモンゴメリ乗算除数要因

とモンゴメリパラメータの設定を行います。モンゴメリ演算の動作時に除数の格納用にMLABLK01 を使用します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_20, TrueAData);
MLA_WriteDataBlkNum(MLA_BLK_22, TrueBData);
/* Sets the divisor used to Montgomery calculation */
MLA_WriteDataBlkNum(MLA_BLK_1, DivisorData) ;
/* Sets montgomery parameter */
MLA_SetMontgomeryParameter(0x12345678);
MLA_Config(MLA_BLK_20, MLA_BLK_22, MLA_BLK_26);
```

MLA 演算モードの設定を行い、MLA コアをモンゴメリ乗算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();

    /* Set the calculation mode */
    MLA_SetCalculationMode(MLA_COM_MODE_MUL);
}
```

演算結果を取得し、データ出力を行います。

```
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /* Get the calculation result */
        Delay();
        MLA_ReadDataBlkNum(MLA_BLK_26, TrueData);
```

演算入力データを指定されたデータブロックへ設定します。次にモンゴメリ乗算除数要因とモンゴメリパラメータの設定を行います。モンゴメリ演算の動作時に除数の格納用にMLABLK01 を使用します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_2, FalseAData);
MLA_WriteDataBlkNum(MLA_BLK_5, FalseBData);
/* Sets the divisor used to Montgomery calculation */
MLA_WriteDataBlkNum(MLA_BLK_1, DivisorData) ;
/* Sets montgomery parameter */
MLA_SetMontgomeryParameter(0x12345678);
```

```
MLA_Config(MLA_BLK_2, MLA_BLK_5, MLA_BLK_0);
```

MLA 演算モードの設定を行い、MLA コアをモンゴメリ乗算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();

    /* Set the calculation mode */
    MLA_SetCalculationMode(MLA_COM_MODE_MUL);
}
```

演算結果を取得し、データ出力を行います。

```
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /* Get the calculation result */
    Delay();
    MLA_ReadDataBlkNum(MLA_BLK_0, FalseData);
```

MLA 処理の確認のため、OriginaData と TrueData、OriginalData と FalseData をそれぞれ比較します。TrueResult = SUCCESS かつ FalseResult = ERROR の場合は比較結果に問題がないと判断します。

```
/* Compare original data calculation results with true data calculation results. */
TrueResult = MLAValueCompare(OriginaData, TrueData, 8U);
/* Compare original data calculation results with false data calculation results. */
FalseResult = MLAValueCompare(OriginaData, FalseData, 8U);

if ((TrueResult == SUCCESS) && (FalseResult == ERROR)) {
    common_uart_disp("MLA processed successfully !\n");
} else {
    common_uart_disp("MLA processed with error !\n");
}
```

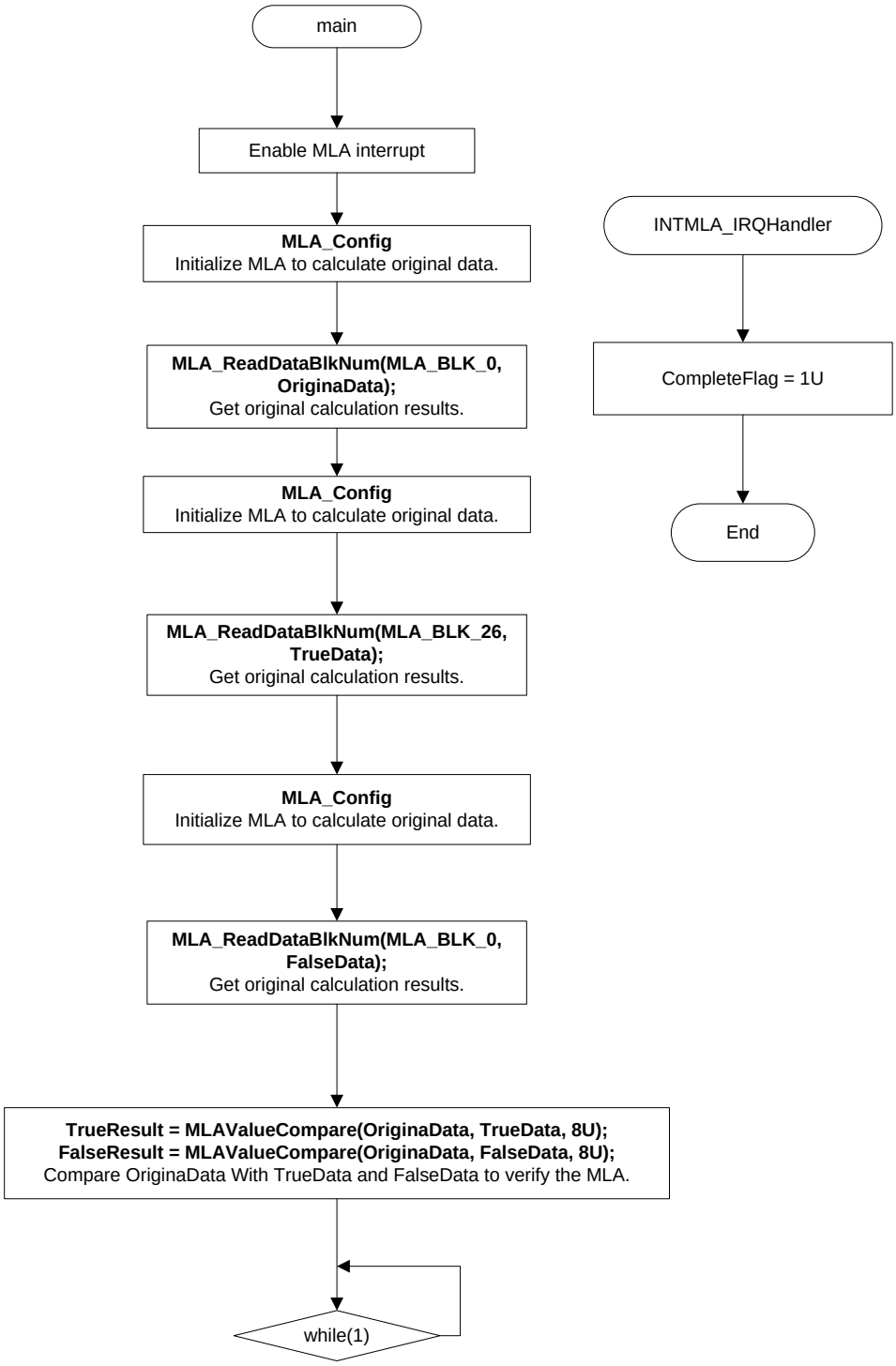
7-12-2　例: MLA 多倍長加算モード

ペリフェラルドライバ(MLA)を使用したサンプルプログラムです。

このサンプルプログラムでは以下を行います。

1. 多倍長加算モードの設定を行います。
2. 演算結果の取得を行います。

- **フローチャート**


main
Enable MLA interrupt
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_0, OriginaData);
Get original calculation results.
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_26, TrueData);
Get original calculation results.
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_0, FalseData);
Get original calculation results.
TrueResult = MLAValueCompare(OriginaData, TrueData, 8U);
FalseResult = MLAValueCompare(OriginaData, FalseData, 8U);
Compare OriginaData With TrueData and FalseData to verify the MLA.
while(1)
INTMLA_IRQHandler
CompleteFlag = 1U
End

- **サンプルプログラムのコードと説明**

MLA 割り込みの設定を行います。

```
/* enable MLA calculation completion interrupt */
NVIC_ClearPendingIRQ(INTMLA_IRQn);
NVIC_EnableIRQ(INTMLA_IRQn);
__enable_irq();
```

演算入力データを指定されたデータブロックへ設定します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_2, OriginaAData);
MLA_WriteDataBlkNum(MLA_BLK_5, OriginaBData);
MLA_Config(MLA_BLK_2, MLA_BLK_5, MLA_BLK_0);
```

MLA 演算モードの設定を行い、MLA コアを多倍長乗算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();

    /* Set the calculation mode */
    MLA_SetCalculationMode(MLA_COM_MODE_ADD);
}
```

演算結果を取得し、データ出力を行います。

```
/* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /* Get the calculation result */
        MLA_ReadDataBlkNum(MLA_BLK_0, OriginaData);
```

演算入力データを指定されたデータブロックに設定します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_20, TrueAData);
MLA_WriteDataBlkNum(MLA_BLK_22, TrueBData);
MLA_Config(MLA_BLK_20, MLA_BLK_22, MLA_BLK_26);
```

MLA 演算モードの設定を行い、MLA コアを多倍長乗算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロ

ック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();

    /* Set the calculation mode */
    MLA_SetCalculationMode(MLA_COM_MODE_ADD);
}
```

演算結果を取得します。

```
/* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /* Get the calculation result */
        MLA_ReadDataBlkNum(MLA_BLK_26, TrueData);
```

演算入力データを指定されたブロックに設定します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_2, FalseAData);
MLA_WriteDataBlkNum(MLA_BLK_5, FalseBData);
MLA_Config(MLA_BLK_2, MLA_BLK_5, MLA_BLK_0);
```

MLA 演算モードの設定を行い、MLA コアを多倍長乗算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();
```

```
        /* Set the calculation mode */
        MLA_SetCalculationMode(MLA_COM_MODE_ADD);
    }
```

演算結果を取得し、データ出力を行います。

```
    /* Wait for calculation completion */
            while (CompleteFlag != 1U) {
                /* Do nothing */
            }
            /* Get the calculation result */
            MLA_ReadDataBlkNum(MLA_BLK_0, FalseData);
```

MLA 処理の確認のため、OriginaData と TrueData、OriginalData と FalseData をそれぞれ比較します。TrueResult = SUCCESS かつ FalseResult = ERROR の場合は比較結果に問題がないと判断します。

```
    /* Compare original data calculation results with true data calculation results. */
    TrueResult = MLAValueCompare(OriginaData, TrueData, 8U);
    /* Compare original data calculation results with false data calculation results. */
    FalseResult = MLAValueCompare(OriginaData, FalseData, 8U);

    if ((TrueResult == SUCCESS) && (FalseResult == ERROR)) {
        common_uart_disp("MLA processed successfully !\n");
    } else {
        common_uart_disp("MLA processed with error !\n");
    }
```

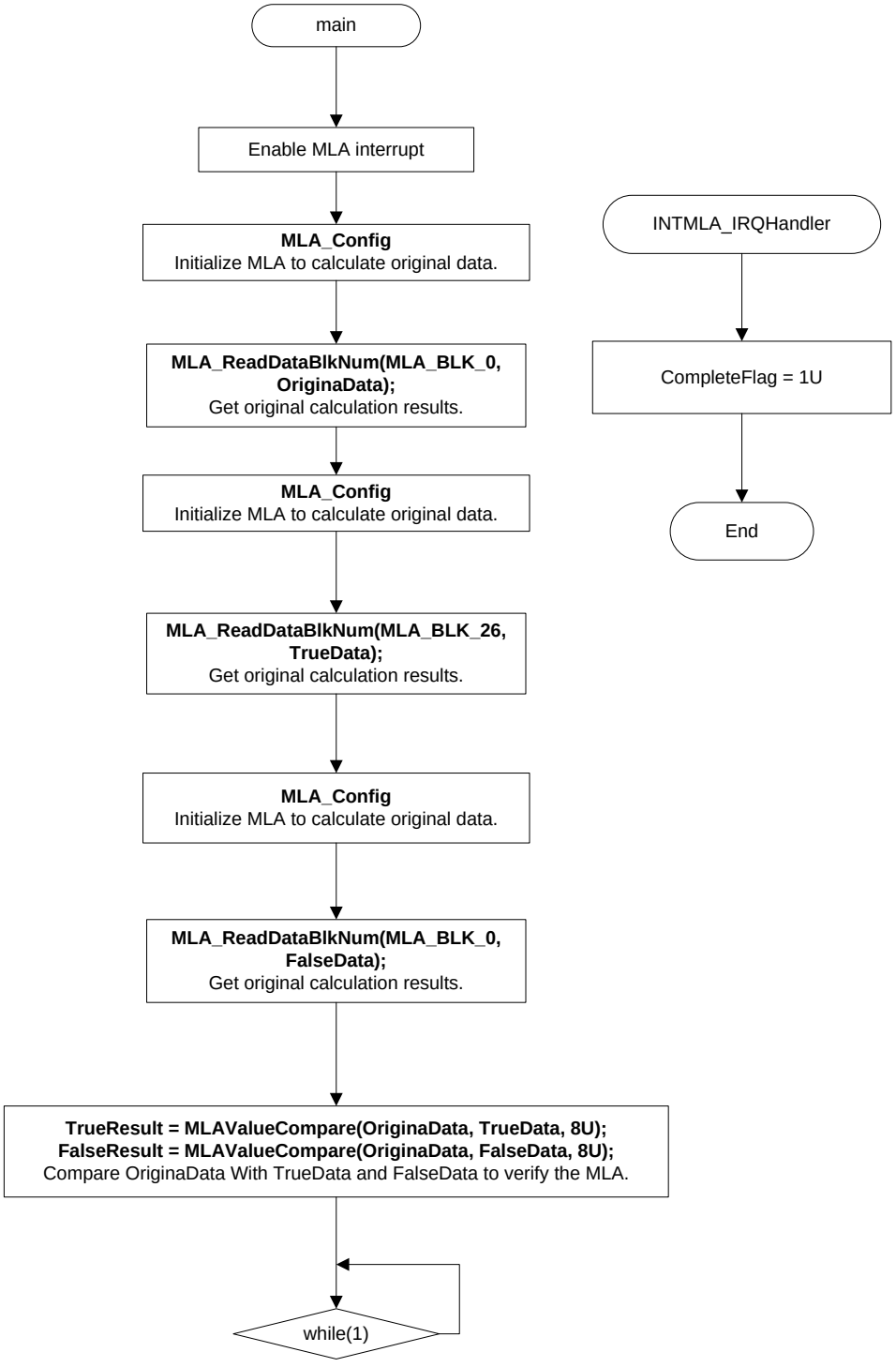
7-12-3　例: MLA 多倍長減算モード

ペリフェラルドライバ(MLA)を使用したサンプルプログラムです。

このサンプルプログラムでは以下を行います。

1. 多倍長減算モードの設定を行います。
2. 演算結果の取得を行います。

- **フローチャート**


main
Enable MLA interrupt
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_0, OriginaData);
Get original calculation results.
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_26, TrueData);
Get original calculation results.
MLA_Config
Initialize MLA to calculate original data.
MLA_ReadDataBlkNum(MLA_BLK_0, FalseData);
Get original calculation results.
TrueResult = MLAValueCompare(OriginaData, TrueData, 8U);
FalseResult = MLAValueCompare(OriginaData, FalseData, 8U);
Compare OriginaData With TrueData and FalseData to verify the MLA.
while(1)
INTMLA_IRQHandler
CompleteFlag = 1U
End

- **サンプルプログラムのコードと説明**

MLA 割り込みの設定を行います。

```
/* enable MLA calculation completion interrupt */
NVIC_ClearPendingIRQ(INTMLA_IRQn);
NVIC_EnableIRQ(INTMLA_IRQn);
  __enable_irq();
```

演算入力データを指定されたデータブロックへ設定します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_2, OriginaAData);
MLA_WriteDataBlkNum(MLA_BLK_5, OriginaBData);
MLA_Config(MLA_BLK_2, MLA_BLK_5, MLA_BLK_0);
```

MLA 演算モードの設定を行い、MLA コアを多倍長減算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
      /* Check the parameters */
      assert_param(IS_MLA_BLK_NUM(ABlkNum));
      assert_param(IS_MLA_BLK_NUM(BBlkNum));
      assert_param(IS_MLA_BLK_NUM(WBlkNum));

      /* Set the data block number */
      MLA_SetADataBlkNum(ABlkNum);
      Delay();
      MLA_SetBDataBlkNum(BBlkNum);
      Delay();
      MLA_SetWDataBlkNum(WBlkNum);
      Delay();

      /* Set the calculation mode */
      MLA_SetCalculationMode(MLA_COM_MODE_SUB);
}
```

演算結果を取得し、データ出力を行います。

```
/* Wait for calculation completion */
            while (CompleteFlag != 1U) {
                  /* Do nothing */
            }
            /* Get the calculation result */
            MLA_ReadDataBlkNum(MLA_BLK_0, OriginaData);
```

演算入力データを指定されたデータブロックに設定します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_20, TrueAData);
MLA_WriteDataBlkNum(MLA_BLK_22, TrueBData);
MLA_Config(MLA_BLK_20, MLA_BLK_22, MLA_BLK_26);
```

MLA 演算モードの設定を行い、MLA コアを多倍長減算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロ

ック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();

    /* Set the calculation mode */
    MLA_SetCalculationMode(MLA_COM_MODE_SUB);
}
```

演算結果を取得します。

```
/* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
        /* Get the calculation result */
        MLA_ReadDataBlkNum(MLA_BLK_26, TrueData);
```

演算入力データを指定されたブロックに設定します。

```
/* Sets calculation input data. */
MLA_WriteDataBlkNum(MLA_BLK_2, FalseAData);
MLA_WriteDataBlkNum(MLA_BLK_5, FalseBData);
MLA_Config(MLA_BLK_2, MLA_BLK_5, MLA_BLK_0);
```

MLA 演算モードの設定を行い、MLA コアを多倍長減算モードに設定します。次に演算データを入力するレジスタ用にブロック番号を入力します。結果を出力するレジスタ用にブロック番号を入力します。

```
/* Initialize MLA calculation mode */
void MLA_Config(uint8_t ABlkNum, uint8_t BBlkNum, uint8_t WBlkNum)
{
    /* Check the parameters */
    assert_param(IS_MLA_BLK_NUM(ABlkNum));
    assert_param(IS_MLA_BLK_NUM(BBlkNum));
    assert_param(IS_MLA_BLK_NUM(WBlkNum));

    /* Set the data block number */
    MLA_SetADataBlkNum(ABlkNum);
    Delay();
    MLA_SetBDataBlkNum(BBlkNum);
    Delay();
    MLA_SetWDataBlkNum(WBlkNum);
    Delay();
```

```
        /* Set the calculation mode */
        MLA_SetCalculationMode(MLA_COM_MODE_SUB);
    }
```

演算結果を取得し、データ出力を行います。

```
    /* Wait for calculation completion */
            while (CompleteFlag != 1U) {
                /* Do nothing */
            }
            /* Get the calculation result */
            MLA_ReadDataBlkNum(MLA_BLK_0, FalseData);
```

MLA 処理の確認のため、OriginaData と TrueData、OriginalData と FalseData をそれぞれ比較します。TrueResult = SUCCESS かつ FalseResult = ERROR の場合は比較結果に問題がないと判断します。

```
    /* Compare original data calculation results with true data calculation results */
        TrueResult = MLAValueCompare(OriginaData, TrueData, 8U);
    /* Compare original data calculation results with false data calculation results */
        FalseResult = MLAValueCompare(OriginaData, FalseData, 8U);

    if ((TrueResult == SUCCESS) && (FalseResult == ERROR)) {
        common_uart_disp("SHA processed successfully !\n");
    } else {
        common_uart_disp("SHA processed with error !\n");
    }
```

7-13 RTC

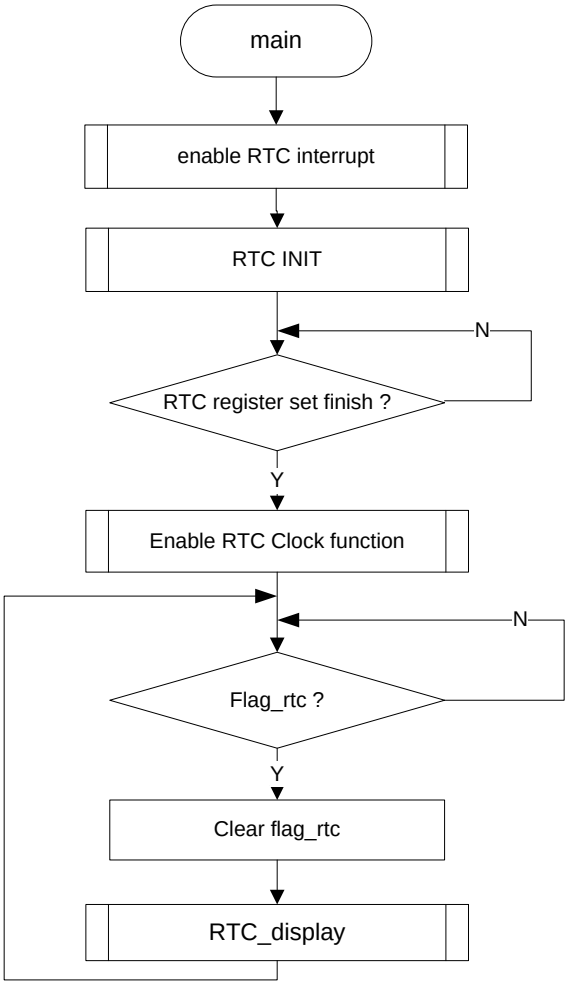
7-13-1 例: RTC

ペリフェラルドライバ (RTC, CG) のサンプルプログラムです。

この例では以下を行います。

1 RTC の時間設定
2 RTC の時間取得

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、スリープモードから抜けるためにソース(RTC 割り込み)を設定します。

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

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

```
RTC_DateTypeDef Date_Struct;
RTC_TimeTypeDef Time_Struct;

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

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

```
    Time_Struct.Sec = (uint8_t) 55U;
```

クロックとアラーム機能を禁止します。

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

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

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

RTC の時間と日付の値を設定します。

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

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

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

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

```
    fRTC_1HZ_INT = 1U;
    /* Clear RTC interrupt request */
    CG_ClearINTReq(CG_INT_SRC_RTC);
```

割り込みが発生すると、RTC の秒の値の 1 の位の値を UART に出力します。
以下は、RTC データと時間値をどのように取得するかを示します。

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

7-14 SHA

7-14-1 例: SHA

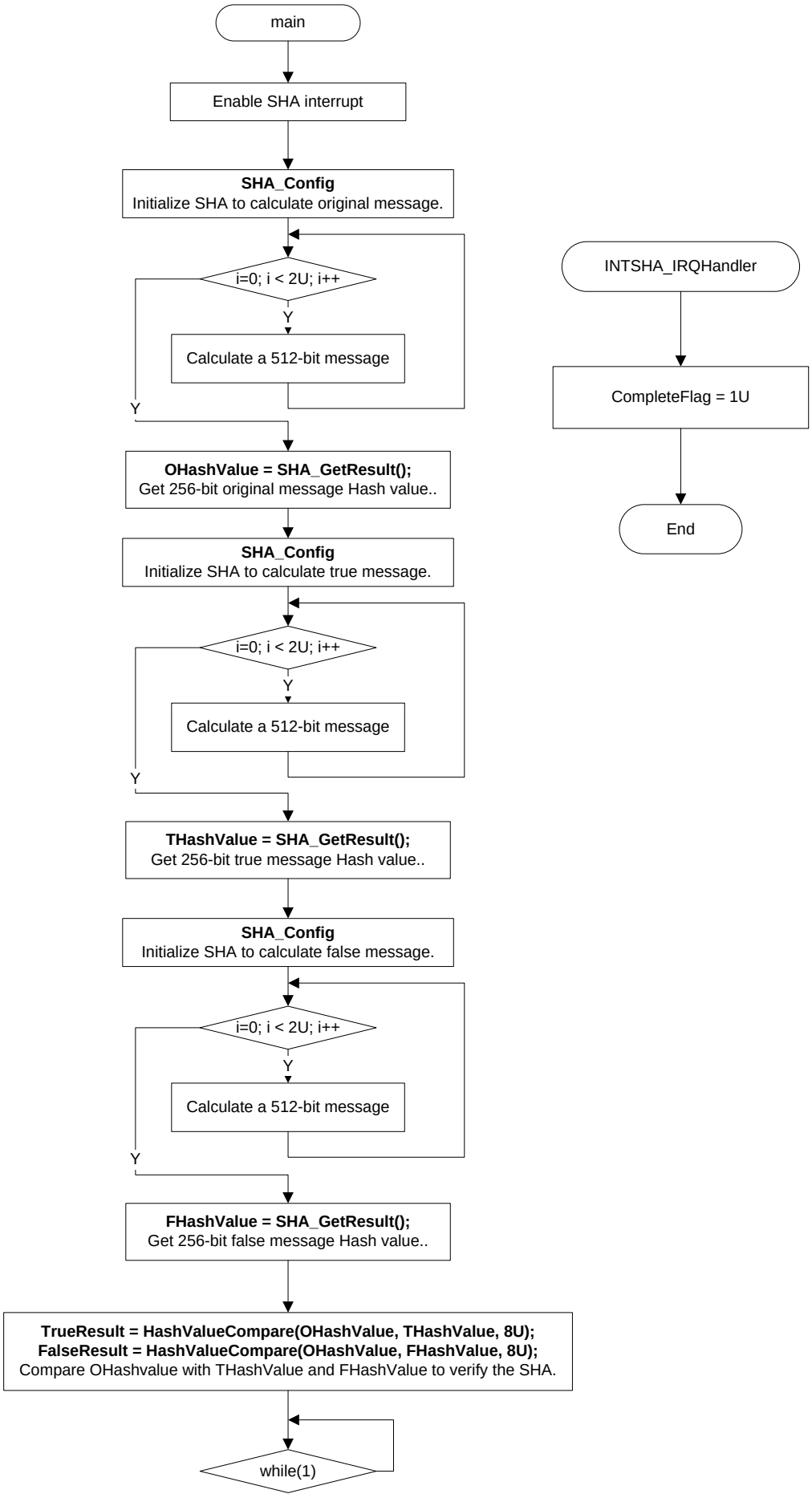
ペリフェラルドライバ(SHA, UART)を使用したサンプルプログラムです。

このサンプルプログラムでは以下を行います。

1. オリジナルメッセージのハッシュ値を演算するための設定を行います。
2. 演算を開始し、オリジナルメッセージのハッシュ値を取得します。
3. 正しいメッセージのハッシュ値を演算するための設定を行います。
4. 演算を開始し、正しいメッセージのハッシュ値を取得します。

5. 正しくないメッセージのハッシュ値を演算するための設定を行います。
6. 演算を開始し、正しくないメッセージのハッシュ値を取得します。
7. 正しいメッセージと正しくないメッセージをそれぞれオリジナルメッセージのハッシュ値と比較し、SHA 処理の正常動作を確認します。

- **フローチャート**


main
Enable SHA interrupt
SHA_Config
Initialize SHA to calculate original message.
i=0; i < 2U; i++
Y
Calculate a 512-bit message
Y
OHashValue = SHA_GetResult();
Get 256-bit original message Hash value..
SHA_Config
Initialize SHA to calculate true message.
i=0; i < 2U; i++
Y
Calculate a 512-bit message
Y
THashValue = SHA_GetResult();
Get 256-bit true message Hash value..
SHA_Config
Initialize SHA to calculate false message.
i=0; i < 2U; i++
Y
Calculate a 512-bit message
Y
FHashValue = SHA_GetResult();
Get 256-bit false message Hash value..
TrueResult = HashValueCompare(OHashValue, THashValue, 8U);
FalseResult = HashValueCompare(OHashValue, FHashValue, 8U);
Compare OHashvalue with THashValue and FHashValue to verify the SHA.
while(1)
INTSHA_IRQHandler
CompleteFlag = 1U
End

- **サンプルプログラムのコードと説明**

SHA 割り込みの設定を行います。

```
/* enable SHA calculation completion interrupt */
NVIC_ClearPendingIRQ(INTSHA_IRQn);
NVIC_EnableIRQ(INTSHA_IRQn);
__enable_irq();
```

SHA の初期設定を行います。

```
void SHA_Config(void)
{
    /* Confirm calculation completion */
    while(SHA_GetCalculationStatus() != SHA_CALCULATION_COMPLETE) {};

    /* The Hash value specified with the SHAINITx register */
    SHA_SetInitMode(SHA_INIT_VALUE_REG);

    /* An interrupt is output at each calculation */
    SHA_SetCalculationInt(SHA_INT_EACH_CALCULATION);

    /* Set the Hash initial value */
    SHA_SetInitValue(HashInit);

    /* Set the whole message length and unhandled message length */
    SHA_SetMsgLen(MsgLen);
    SHA_SetRmnMsgLen(RmnMsgLen);
}
```

SHA はオリジナルメッセージから 512 ビットのハッシュ値を生成します。

```
for (i = 0U; i < 2U; i++) {
    CompleteFlag = 0U;
    /* Set a 512-bit message. */
    SHA_SetMessage(OriginalMessage[i]);
    /* Start the SHA processor. */
    SHA_SetRunState(SHA_START);
    /* Wait for calculation completion */
    while (CompleteFlag != 1U) {
        /* Do nothing */
    }
}
```

256 ビットのオリジナルメッセージのハッシュ値を取得します。

```
/* Get 256-bit original message Hash value. */
SHA_GetResult(OHashValue);
```

SHA は正しいメッセージから 256 ビットのハッシュ値を生成します。

```
for (i = 0U; i < 2U; i++) {
    CompleteFlag = 0U;
    /* Set a 512-bit message. */
    SHA_SetMessage(TrueMessage[i]);
    /* Start the SHA processor. */
    SHA_SetRunState(SHA_START);
    /* Wait for calculation completion */
```

```
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
    }
```

正しいメッセージの 256 ビットのハッシュ値を取得します。

```
    /* Get 256-bit true message Hash value. */
    SHA_GetResult(THashValue);
```

SHA は正しくないメッセージから 256 ビットのハッシュ値を生成します。

```
    for (i = 0U; i < 2U; i++) {
        CompleteFlag = 0U;
        /* Set a 512-bit message. */
        SHA_SetMessage(FalseMessage[i]);
        /* Start the SHA processor. */
        SHA_SetRunState(SHA_START);
        /* Wait for calculation completion */
        while (CompleteFlag != 1U) {
            /* Do nothing */
        }
    }
```

正しくないメッセージの 256 ビットのハッシュ値を取得します。

```
    /* Get 256-bit false message Hash value. */
    SHA_GetResult(FHashValue);
```

SHA 処理の確認のため、OHashvalue と THashvalue、OHashvalue と FHashvalue をそれぞれ比較します。TrueResult = SUCCESS かつ FalseResult = ERROR の場合は比較結果に問題がないと判断します。

```
    /* Compare original message Hash value with true message Hash value. */
    TrueResult = HashValueCompare(OHashValue, THashValue, 8U);
    /* Compare original message Hash value with false message Hash value. */
    FalseResult = HashValueCompare(OHashValue, FHashValue, 8U);

    if ((TrueResult == SUCCESS) && (FalseResult == ERROR)) {
        common_uart_disp("SHA processed successfully !\n");
    } else {
        common_uart_disp("SHA processed with error !\n");
    }
```

7-15 SSP

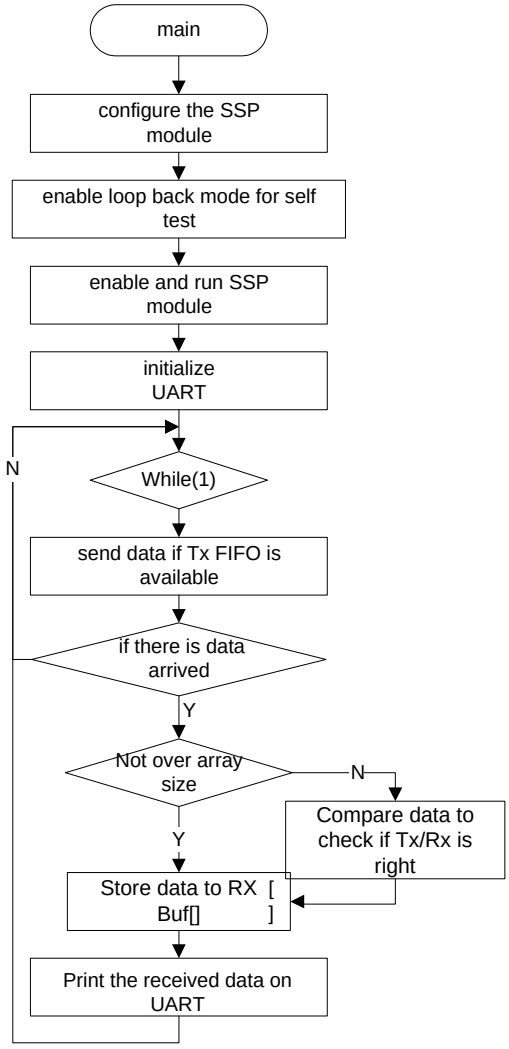
7-15-1 例: SSP0 セルフループバック

ペリフェラルドライバ (SSP, GPIO) のプログラムサンプルです。

この例では以下を行います。

1. SSP0 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**

SPI フレームフォーマットを選択します。

```
    /* Configure the SSP module */
    initSSP.FrameFormat = SSP_FORMAT_SPI;
```

最大または最小ビットレートを選択します。

```
    /* 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;
SSP_Init(TSB_SSP0, &initSSP);
```

SSP0 をループバックモードに設定します。

```
/* Enable loop back mode for self test */
SSP_SetLoopBackMode(TSB_SSP0, ENABLE);

/* Enable and run SSP module */
SSP_Enable(TSB_SSP0);
```

LED の設定を行います。

```
/* Initialize LEDs on M46B board before display something */
LED_Init();
hardware_init(UART_RETARGET);
```

whileループの中で、データ送信を行い、受信データが同じ場合はLED2とLED3を点灯し、異なる場合はLED0とLED1を点灯します。受信データはUART経由で確認することができます。

```
while (1) {

    datTx++;
    /* Send data if Tx FIFO is available */
    fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_TX);
    if ((fifoState == SSP_FIFO_EMPTY) || (fifoState == SSP_FIFO_NORMAL))
    {
        SSP_SetTxData(TSB_SSP0, 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(TSB_SSP0, SSP_RX);
    if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
        receive = SSP_GetRxData(TSB_SSP0);
        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 transmitted from Tx_Buf is received in Rx_Buf. */
            /* When the line "#define BITRATE_MIN" is commented, the SSP
               is run in maximum */
            /* 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);
                if (result   == NOT_SAME)
                {
                      LED_On(LED0);
                      LED_On(LED1);
                } else
                {
                      LED_On(LED2);
                      LED_On(LED3);
                }
            }

        } else {
            /* Do nothing */
        }

        sprintf((char *) SSP_RX_Data, “SSP RX DATA : %d", receive);
        common_uart_disp(SSP_RX _Data);
        common_uart_disp(“\n”);
    }
```

7-16　TMRB

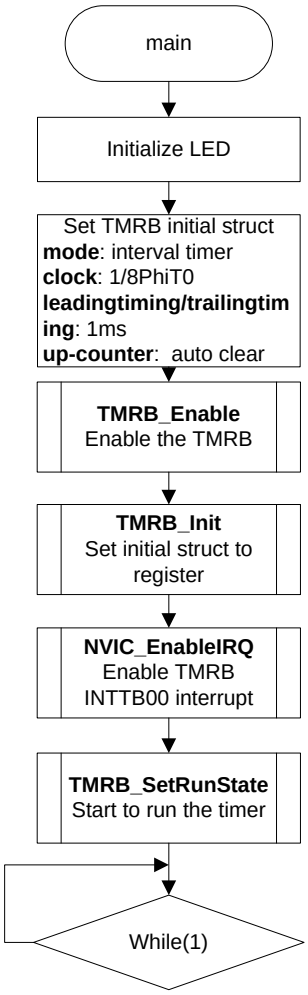
7-16-1　例: 汎用タイマ

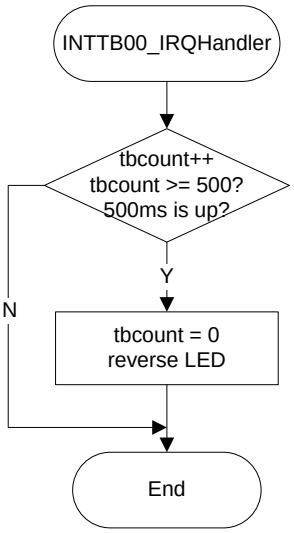
ペリフェラルドライバ (TMRB, GPIO) のプログラムサンプルです。

この例では以下を行います。

1. 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**

まず LED の初期化を来ない、LED を ON します。

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

TMRB 設定用の構造体を用意し TMRB モード、クロック、アップカウンタクリア方法、周期とデューティを設定します。このサンプルプログラムでは、1ms の周期とデューティを設定します。このカウント値は Tmrb_Calculator 関数で算出します。

```
TMRB_InitTypeDef m_tmrb;

m_tmrb.Mode = TMRB_INTERVAL_TIMER;       /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;          /* 1/8PhiT0 */
/* periodic time is 1ms(require 1000us) */
m_tmrb.TrailingTiming = Tmrb_Calculator(1000U, m_tmrb.ClkDiv);
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;      /* up-counter auto clear */
/* periodic time is 1ms(require 1000us) */
```

```
m_tmrb.LeadingTiming = Tmrb_Calculator(1000U, m_tmrb.ClkDiv);
```

TMRB 動作の許可、および初期化を行います。INTTB0 割り込み(1ms 毎にトリガ)を許可し、最後に TMRB を動作します。

```
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*/
```

while(1)内にて割り込み発生を待ちます。
割り込みハンドラ内でカウントアップし、500ms までカウントすると 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(LED_ALL);
    } else {
        LED_On(LED_ALL);
    }
} else {
    /* Do nothing */
}
```

Tmrb_Calculator()関数 :

```
uint16_t Tmrb_Calculator(uint16_t Tmrb_Require_us, uint32_t ClkDiv)
{
    uint32_t T0 = 0U;
    const uint16_t Div[8U] = {1U, 2U, 8U, 32U, 64U, 128U, 256U, 512U};

    SystemCoreClockUpdate();

    T0 = SystemCoreClock / (1U << ((TSB_CG->SYSCR >> 8U) & 7U));
    T0 = T0/((Div[ClkDiv])*1000000U);

    return(Tmrb_Require_us * T0);
}
```

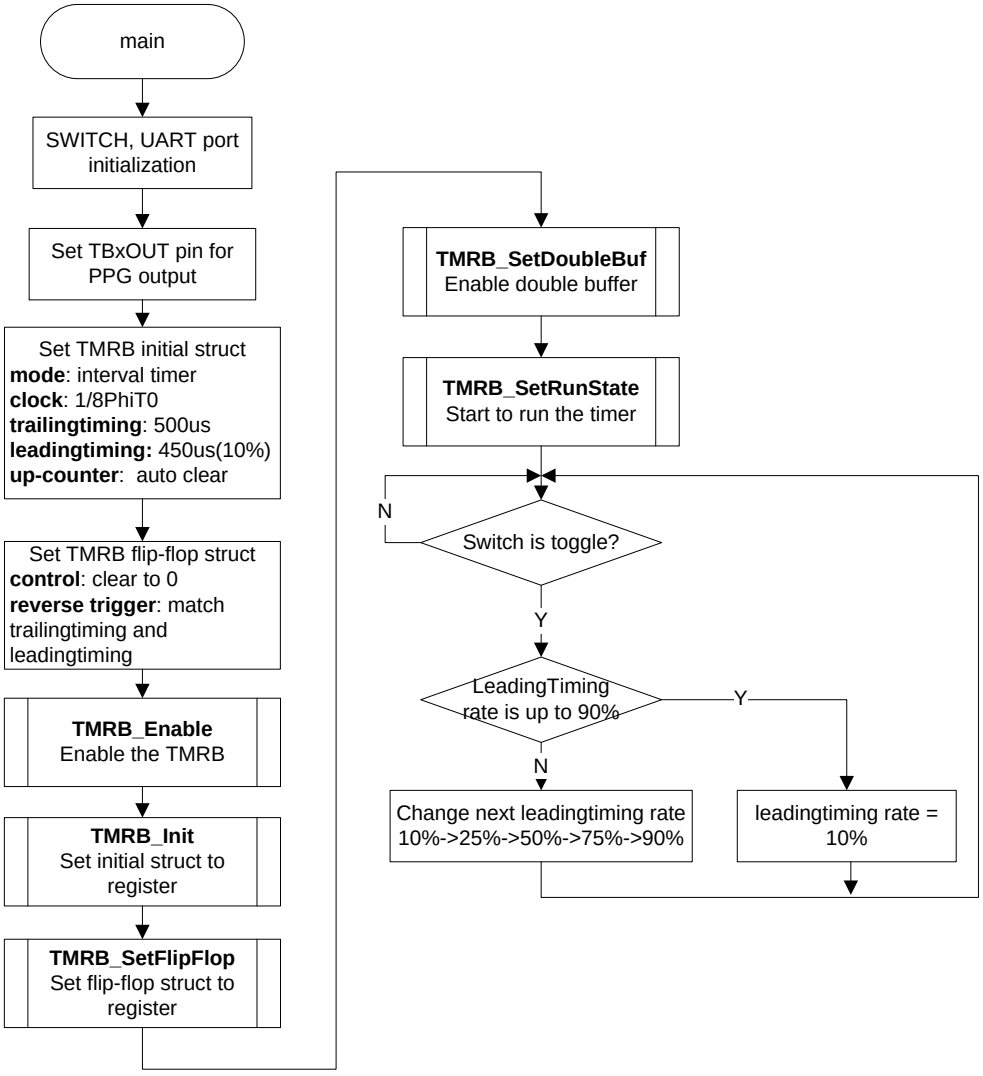
7-16-2　例: PPG 出力

ペリフェラルドライバ (TMRB, GPIO, UART) 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. TMRB6 の初期化
2. PPG 動作の設定
3. PPG 波形の調整

- **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、配列 tgtLeadingTiming、LeadingTimingus、LeadingTiming を初期化し、 PF1 を PPG 出力の TB6OUT に設定します。

```
    TMRB_InitTypeDef m_tmrb;
    TMRB_FFOutputTypeDef PPGFFInital;
    uint8_t keyvalue;
    uint32_t i = 0U;
    uint32_t tgtLeadingTiming[5U] = { 10U, 25U, 50U, 75U, 90U }; /* leadingtiming:
10%, 25%, 50%, 75%, 90% */
    uint32_t LeadingTimingus[5U] = {0U, 0U, 0U, 0U, 0U};
    uint32_t LeadingTiming[5U] = {0U, 0U, 0U, 0U, 0U};

    /* LeadingTimingus: 50, 125, 250, 375, 450 */
    for (i=0U;i<=4U;i++) {
        LeadingTimingus[i] = tgtLeadingTiming[i] * 5U;
```

```
    }

    /* UART & switch initialization */
    hardware_init(UART_RETARGET);
    SW_Init();

    /* Set PK1 as TB6OUT for PPG output */
    GPIO_SetOutput(GPIO_PK, GPIO_BIT_1);
    GPIO_EnableFuncReg(GPIO_PK, GPIO_FUNC_REG_4, GPIO_BIT_1);
```

TMRB 初期化構造体を用意し、TMRB モード、クロック、アップカウンタクリア方法周期、デューティを設定します。本サンプルプログラムでは、500μs の周期とデューティを設定します。周期とデューティの設定値は Tmrb_Calculator()関数で算出します。

```
    TMRB_InitTypeDef m_tmrb;

    m_tmrb.Mode = TMRB_INTERVAL_TIMER;        /* internal timer */
    m_tmrb.ClkDiv = TMRB_CLK_DIV_8;                /* 1/8PhiT0 */
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
    for(i=0U;i<=4U;i++) {
        LeadingTiming[i] = Tmrb_Calculator(LeadingTimingus[i], m_tmrb.ClkDiv);
    }
    m_tmrb.TrailingTiming  =  Tmrb_Calculator(500U,  m_tmrb.ClkDiv);        /*
trailingtiming is 500us */
    m_tmrb.LeadingTiming  =  LeadingTiming[Rate];                           /*
leadingtiming, initial value 10% */
```

フリップフロップ初期化構造体、フリップフロップ制御、反転トリガ引数を設定します。反転トリガは、デューティとサイクルと一致するよう設定します。

```
    PPGFFInital.FlipflopCtrl = TMRB_FLIPFLOP_SET;
    PPGFFInital.FlipflopReverseTrg=TMRB_FLIPFLOP_MATCH_TRAILING|
    TMRB_FLIPFLOP_MATCH_LEADING;
```

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_TB6);
    TMRB_Init(TSB_TB6, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB6, &PPGFFInital);
    /* enable double buffer */
    TMRB_SetDoubleBuf(TSB_TB6,ENABLE, TMRB_WRITE_REG_SEPARATE);
    TMRB_SetRunState(TSB_TB6, TMRB_RUN);
```

スイッチが Low から High になるまで待ち、同時に UART に現在のデューティを表示します。

```
    do {                              /* wait if switch is Low */
            keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_0);
            LeadingTiming_display(); /* display current leadingtiming */
       } while (GPIO_BIT_VALUE_0 == keyvalue);
```

スイッチが High になると、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 90% から、また 10% になります。

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

    TMRB_ChangeLeadingTiming(TSB_TB6,  LeadingTiming[Rate]);     /*  change
leadingtiming rate */
```

Tmrb_Calculator 関数:

```
    uint16_t Tmrb_Calculator(uint16_t Tmrb_Require_us, uint32_t ClkDiv)
    {
        uint32_t T0 = 0U;
        const uint16_t Div[8U] = {1U, 2U, 8U, 32U, 64U, 128U, 256U, 512U};

        SystemCoreClockUpdate();

        T0 = SystemCoreClock / (1U << ((TSB_CG->SYSCR >> 8U) & 7U));
        T0 = T0/((Div[ClkDiv])*1000000U);

        return(Tmrb_Require_us * T0);
    }
```

7-17 SIO/UART

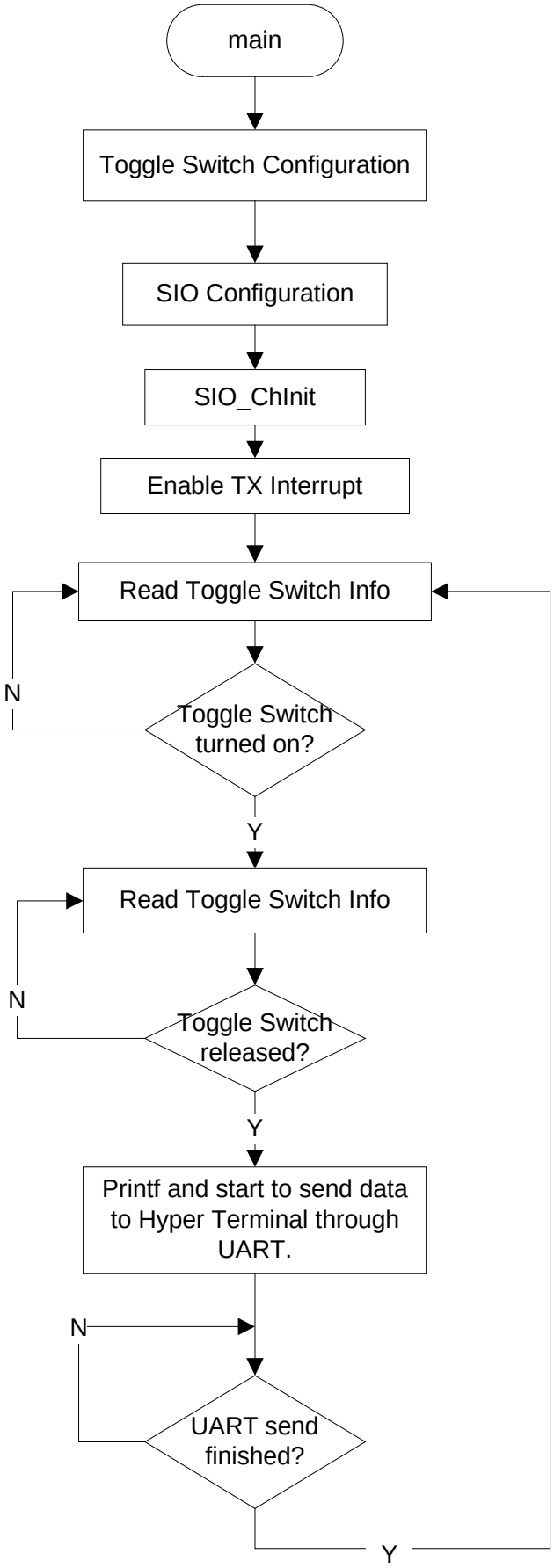
7-17-1 例: リターゲット(UART)

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:**


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

- **サンプルプログラムのコードと説明**

まず、GPIO 設定と UART の初期化を行います。
GPIO ペリフェラルドライバを使い、GPIO を UART に設定します。

```
    GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_2, ENABLE);
    GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_2, DISABLE);
    GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_2);
```

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)
```

送信データは TxBuffer に文字列として確認できます。

```
    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;
}
```

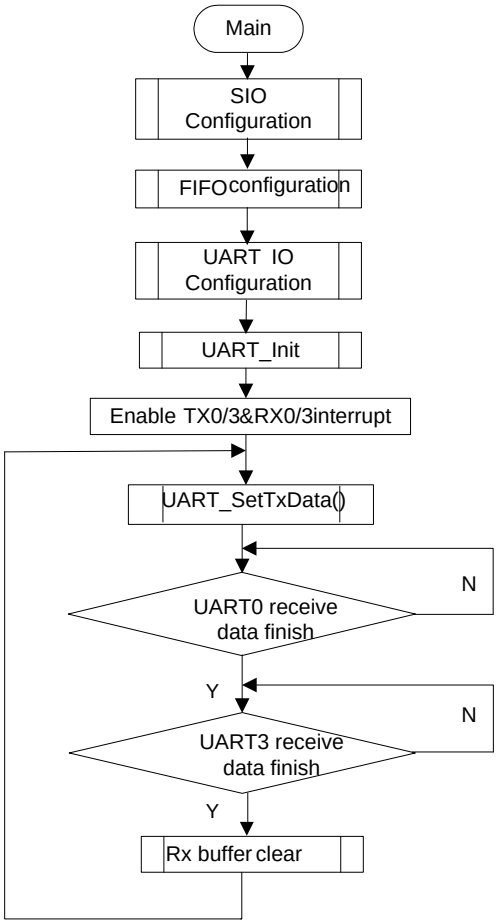
7-17-2　例: UART FIFO

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**

まず GPIO と UART の初期設定を行います。
GPIO を UART0 と UART3 に設定します。

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
	if (SCx == TSB_SC0) {
		GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_2, ENABLE);
		GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_2, DISABLE);
		GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_2);
		GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_1, DISABLE);
		GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_1, ENABLE);
		GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_1);
	} else if (SCx == TSB_SC1) {
		GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_5, ENABLE);
		GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_5, DISABLE);
		GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_5);
		GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_6, DISABLE);
		GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_6, ENABLE);
		GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_6);
	} else if (SCx == TSB_SC2) {
		GPIO_SetOutputEnableReg(GPIO_PL, GPIO_BIT_2, ENABLE);
		GPIO_SetInputEnableReg(GPIO_PL, GPIO_BIT_2, DISABLE);
		GPIO_EnableFuncReg(GPIO_PL, GPIO_FUNC_REG_5, GPIO_BIT_2);
		GPIO_SetOutputEnableReg(GPIO_PL, GPIO_BIT_1, DISABLE);
```

```
            GPIO_SetInputEnableReg(GPIO_PL, GPIO_BIT_1, ENABLE);
            GPIO_EnableFuncReg(GPIO_PL, GPIO_FUNC_REG_5, GPIO_BIT_1);
        } else if (SCx == TSB_SC3) {
            GPIO_SetOutputEnableReg(GPIO_PB, GPIO_BIT_0, ENABLE);
            GPIO_SetInputEnableReg(GPIO_PB, GPIO_BIT_0, DISABLE);
            GPIO_EnableFuncReg(GPIO_PB, GPIO_FUNC_REG_3, GPIO_BIT_0);
            GPIO_SetOutputEnableReg(GPIO_PB, GPIO_BIT_1, DISABLE);
            GPIO_SetInputEnableReg(GPIO_PB, GPIO_BIT_1, ENABLE);
            GPIO_EnableFuncReg(GPIO_PB, GPIO_FUNC_REG_3, GPIO_BIT_1);
        }
}
```

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

```
    UART_InitTypeDef myUART;

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

UART0/3 の有効化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 の初期設定を行います。

```
    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_RXFIFO4B_FLEVLE_4_2B);
    UART_RxFIFOFillLevel(UART3, 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_TxFIFOFillLevel(UART0, UART_TXFIFO4B_FLEVLE_0_0B);
    UART_TxFIFOFillLevel(UART3, 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 の送信割り込みを有効にします。

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

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

UART0 の送信割り込みルーチン:

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

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

UART3 の送信割り込みルーチン:

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

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

UART0 の受信割り込みルーチン:

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

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

UART3 の受信割り込みルーチン:

```
void INTRX3_IRQHandler(void)
```

```
{
	volatile UART_Err err;

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

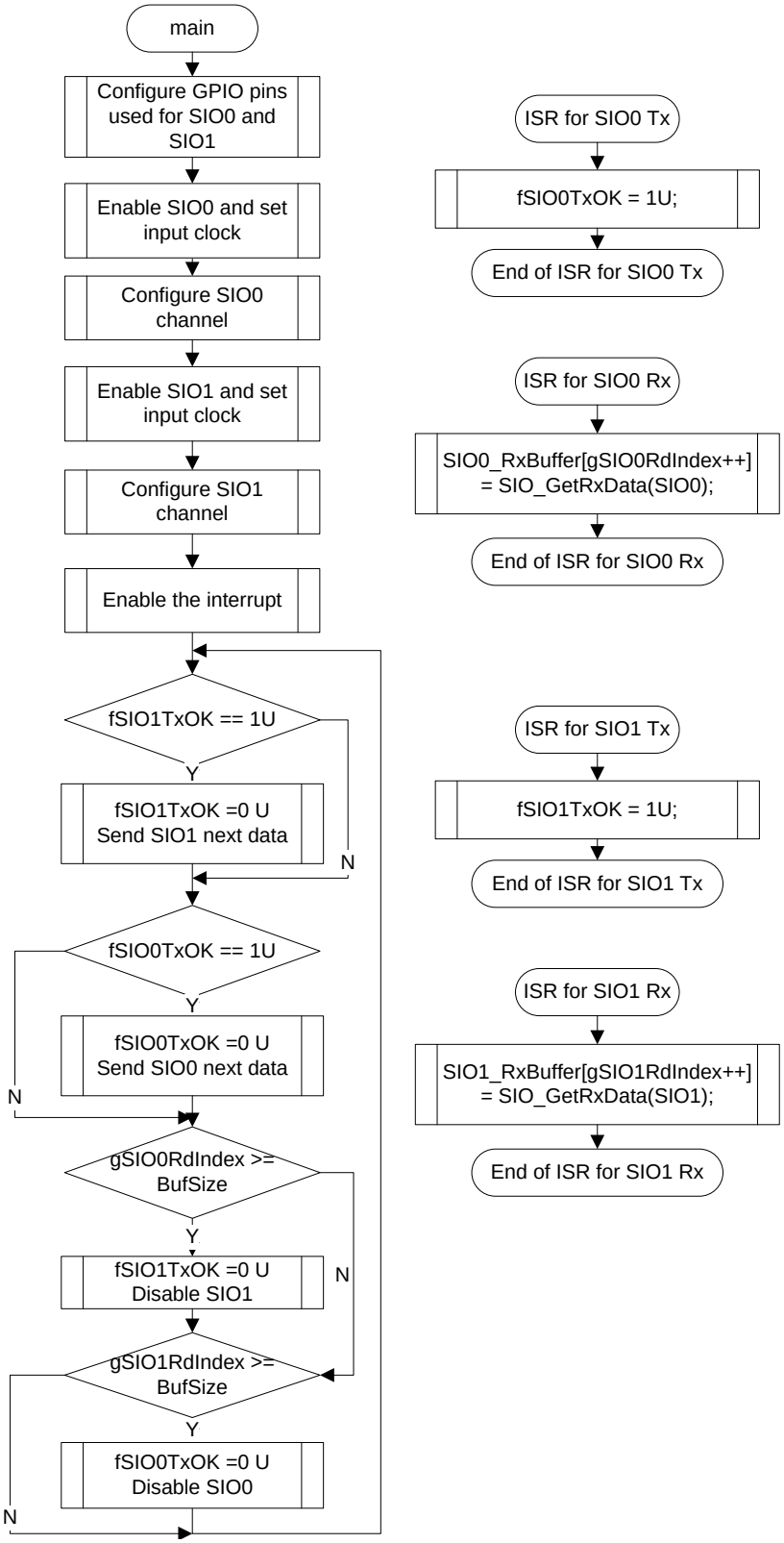
7-17-3　例: SIO

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 と SIO1 間のデータ転送
3. SIO の送受信割り込み

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず GPIO 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.TIDLE = SIO_TIDLE_HIGH;
    SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_8;
    SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
    SIO0_Init.TransferDir = SIO_LSB_FRIST;
    SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
    SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
    SIO0_Init.BaudRateClock = SIO_BR_CLOCK_TS2;
    SIO0_Init.Divider = SIO_BR_DIVIDER_2;

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

次に 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.TIDLE = SIO_TIDLE_HIGH;
    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;
    SIO1_Init.TXDEMP = SIO_TXDEMP_HIGH;
    SIO1_Init.EHOLDTime = SIO_EHOLD_FC_64;

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

SIO 送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);
```

すべて基本的な設定を行った後、データ送信を行います。

```
   while (1) {
            /* SIO1 send data from TXD1*/
            if (fSIO1TxOK == 1U) {
                fSIO1TxOK = 0U;
                SIO_SetTxData(SIO1, SIO1_TxBuffer[gSIO1WrIndex++]);
            } else {
                /*Do Nothing */
            }
            /* SIO0 send data from TXD0*/
            if (fSIO0TxOK == 1U) {
                fSIO0TxOK = 0U;
                SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
            } else {
```

```
                /*Do Nothing */
            }

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

            /* Print receive buffer */
            if ((gSIO0RdIndex == BufSize) && (gSIO1RdIndex == BufSize)) {
#ifdef DEBUG
                printf((char *)SIO0_RxBuffer);
                printf((char *)SIO1_RxBuffer);
#endif
        }
```

SIO0 送信の割り込みハンドラにおいて、転送完了フラグをセットします。

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

SIO0 受信の割り込みハンドラにおいて、受信バッファからデータを取得します。

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

SIO1 送信の割り込みハンドラにおいて、転送完了フラグをセットします。

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

SIO1 受信の割り込みハンドラにおいて、受信バッファからデータを取得します。

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

7-18 uDMAC

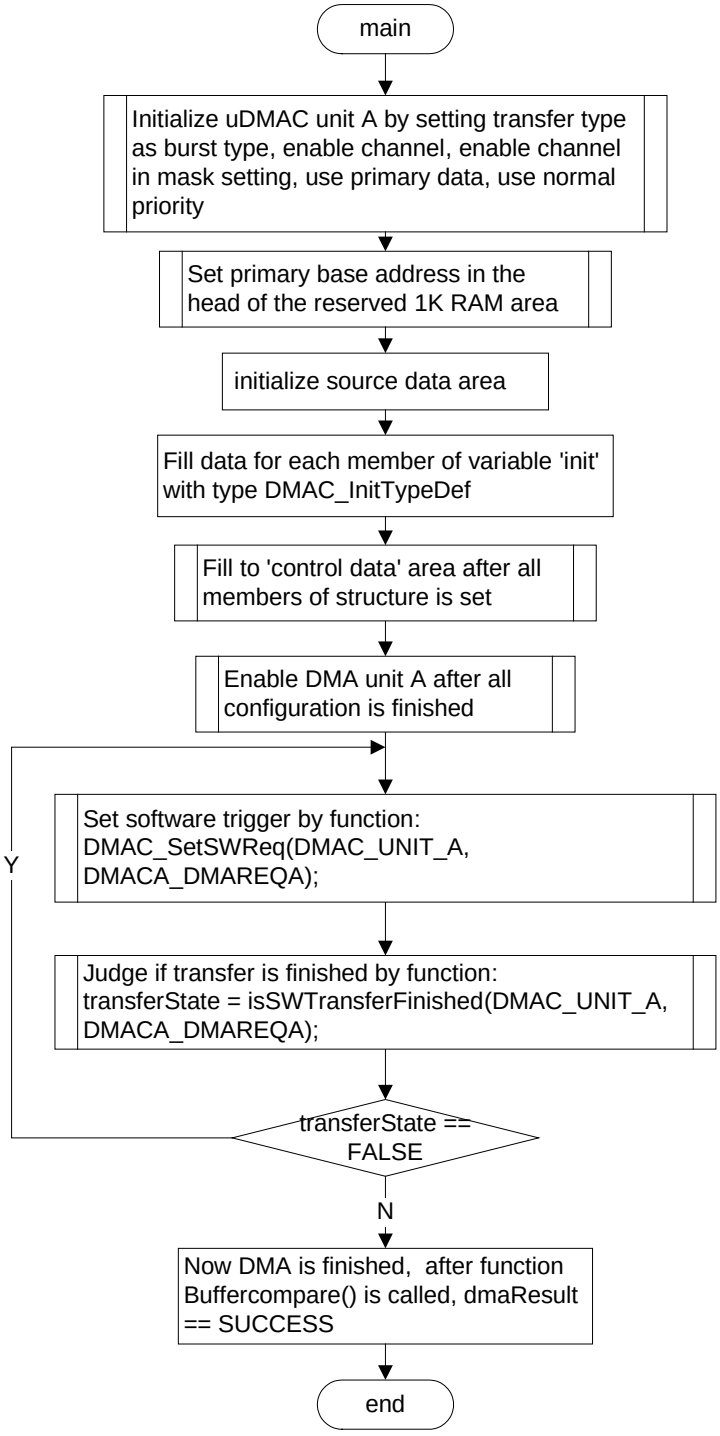
7-18-1 例: メモリ→メモリの DMA 転送

ペリフェラルドライバ(μDMAC、GPIO) を使用したサンプルプログラムです。
補足: LED0~LED3 は PB0~PB3 と接続します。

この例では以下を行います。

1. 1K RAM 領域を μDMA 制御データ用に確保
2. μDMA 設定と初期化
3. ソフトウェアトリガによるメモリ間 DMA 転送開始

- **フローチャート:**



- **サンプルプログラムのコードと説明**

μDMAC は構成データを保存するため、1K バイトの RAM 領域を必要とします。本領域は、ベースアドレスのビット 0～9 を 0 にして、アプリケーションコード中に確保しなければなりません。例えば、0x20000400 は OK ですが、0x20000300 は使用できません。

IAR EWARM と Keil MDK(RealView)における、RAM 領域を確保するための例です。

```
#if defined (   __ICCARM__   )     /* IAR EWARM */
/* For TMPM46BF10 uDMAC_CFG_A/B are defined in file
```

```
TMPM46BF10_Flash_For_uDMAC.icf */
uint32_t uDMAC_A_Control_Data[256U] @ ".uDMAC_CFG_A" ;
uint32_t uDMAC_B_Control_Data[256U] @ ".uDMAC_CFG_B" ;

#elif defined ( __CC_ARM   )       /* Keil MDK */
#include <absacc.h>
#define uDMAC_CFG_A         (0x20000400U)
#define uDMAC_CFG_B         (uDMAC_CFG_A + 0x400U)
uint32_t uDMAC_A_Control_Data[256U] __at (uDMAC_CFG_A);
uint32_t uDMAC_B_Control_Data[256U] __at (uDMAC_CFG_B);
#endif
```

Keil MDK では、キーワード '__at'が使用されますが、IAR EWARM では十分ではありません。リンク設定ファイル(.lcf) が、下記内容に変更する必要があります。(UNIT A のみ). 詳細は、"TMPM46B_Flash_For_uDMAC.icf"ファイルを参照してください。

```
    /* reserve 1K RAM for uDMAC configuration */
    define symbol uDMAC_RAM_START_A = 0x20000400;
    define symbol uDMAC_RAM_END_A    = 0x200007FF;

    define region uDMAC_CFG_RAM_A   = mem:[from uDMAC_RAM_START_A to
uDMAC_RAM_END_A];

    place in   uDMAC_CFG_RAM_A    { readwrite section .uDMAC_CFG_A };
```

RAM を確保した後、転送タイプとしてバーストタイプを設定し、チャネルをイネーブル、マスク設定のチャネルをイネーブル、初期データを使用、優先度通常を使用してください。

```
        DMACA_SetTransferType(DMACA_DMAREQA, DMAC_BURST);
        DMAC_SetChannel(DMAC_UNIT_A, DMACA_DMAREQA, ENABLE);
        DMAC_SetMask(DMAC_UNIT_A, DMACA_DMAREQA, ENABLE);
        DMAC_SetPrimaryAlt(DMAC_UNIT_A,                DMACA_DMAREQA,
DMAC_PRIMARY);
        DMAC_SetChannelPriority(DMAC_UNIT_A, DMACA_DMAREQA,
                                        DMAC_PRIOTIRY_NORMAL);
```

確保した1K バイトの RAM 領域の先頭に初期ベースアドレスを設定してください。

```
        DMAC_SetPrimaryBaseAddr(DMAC_UNIT_A,
                                    (uint32_t)&uDMAC_A_Control_Data);
```

転送する送信元のデータ領域を初期化してください。

```
        for(idx = 0U; idx < TX_NUMBERS; idx ++ ) {
            src[idx] = idx;
        }
```

"DMAC_InitTypeDef"タイプの変数'init'のメンバ設定を開始します。
送信元と送信先の最終アドレスを設定します。

```
        tmpAddr = (uint32_t)&src;
        init.SrcEndPointer = tmpAddr + ((TX_NUMBERS - 1U) * sizeof(src[0U]));
        tmpAddr = (uint32_t)&dst;
        init.DstEndPointer = tmpAddr + ((TX_NUMBERS - 1U) * sizeof(dst[0U])) ;
```

BASIC あるいは AUTOMATIC モードを選択します。

```
#if defined(DMA_DEMOMODE_BASIC )
init.Mode = DMAC_BASIC;
#elif defined(DMA_DEMOMODE_AUTOMATIC)
init.Mode = DMAC_AUTOMATIC;
#endif
```

その他のメンバを設定します。

```
init.NextUseBurst = DMAC_NEXT_NOT_USE_BURST;
init.TxNum = TX_NUMBERS;
init.ArbitrationMoment = DMAC_AFTER_32_TX;

/* now both src and dst are use uint16_t type which is 2bytes long */
init.SrcWidth = DMAC_HALF_WORD;
init.SrcInc = DMAC_INC_2B;
init.DstWidth = DMAC_HALF_WORD;
init.DstInc = DMAC_INC_2B;
```

全ての構成メンバを設定した後、'control data' 領域を入力します。

```
DMAC_FillInitData(DMAC_UNIT_A, DMACA_DMAREQA, &init);
```

全設定が終了した後、DMA ユニット A をイネーブルにします。

```
DMAC_Enable(DMAC_UNIT_A);
```

ソフトウェアトリガを設定します。転送が完了したかどうかを確認します。

```
do{
   /* Because of "init.Mode = DMAC_BASIC" above, here need to trigger it
until transfer is finished, */
   /* If DMAC_AUTOMATIC is used, only need to trigger it once */
   DMAC_SetSWReq(DMAC_UNIT_A, DMACA_DMAREQA);

   transferState = isSWTransferFinished(DMAC_UNIT_A,
DMACA_DMAREQA);

}while ( transferState == false );
```

転送が完了している場合、関数 Buffercompare()を呼び出し dmaResult が SUCCESS であるかを確認してください。

```
dmaResult = ERROR;
dmaResult = Buffercompare( src, dst, TX_NUMBERS);
if ( dmaResult  == SUCCESS ) {
    LED_On(LED0);
    LED_On(LED1);
} else {
    LED_On(LED2);
    LED_On(LED3);
}
```

7-19 WDT

7-19-1 例:WDT

ペリフェラルドライバ (WDT, GPIO) のプログラムサンプルです。

この例では以下を行います。

1. 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 LED0 を点滅させます。

- **サンプルプログラムのコードと説明**

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

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

WDT を初期化し、その後 WDT を有効にします。

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

DEMO1 では、NMI 割り込みの発生を待ちます。

```
while(1)
{
}
```

DEMO1 では、NMI 割り込み発生時に WDT を禁止にし、LED1 の点滅を停止します。

```
WDT_Disable();
```

DEMO2 では、WDT クリアを行い、常に LED0 を点滅させます。

```
WDT_WriteClearCode();
```

補足:
LED0 は PB0 と接続し、LED1 は PB1 と接続します。