TOSHIBA

TX03 ペリフェラルドライバ使用例 (TMPM361/362/363/364)

第一版

2017 年 9 月

東芝デバイス&ストレージ株式会社

CMDR-M364UE-01J

本製品取り扱い上のお願い

- ソフトウエア使用権許諾契約書の同意無しに使用しないで下さい。

目次

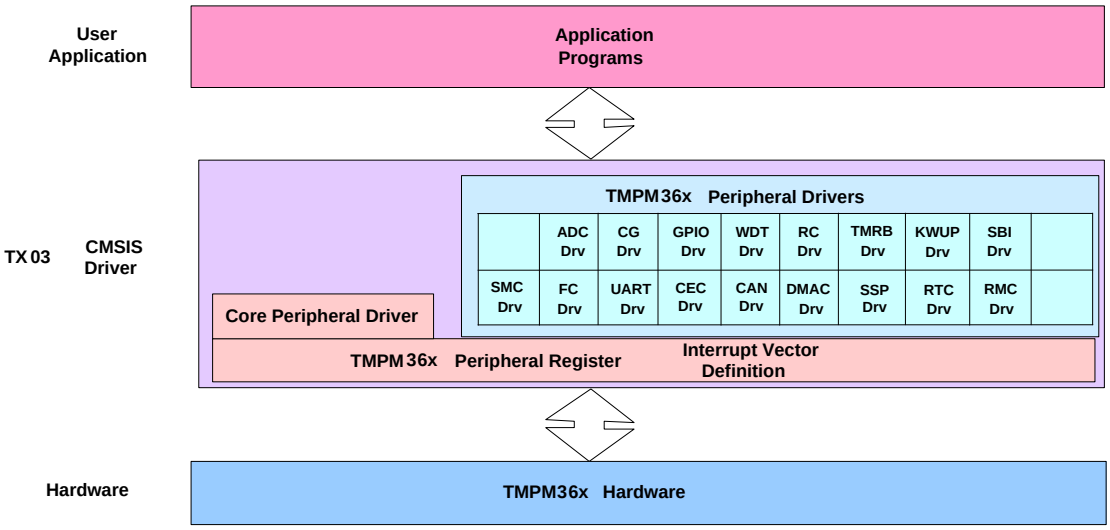
1 はしがき

本サンプルプログラムは、東芝製マイコンTMPM36x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。

※本ドキュメントでは、TMPM361/362/363/364 を表す部分を"TMPM36x"と表現します。

2 概要

TX03ペリフェラルドライバを下記のように使用します。



3 使用する機能

機能	チャネル	使用/未使用
CG	クロックギア	使用
	PLL	PLL4 逓倍
スタンバイモード	-	使用 (SLEEP モード)
SysTick	-	未使用
ウォッチドッグタイマ (WDT)	-	使用（WDT サンプル）
外部割込み(INT)	INTTB0	使用 TMRB サンプル
	INTTX11	使用 SIO/UART サンプル
	INTCANRX	使用 CAN サンプル
	INTCECRX	使用 CEC サンプル
	INTCECTX	使用 CEC サンプル

機能	チャネル	使用/未使用
	INTDMACTC0	使用 DMAC サンプル
	INTRMCRX0	使用 RMC サンプル
	INTRTC	使用 RTC サンプル
	INTSBI2	使用 SBI サンプル
	上記以外	未使用
シリアルチャネル(SIO/UART)	SIO11/UART11	使用(UART サンプル)
	上記以外	未使用
CAN	-	使用(CAN サンプル)
CEC	-	使用(CEC サンプル)
16 ビットタイマ/イベントカウンタ(TMRB)	TMRB0	使用(TMRB サンプル)
	上記以外	未使用
12ビットA/Dコンバータ	AIN0	使用(ADC サンプル)
	上記以外	未使用
DMA コントローラ	-	使用(DMAC サンプル)
リアルタイムクロック(RTC)	-	使用(RTC サンプル)
シリアルバスインタフェース(SBI)	SBI2	使用(SBI サンプル)
	上記以外	未使用
同期式シリアルインタフェース(SSP)	SSP0	使用(SSP0 サンプル)
リモコン判定機能(RMC)	RMC0	使用(RMC サンプル)
	上記以外	未使用
キーオンウェイクアップ(KWUP)	-	使用(KWUP サンプル) 未使用
RC	-	使用(RC サンプル)
スタティックメモリコントローラ(SMC)	-	使用(SMC 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TMPM36x用評価ボードを用いてテストされています。
以下に、端子用途を説明します。

No				Name	Usage
M361	M362	M363	M364		
	1		1	PK6, AIN14	Unused
	2		2	PK7, AIN15	Unused
1	3	1	3	AVSS	AD converter GND
2	4	2	4	VREFH	Supplying the AD converter with a reference power supply
3	5	3	5	RESETn	Reset input
4	6	4	6	MODE	Unused

5	7	5	7	PL0, SDA0, TB0OUT	Unused
6	8	6	8	PL1, SCL0, TB1OUT	Unused
7	9	7	9	PL2, SCK0, TB2OUT	Unused
8	10	8	10	PL3, INT0, TB3OUT	Unused
9	11	9	11	PL4, TXD1, TB4OUT	Unused
10	12	10	12	PL5, RXD1, TB5OUT	Unused
11	13	11	13	PL6, SCLK1, TB6OUT, CTS1n	Unused
12	14	12	14	PL7, INT1, TB7OUT	Unused
13	15	13	15	DVSS	GND
14	16	14	16	PM0, SCLK2, TB1IN0, CTS2n	Unused
15	17	15	17	PM1, TXD2, TB1IN1	Unused
16	18	16	18	PM2, RXD2, ALARMn	Unused
17	19	17	19	PM3, INT2, TB3OUT	Unused
18	20	18	20	PM4, SCLK3, CTS3n	LED1
19	21	19	21	PM5, TXD3	LED2
20	22	20	22	PM6, RXD3	LED3
21	23	21	23	PM7, INT3	LED4
22	24	22	24	PN0, TXD4	Unused
23	25	23	25	PN1, RXD4	Unused
24	26	24	26	PN2, SCLK4, TB2IN0, CTS4n	Unused
25	27	25	27	PN3, INT4, TB2IN1, RMIN0	RMC RX
	28		28	PN4, TXD5	Unused
	29		29	PN5, RXD5	Unused
	30		30	PN6, SCLK5, TBFIN0, CTS5n	Unused
	31		31	PN7, INT8, TBFIN1, RMIN1	Unused
	32		32	PO0, TXD6, TB8OUT	Unused
	33		33	PO1, RXD6, TB9OUT	Unused
	34		34	PO2, SCLK6, TBAOUT, CTS6n	Unused
	35		35	PO3, INT9, TBBOUT	Unused
	36		36	PO4, TXD7, TBCOUT	Unused
	37		37	PO5, RXD7, TBDOUT	Unused
	38		38	PO6, SCLK7, TBEOUT, CTS7n	Unused
	39		39	PO7, INTA, TBFOUT	Unused
26	40	26	40	PP0, CS2n	Unused
27	41	27	41	PP1	Unused
28	42	28	42	PP2, BLS0n, SPDO	SPP SPDO
29	43	29	43	PP3, BLS1n, SPDI	SPP SPDI
30	44	30	44	PP4, WEn, SPCLK	SPP SPCLK
31	45	31	45	PP5, OEn, SPFSS	SPP SPFSS
32	46	32	46	PP6, ALEn	Unused
33	47	33	47	DVDD3B	+3.3V
34	48	34	48	DVSS	GND

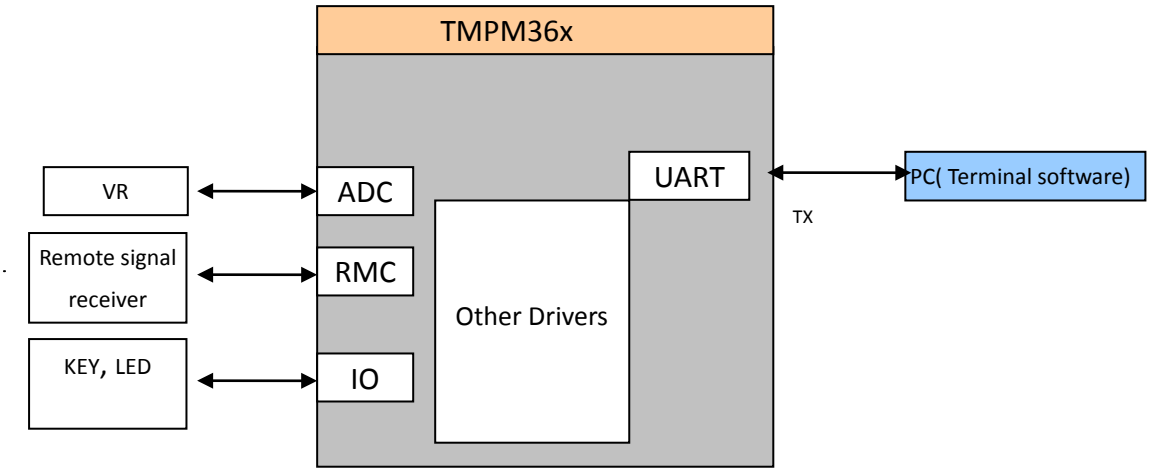
35	49	35	49	PA0, D0, AD0	Unused
36	50	36	50	PA1, D1, AD1	Unused
37	51	37	51	PA2, D2, AD2	Unused
38	52	38	52	PA3, D3, AD3	Unused
39	53	39	53	PA4, D4, AD4	Unused
40	54	40	54	PA5, D5, AD5	Unused
41	55	41	55	PA6, D6, AD6	Unused
42	56	42	56	PA7, D7, AD7	Unused
43	57	43	57	PB0, D8, AD8	Unused
44	58	44	58	PB1, D9, AD9	Unused
45	59	45	59	PB2, D10, AD10	Unused
46	60	46	60	PB3, D11, AD11	Unused
47	61	47	61	PB4, D12, AD12	Unused
48	62	48	62	PB5, D13, AD13	Unused
49	63	49	63	PB6, D14, AD14	Unused
50	64	50	64	PB7, D15, AD15	Unused
	65		65	PC0, A1, TXD8	Unused
	66		66	PC1, A2, RXD8	Unused
	67		67	PC2, A3, SCLK8, CTS8n	Unused
	68		68	PC3, A4	Unused
	69		69	PC4, A5, TXD9	Unused
	70		70	PC5, A6, RXD9	Unused
	71		71	PC6, A7, SCLK9, CTS9n	Unused
	72		72	PC7, A8	Unused
	73		73	PD0, A9, TXD10	Unused
	74		74	PD1, A10, RXD10	Unused
	75		75	PD2, A11, SCLK10, CTS10n	Unused
	76		76	PD3, A12	Unused
	77		77	PD4, A13, TXD11	UART TX
	78		78	PD5, A14, RXD11	UART RX
	79		79	PD6, A15, SCLK11, CTS11n	Unused
	80		80	PD7, A16, INTB	Unused
51	81	51	81	PE0, A17, TB5IN0	Unused
52	82	52	82	PE1, A18, TB5IN1	Unused
53	83	53	83	PE2, A19, TB6IN0	Unused
54	84	54	84	PE3, A20, TB6IN1	Unused
55	85	55	85	PE4, A21, TXD0, CTXD	CAN TX
56	86	56	86	PE5, A22, RXD0, CRXD	CAN RX
57	87	57	87	PE6, A23, SCLK0, CTS0n	Unused
58	88	58	88	PE7, INT5, SCOUT	Unused
59	89	59	89	DVDD3B	+3.3V
60	90	60	90	DVSS	GND

61	91	61	91	SWDIO	Unused
62	92	62	92	SWCLK	Unused
63	93	63	93	PF0, TRACECLK	Unused
64	94	64	94	PF1, TRACEDATA0, SWV	Unused
65	95	65	95	PF2, TRACEDATA1	Unused
66	96	66	96	PF3, TRACEDATA2	Unused
67	97	67	97	PF4, TRACEDATA3	Unused
68	98	68	98	PG0, SDA1, TB7IN0	Unused
69	99	69	99	PG1, SCL1, TB7IN1	Unused
70	100	70	100	PG2, SCK1, CS0n	Unused
71	101	71	101	PG3, INT6, CS1n	Unused
72	102	72	102	PG4, SDA2, TB9IN0	SBI master TX
73	103	73	103	PG5, SCL2, TB9IN1	SBI master CLK
74	104	74	104	PG6, SCK2, USBPON, CS3n	Unused
75	105	75	105	PG7, INT7, USBOC, WDTOUTn	Unused
	106		106	PH0, SDA3, TBAIN0	Unused
	107		107	PH1, SCL3, TBAIN1	Unused
	108		108	PH2, SCK3, TBBIN0	Unused
	109		109	PH3, INTC, TBBIN1	Unused
	110		110	PH4, SDA4, TBDIN0	Unused
	111		111	PH5, SCL4, TBDIN1	Unused
	112		112	PH6, SCK4, TBEIN0	Unused
	113		113	PH7, INTD, TBEIN1	Unused
76	114	76	114	RVDD3	+3.3V
77	115	77	115	XT1	Connect to low-speed oscillator
78	116	78	116	XT2	Connect to low-speed oscillator
79	117	79	117	DVDD3A	+3.3V
80	118	80	118	X1	Connect to high-speed oscillator
81	119	81	119	DVSS	GND
82	120	82	120	X2	Connect to high-speed oscillator
83	121	83	121	DVDD3B	+3.3V
84	122	84	122	DVSS	GND
85	123	85	123	D+	Unused
86	124	86	124	D-	Unused
87	125	87	125	NMIn	Unused
88	126	88	126	TEST1	Unused
89	127	89	127	TEST2	Unused
90	128	90	128	PI0, BOOTn	Unused

91	129	91	129	PI1, CEC	CEC
92	130	92	130	AVDD3	Supplying AD converter with power supply
93	131	93	131	PJ0, AIN0	ADC IN
94	132	94	132	PJ1, AIN1	Unused
95	133	95	133	PJ2, AIN2	Unused
96	134	96	134	PJ3, AIN3, ADTRGn	Unused
97	135	97	135	PJ4, AIN4, KWUP0	KEY1
98	136	98	136	PJ5, AIN5, KWUP1	KEY2
99	137	99	137	PJ6, AIN6, KWUP2	KEY3
100	138	100	138	PJ7, AIN7, KWUP3	KEY4
	139		139	PK0, AIN8	Unused
	140		140	PK1, AIN9	Unused
	141		141	PK2, AIN10	Unused
	142		142	PK3, AIN11	Unused
	143		143	PK4, AIN12	Unused
	144		144	PK5, AIN13	Unused

5 開発環境

下記に開発環境の構成を示します。



1. ハードウェア:
   TMPM364 用評価ボード(非売品)+TMPM36xF10 MCU (x=1, 2, 3, 4)

2. 開発ツール:
   - IAR:
     1) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
     2) IDE:IAR Embedded workbench 5.5 version
   - KEIL:

1) U-Link: RealView ULINK2
2) IDE: KEIL uVision 4.10

6 機能

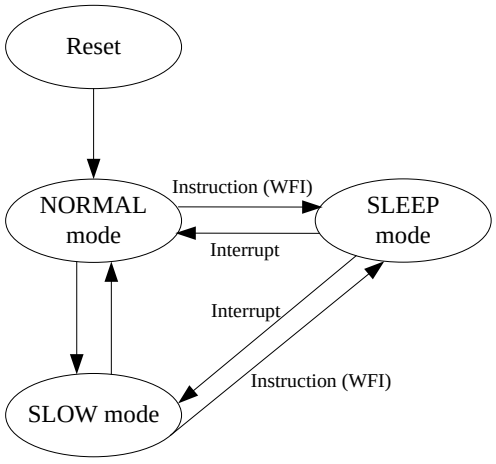
6-1 動作モード選択

本デバイスは 6 つの動作モードがあります: NORMAL, SLOW, IDLE2/1, SLEEP, STOP

- **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。
- **SLOW モード:**
  高速クロックを停止させ、CPU コア，周辺ハードウェアを低速クロックで動作させるモードです。NORMAL モードに比べ消費電力を低減することができます。SLOW モードでは動作可能な周辺機能が限られます。使用できる周辺機能は、I/O ポート(PORT)、タイマ(TMRB)、リアルタイムクロック(RTC), CEC 機能(CEC)、リモコン判定機能(RMC)、キーオンウェイクアップ機能(KWUP)です。
- **SLEEP モード:**
  内部低速発振器とRTC, CEC 機能，リモコン判定機能が動作します。
  SLEEP モードが解除されると、SLEEP モードへ移行する直前の動作モードへ復帰し、動作を開始します。
  SLEEPモードの解除は、RTC割り込み、CEC割り込み、RMC割り込み、外部割り込みによって行うことができます。

IDLE2/1, SLEEP, STOP の各モードは低消費電力モードです。
低消費電力モードへ移行するには、システ制御レジスタ 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: SLEEP 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

PJ6/AIN0 に接続されたポテンションメータの値を測定します。

6-3 CAN

本サンプルプログラムは、CAN API のテストループバックモードにおけるメールボックスメッセージ設定、メールボックス設定、メッセージ送受信、モード変更（テストループバックモード）、ステータス、割り込みなどを行います。

動作シーケンス:

1. PE4 と PE5 を接続します。
2. 電源を投入します。LED PM4～PM7 が OFF します。
3. PJ4 を ON すると、CAN テストループバックモードと ADC 動作が開始します。
4. AD コンバータはポテンショメーターAIN0 をサンプリングし、サンプル値を読み出します。
5. メールボックス 0 (送信メールボックス) に値を書き込みます。メールボックス 10 (受信メールボックス) を有効にします。
6. メールボックス 10 が 値を取得後、受信データを LCD に表示します。
7. LCD 値はポテンショメーターAON0 を回転させると、変化します。

6-4 CEC

DVDプレイヤーに接続されたCEC端子からデータを受信します。スタートビットと8ビットのデータをUART出力します。
サンプルプログラムがCECメッセージデータをCECラインに送信し、同時にスタートビットと8ビットのデータをUART出力します。

動作シーケンス:

1. DVD プレイヤーから CEC メッセージを受信します。
2. CEC メッセージデータを UART 出力します。
3. DVD プレイヤーから"Active_Source"(CEC コマンド)を受信すると(DVD プレイヤーの電源を ON するとこのコマンドが送信されます)、DVD プレイヤーへ"Standby"コマンドを送信します。
4. DVDプレイヤーはスタンバイ状態へ移行します。

6-5 CG

CPU の動作モードを変更します。NORMAL と SLEEP の 2 モードのみサポートします。

6-6 DMAC

このサンプルプログラムは、ソフトウェアトリガにより RAM の"SRC_BUFFER"から"DST_BUFFER"へのデータ転送を行います。

動作シーケンス:

1 転送タイプとして、バーストタイプを選択し、DMAC を初期化します。
2 ソフトウェアトリガを設定します。
3 すべて設定後、DMAC 動作を開始します。
4 転送終了後、送信元と送信先のデータを比較します。

6-7 FLASH

このサンプルプログラムは、FC レジスタのリード／ライトを行います。

6-8 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。

6-9 KWUP

KWUP と GPIO のドライバを使用して、スイッチ 1(KEY1)を ON すると、CPU は低消費電力モードとなり(LED は消灯します)、スイッチ 0(KEY0)を ON すると、低消費電力モードが解除されるサンプルプログラムです。

6-10 RC

このサンプルプログラムは、RC レジスタのリード／ライトを行います。

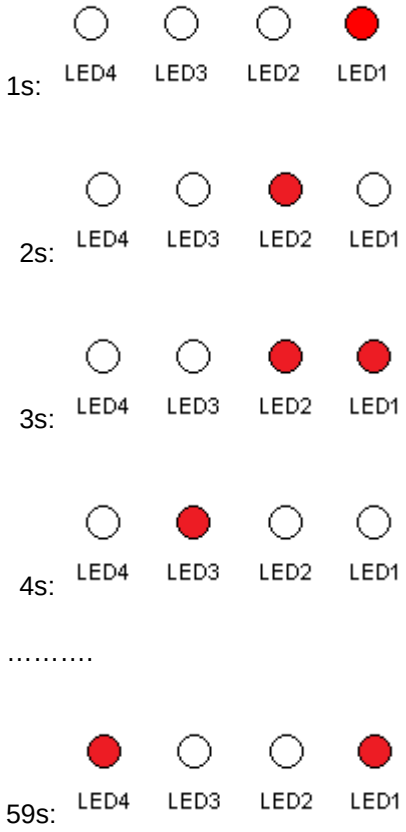
6-11 RMC

このサンプルプログラムはリモコン信号の受信を行い、デコードします。
デコードされたデータは、UART 出力にて確認できます。カスタムコードまたはアドレスコードは、HEX 形式で表示されます。

6-12 RTC

このサンプルプログラムは、RTC を利用して LED に秒表示を行います。

LED 表示例:



6-13 SBI

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。
I2C 割り込みを使用して SBI/I2C バスのリード/ライトを行います。

I2C スレーブ(I2C)は 2 バイトデータを SBI2 から EEPROM(アドレスは 0xA0 です)へライトし、I2C スレーブ(I2C)は 2 バイトデータを EEPROM から SBI2 へリードします。
受信結果はデバッガにて RAM 変数 gI2C_RxData[]にて確認できます。

補足:本サンプルソフトを実行するためには、評価ボードの SBI2 と M330 IAR スタータキットボードの EEPROM が必要です。

6-14 SMC

このサンプルプログラムは、SRAM の初期化を行います。

6-15 SSP

このサンプルプログラムは、SSP の API を使用したループバック処理を行います。
データを送信し、その後、受信データを確認します。受信データが送信したものと同一かそうでないかの確認結果を LED に表示します。

6-16 TMRB

このサンプルプログラムは、MCU のタイマを使って、汎用タイマ処理を行います。
時間周期は 1ms です。

6-17 SIO/UART

6-17-1 UART 出力

このサンプルプログラムは、UART 出力を行います。

6-17-2 UART FIFO

このサンプルプログラムは、UART0 から"TMPM3611"というデータを FIFO を使用した UART1 へ送信します。また同時に UART0 は"TMPM3612"というデータを FIFO を使用した UART1 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=”TMPM3612”となり、RxBuffer1="TMPM3611"となると設定したブレークポイントで停止します。

6-17-3 SIO

このサンプルプログラムは、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)

6-18 WDT

このサンプルプログラムは、ウォッチドッグタイマ API を使用して NMI 割り込みの発生を行います。

7 ソフトウェア

本ソフトウェアは、TMPM364 評価ボードを使用し TMPM36xF10 MCU (x=1, 2, 3, 4) 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:

```
\---TMPM36x
    +---Libraries
    |   +---TX03_CMSIS
    |   |   |   system_TMPM36x.c
    |   |   |   system_TMPM36x.h
    |   |   |   TMPM36x.h
    |   |   \---startup
    |   |       +---arm
    |   |       |       startup_TMPM36x.s
    |   |       \---iar
    |   |               startup_TMPM36x.s
    |   \---TX03_Periph_Driver
    |       +---inc
    |       |       tmpm36x_adc.h
    |       |           :
    |       |       tmpm36x_wdt.h
    |       |       tx03_common.h
    |       \---src
    |               tmpm36x_adc.c
    |                   :
    |               tmpm365_wdt.c
    \---Project
        +---Examples            //only 1 examples listed here
        |   +---ADC
        |   |   \---ADC_Data_Read
        |   |       +---App
        |   |       |       main.c
        |   |       +---IAR
        |   |       |       ADC_Data_Read.ewd
        |   |       |       ADC_Data_Read.ewp
        |   |       |       ADC_Data_Read.eww
        |   |       \---KEIL
        |   |               ADC_Data_Read.uvopt
```

```
|   |                   ADC_Data_Read.uvproj
|   \---Workspace
|       +---IAR
|       |       Examples_for_M36x_Driver.eww
|       \---KEIL
|                   Examples_for_M36x_Driver.uvmpw
\---Template
    +---IAR
    |       TMPM36x_flash.icf
    \---KEIL
            tmpm36x.sct
```

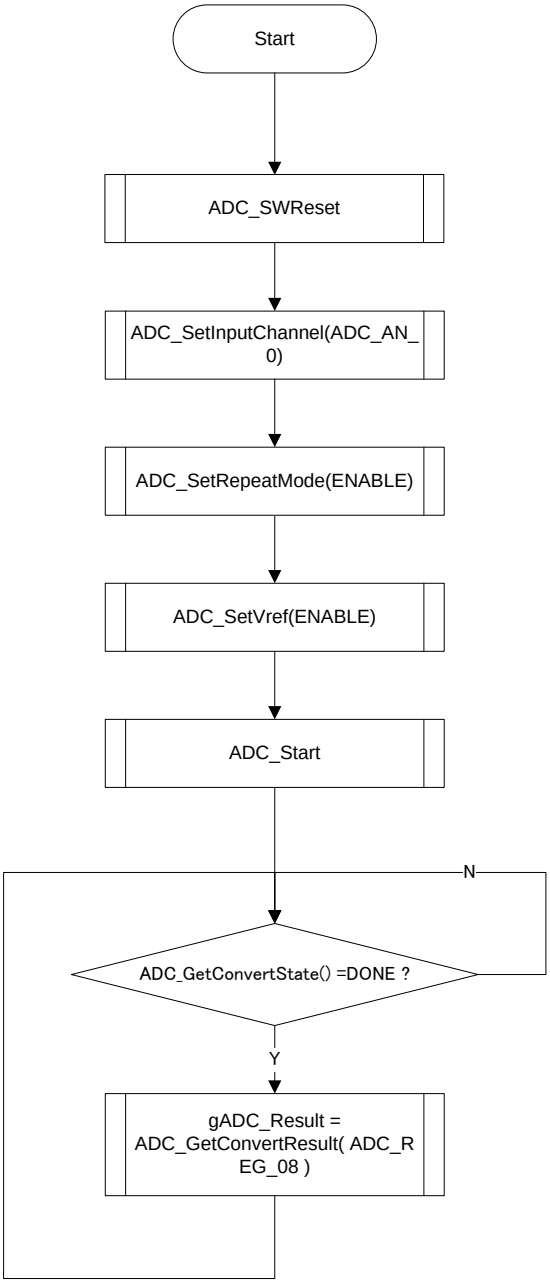
7-1 ADC

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。

このサンプルでは以下を行います。

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に AD コンバータユニットを初期化し、リピートモードの許可、ADC Vref を ON します。

```
ADC_SetInputChannel(ADC_AN_0);
ADC_SetRepeatMode(ENABLE);
ADC_SetVref(ENABLE);
```

ADC を開始します。

```
ADC_Start();
```

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ます。

```
gADC_state = ADC_GetConvertState();
if (DONE == gADC_state) {
    gADC_state = BUSY;
    gADC_Result = ADC_GetConvertResult(ADC_REG_08);
}
```

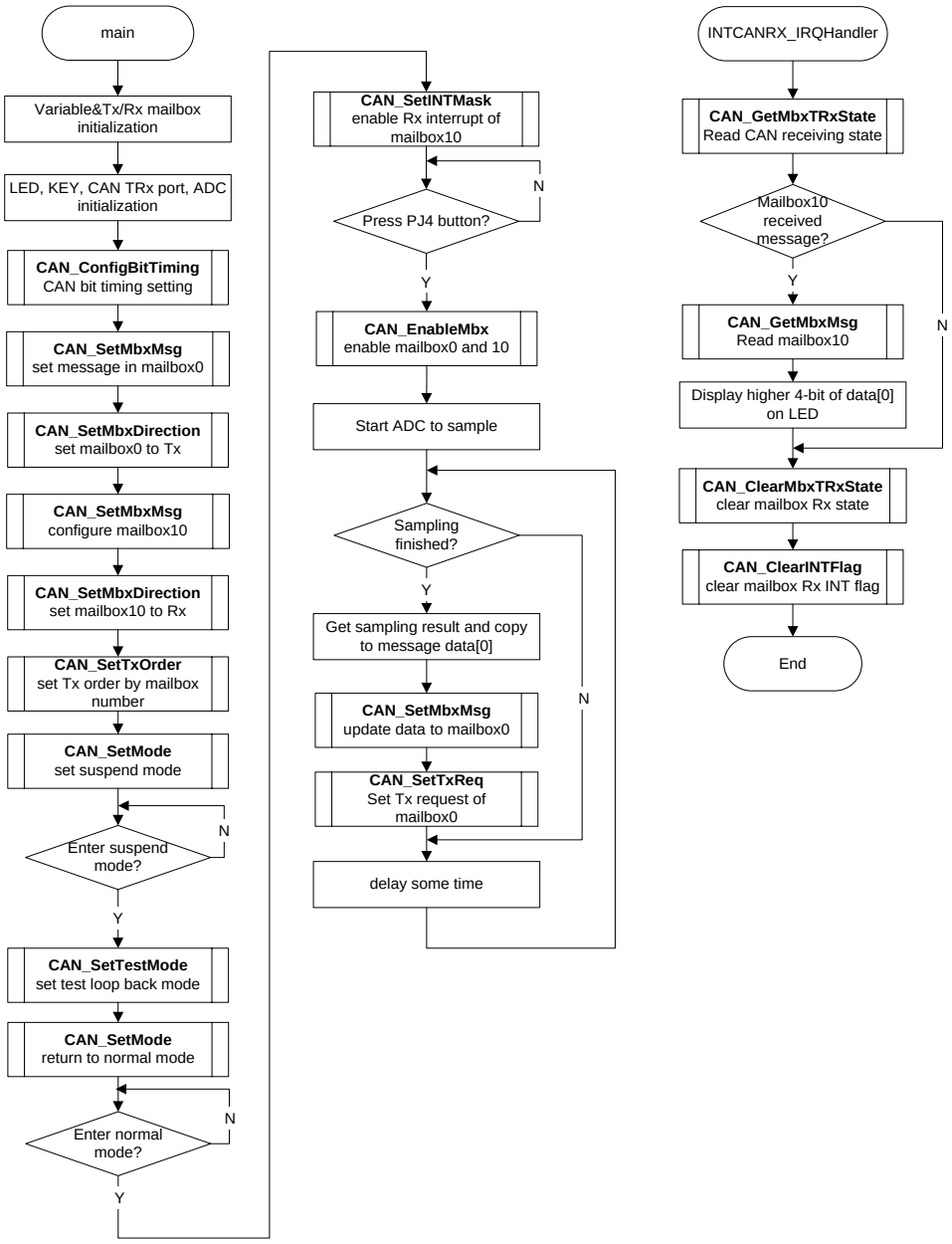
7-2 CAN

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

このサンプルでは以下を行います。

1. CAN メールボックス設定、ビットタイミング設定
2. CAN モード：サスペンド、コンフィグレーション、ノーマルモード、テストループバックモード
3. CAN 送信/受信方法
4. CAN モジュールステータス、割り込み使用

- **フローチャート**



- **サンプルプログラムのコードと説明**

CAN グローバルステート、割り込みマスク、 ADC ステート、 ADC 結果、 CAN ビットタイミングパラメータ、送受信メッセージなど、サンプルプログラムで使用されるすべての変数を初期化します。

```
CAN_GlobalState globalstate;
CAN_INTFactor intmask;
WorkState adc_state = BUSY;
ADC_ResultTypeDef adc_value;
CAN_BitTimingTypeDef bittiming ={ BAUDRATE_PRESCALER,   //Brp = 1023
                                  CAN_TIMING_TSEG1_4TQ, //TSEG1 = 4TQ
                                  CAN_TIMING_TSEG2_4TQ, //TSEG2 = 4TQ
                                  CAN_SINGLE_SAMPLING, //Single sampling
                                  CAN_TIMING_SJW_1TQ };   //SJW = 1TQ
CAN_MsgTypeDef msgtx = { 0U };
```

```
    CAN_MsgTypeDef msgrx = { 0U };
```

評価ボードの LED, KEY, CAN TX/RX 端子, ADC を初期化します。

```
    LED_Configuration();
    KEY_Configuration();
    CAN_PortInit();
    ADC_Init();
```

CAN_ConfigBitTiming()関数をコールし、CAN ビットタイミングを設定します。ボーレート計算式は下記のとおりです。
BR = fosc/((TSEG1+TSEG2+3)*(BRP+1))
たとえば、fosc = 12MHz, TSEG1 = 6 (7TQ), TSEG2 = 3 (4TQ), BRP = 1 の時、ボーレートは BR = 12/((6 + 3 + 3) * (1 + 1)) = 500Kbit/s になります。ビットタイミング設定コンフィグレーションモードでのみ行えます。リセット後に、デフォルトモードは、コンフィグレーションモードになります。

```
    /* CAN bit timing setting */
    CAN_ConfigBitTiming(&bittiming);
```

メールボックス 0 メッセージ ID, データ長を送信メールボックスとして設定します。

```
    /* CAN Tx mailbox0 setting */
    msgtx.MsgID = MSG_ID;
    msgtx.MsgDataLen = 2U;          /* data length: 2 byte */
    CAN_SetMbxMsg(CAN_MBX_0, &msgtx);
    CAN_SetMbxDirection(CAN_MBX_0, CAN_MBX_TX); /* mailbox0: Tx */
```

メールボックス 10 を受信メールボックスとして設定します。メッセージ ID はメールボックス 0 と同じです。

```
    /* CAN Rx mailbox10 setting */
    msgrx.MsgID = msgtx.MsgID;    /* ID of mailbox10 = ID of mailbox0 */
    CAN_SetMbxMsg(CAN_MBX_10, &msgrx);
    CAN_SetMbxDirection(CAN_MBX_10, CAN_MBX_RX); /* mailbox10: Rx */
```

CAN_SetTxOrder()関数をコールし、メールボックス送信順序を設定します。メールボックス番号による順序の場合、数が少ない順に送信されます。ID 順の場合、ID の優先順位の高い順に送信されます。

```
    CAN_SetTxOrder(CAN_TX_ORDER_MBX_NUM);/* Order: by mailbox number */
```

次に、テストループバックモードの設定準備を行いますが、まずテストループバックモードの有効/無効設定をサスペンドモードで行います。API CAN_SetMode()関数をコールし CAN オペレーションモードを設定変更します。設定後、指定されたグローバルステートアクノリッジビットがセットされるのを待ちます。API CAN_GetGlobalState()関数をコールし、グローバル状態を読み出します。

```
    /* Enter suspend mode */
    CAN_SetMode(CAN_SUSPEND_MODE);
    while (globalstate.Bit.SuspendModeAck == 0U) {
        globalstate = CAN_GetGlobalState();
    }
```

サスペンドモードに入ると、API CAN_SetTestMode()関数をコールし、設定します。テストループバックモードでは、CAN は自分が送信したメッセージを受信し、アクノリッジを発生します。他の CAN ノードは必要ありません。

```
    /* Set test loop back mode */
    CAN_SetTestMode(CAN_TEST_LOOP_BACK_MODE, ENABLE);
```

ノーマルモードに戻ります。CAN は、テストループバックモードが有効の状態で、ノーマル状態になります。API CAN_GetGlobalState()関数コールして確認した後、CAN がノーマルモ

ードになるのを待ちます。

```
/* Return normal mode */
CAN_SetMode(CAN_NORMAL_MODE);
while (globalstate.Bit.SuspendModeAck == 1U) {
    globalstate = CAN_GetGlobalState();
}
```

メールボックス 10 受信割り込みを API CAN_SetINTMask()関数をコールして、有効にします。指定されたメールボックスビットが設定されると、割り込みが有効になります。

```
intmask.MbxBit.Mbx10 = 1U;
CAN_SetINTMask(CAN_INT_MBX_MASK, &intmask);
NVIC_EnableIRQ(INTCANRX_IRQn);
```

API CAN_EnableMbx()関数をコールしてメールボックス 0 とメールボックス 10 を有効にします。また、サンプリング用の ADC モジュールも有効にします。

```
/* Enable mailbox0 and 10 */
CAN_EnableMbx(CAN_MBX_0 | CAN_MBX_10);
/* ADC start to run */
ADC_Start();
```

ADC モジュールがデッドループにないか確認します。サンプリングが終了している場合、サンプリング値を読み出し、メッセージデータ [0] に 8 ビットを入れ、API CAN_SetMbxMsg()関数を用いてメールボックス 0 のデータを更新します。次に、API CAN_SetTxReq()関数を用いてメールボックス 0 のデータを更新します。この手順を繰り返し、CAN バスで ADC サンプル値を送信します。
fRcvCAN フラッグを待ちます。API CAN_GetMbxMsg()関数をコールしてメッセージの読み出しを行い、LCD への表示に使用します。

```
while (1) {
    adc_state = ADC_GetConvertState();
    if (DONE == adc_state) {            /* ADC convertion finished */
        adc_state = BUSY;
        adc_value = ADC_GetConvertResult(ADC_REG_08);
        msgtx.MsgData[0] = (uint8_t) adc_value.ADCResultValue;

        CAN_SetMbxMsg(CAN_MBX_0, &msgtx);/* Put the result into mailbox0 */
        CAN_SetTxReq(CAN_MBX_0, ENABLE); /* Start to send */
    } else {
        /* Do nothing */
    }
    delay(50000U);
}
```

メールボックス 10 がメールボックス 0 の送信メッセージを受信すると、 CAN RX 割り込みにトリガーがかかります。IRQ INTCANRX_IRQHandler() で、API CAN_GetMbxTRxState()関数をコールして受信状態を確認します。メールボックスがメッセージを受信すると、指定ビットが設定されます。
その後、 API CAN_GetMbxMsg()関数をコールしてメッセージを読み込み、LED PM4～7 のメッセージデータ[0] の上位 4 ビットを表示します。

```
void INTCANRX_IRQHandler(void)
{
  CAN_MbxState mbxstate = { 0U };
  CAN_MsgTypeDef msgrx = { 0U };
  mbxstate = CAN_GetMbxTRxState(CAN_STATE_ID_RX_ACK); /* Rx state */
  if (mbxstate.Bit.Mbx10 == 1U) {       /* If mailbox10 has received a message */
    CAN_GetMbxMsg(CAN_MBX_10, &msgrx);      /* Read mailbox10 */
    LED_Display(msgrx.MsgData[0] >> 4U);/* Get the data, displays on LED*/
                                            /* PM4~7 */
```

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

最後に、CAN API の API CAN_ClearMbxTRxState()関数をコールしてメールボックスの受信フラグをクリア、API CAN_ClearINTFlag()関数をコールしてメールボックスの十便割り込みフラグをクリアします。それぞれのフラグは別々にクリアします。

```
        CAN_ClearMbxTRxState(CAN_STATE_ID_RX_ACK);    /* Clear the Rx state */
        CAN_ClearINTFlag(CAN_INT_TYPE_RX);    /* Clear the Rx interrupt flag */
    }
```

7-3 CEC

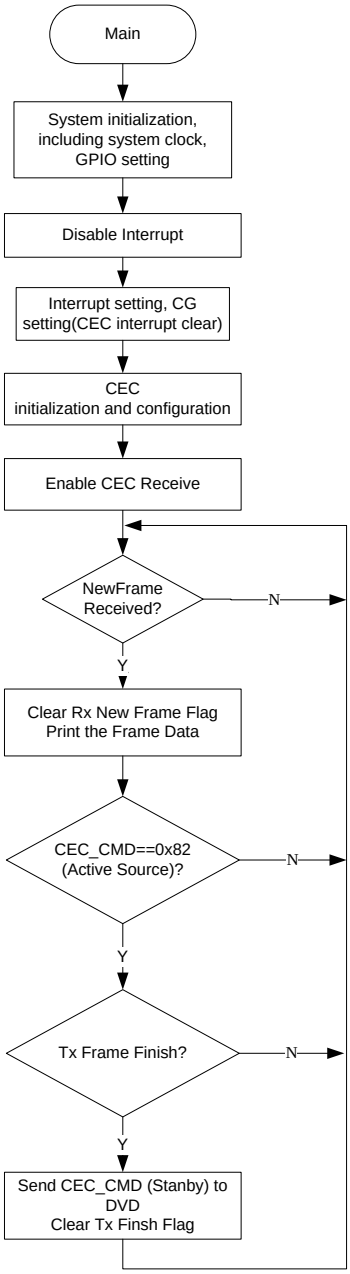
ペリフェラルドライバ(CEC, GPIO)を用いたサンプルプログラムです。

このサンプルでは以下を行います。

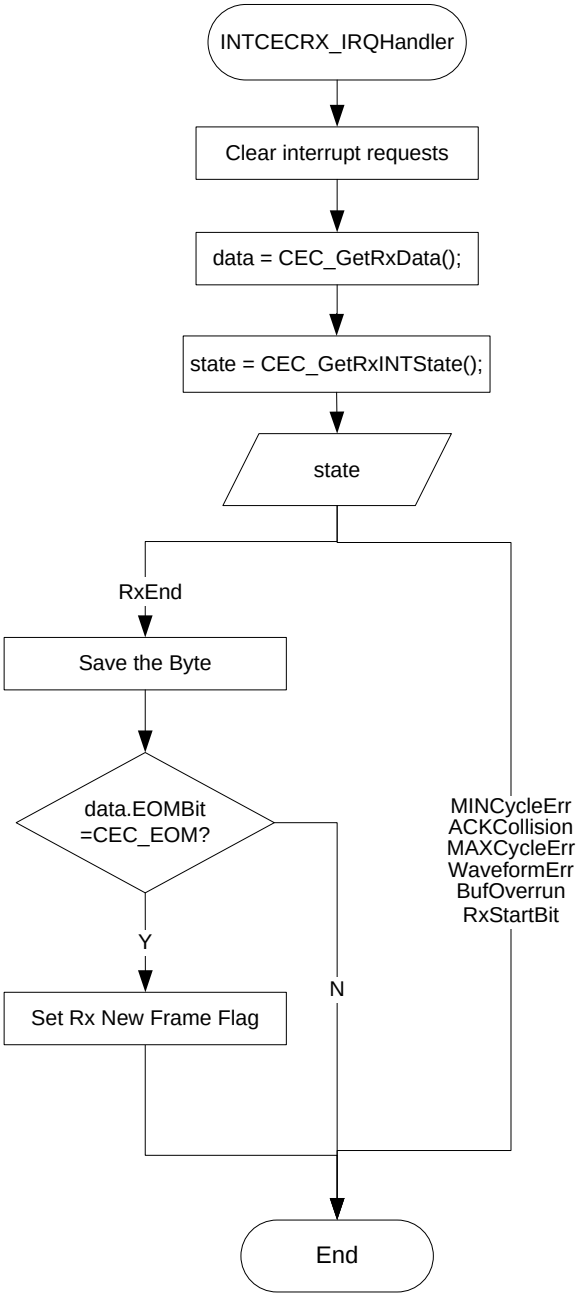
1. CEC 設定と初期化を行います。
2. CEC コマンドを送信します。
3. CEC データを受信します。

- **フローチャート:**

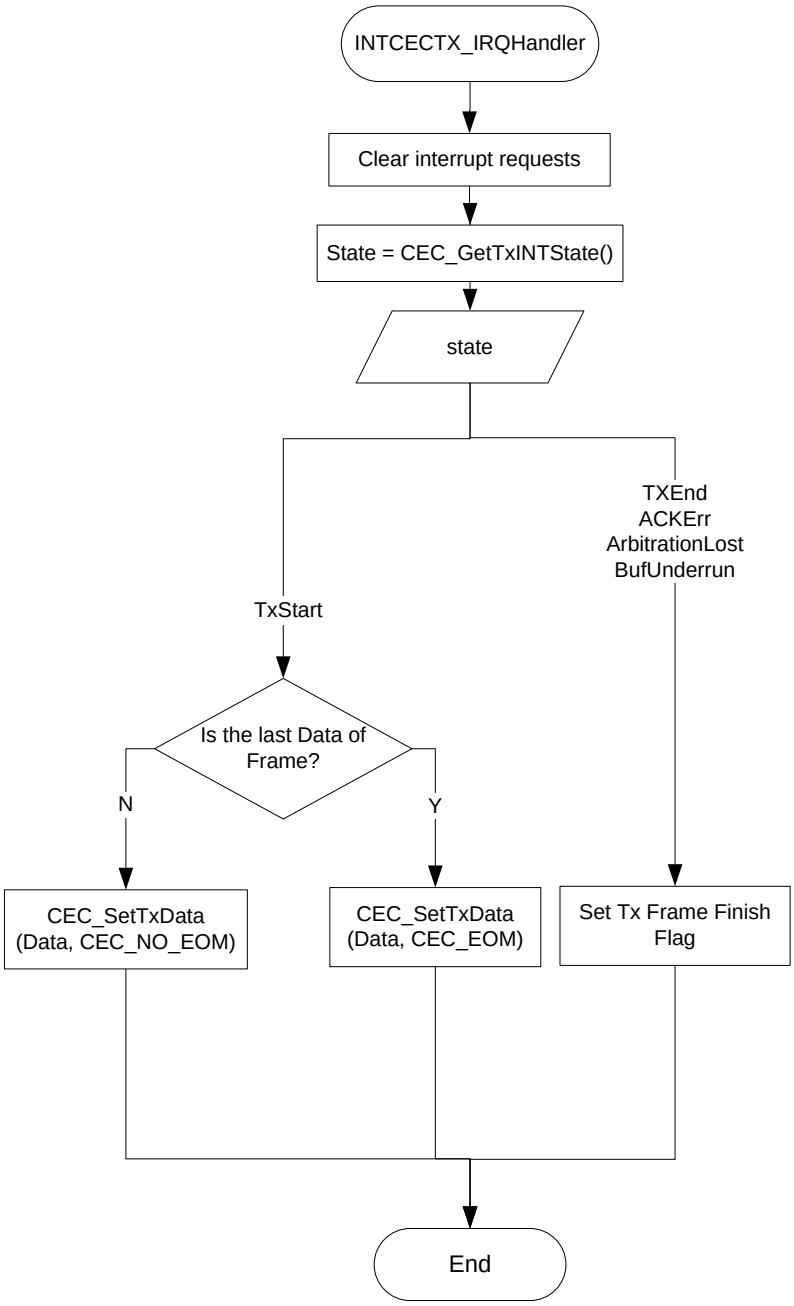
[Main 関数]



[CEC 受信割り込み関数]



[CEC 送信割り込み関数]



- **サンプルプログラムのコードと説明**

[Main 関数]

CEC を許可し、CEC の初期設定を行います。

```
CEC_Enable();
CEC_SWReset();
CEC_DefaultConfig();
```

次にロジカルアドレスを設定し、CEC 受信を許可します。

```
CEC_SetLogicalAddr(CEC_TV);
CEC_SetRxCtrl(ENABLE);
```

上記設定後、最初の 1 フレーム目のデータを送信します。

```
if(CEC_BROADCAST==Destination){
      CEC_SetTxBroadcast(ENABLE);
}else{
      CEC_SetTxBroadcast(DISABLE);
}
CEC_SetTxData(CEC_Data[0], CEC_NO_EOM);
CEC_StartTx();
```

[CEC 受信割り込み関数]

まず INTCECRX_IRQ ハンドラ内で、割り込み状態を取得します。

```
CEC_RxINTState state;
state = CEC_GetRxINTState();
```

次に受信データを取得します。

```
CEC_DataTypeDef data;
data = CEC_GetRxData();
```

[CEC 送信割り込み関数]

まず INTCECTX_IRQ ハンドラ内で、割り込み状態を取得します。

```
CEC_TxINTState state;
state = CEC_GetTxINTState ();
```

次データが最新フレームのデータではないことを確認し、送信を行います。

```
CEC_SetTxData(CEC_Data[current_num],CEC_NO_EOM);
```

さらにその次のデータが最新フレームのデータではないことを確認し、送信を行います。

```
CEC_SetTxData(CEC_Data[current_num],CEC_EOM);
```

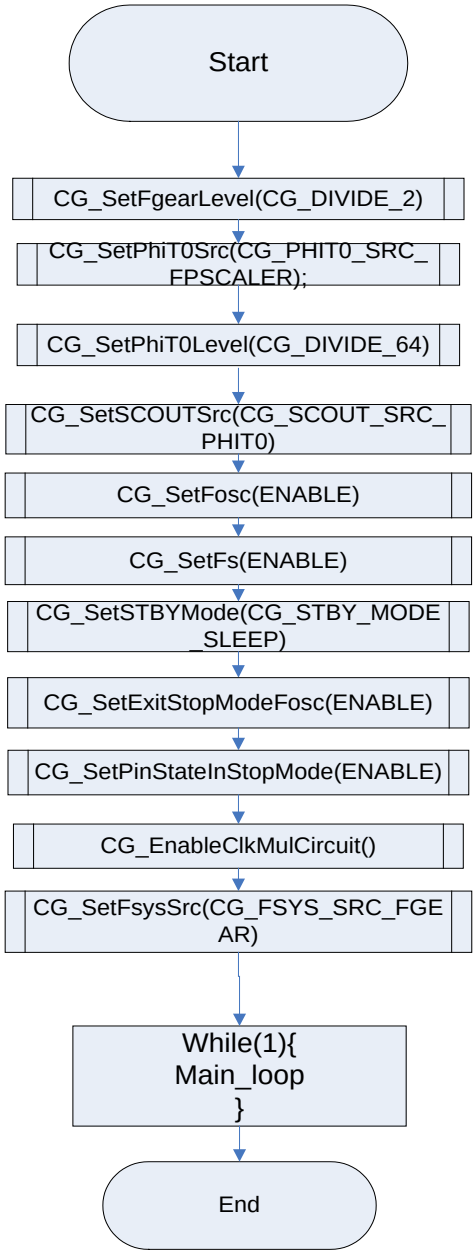
7-4 CG

ペリフェラル・ドライバ(CG)を使用したサンプルプログラムです。

このサンプルでは以下を行います。

1. 基本的な CG 動作の設定
2. NORMAL モードと SLEEP 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。

```
/* set fgear = fc/2 */
CG_SetFgearLevel(CG_DIVIDE_2);

#if defined(__TMPM361_CG_H) || defined(__TMPM362_CG_H) || defined(__TMPM363_CG_H) || defined(__TMPM364_CG_H)
CG_SetPhiT0Src
CG_SetPhiT0Src(CG_PHIT0_SRC_FPSCALER);
#endif
CG_SetPhiT0Level(CG_DIVIDE_64);
```

```
/* set SCOUT source to ΦT0*/
CG_SetSCOUTSrc(CG_SCOUT_SRC_PHIT0);

/* enable high-speed oscillation*/
CG_SetFosc(ENABLE);

/* enable low-speed oscillation*/
CG_SetFs(ENABLE);

/* set low power consumption mode Sleep*/
CG_SetSTBYMode (CG_STBY_MODE_SLEEP);

/* set high-speed oscillation to be enabled after releasing stop mode*/
CG_SetExitStopModeFosc(ENABLE);

/* set pin status in stop mode to "active"*/
CG_SetPinStateInStopMode(ENABLE);

/* set up pll and wait 200us for pll to warm up , set fc source to fpll*/
CG_EnableClkMulCircuit();    /*this module refer to    Programming Example */

/* set system clock to fgear */
CG_SetFsysSrc(CG_FSYS_SRC_FGEAR);
```

[マルチクロック回路の許可: CG_ENABLEClkMulCircuit()]

マルチクロック回路を許可します。

```
#define C200US    0x0096U              /* @12M Oscillator */

Result CG_EnableClkMulCircuit(void)
{
      Result retval = ERROR;
      WorkState st = BUSY;

      CG_SetClkMulTimes(CG_MUL_4TIMES);
      retval = CG_SetPLL(ENABLE);

      if (retval == SUCCESS) {
            /*set warm up time to about 200us*/
            CG_SetWarmUpTime(CG_WARM_UP_SRC_X1, C200US);
            CG_StartWarmUp();
            /*wait warm up to end */
            do {
                  st = CG_GetWarmUpState();
            } while (st != DONE);
            retval = CG_SetFcSrc(CG_FC_SRC_FPLL);

      } else {
            /*Do nothing */
      }
      return retval;
}
```

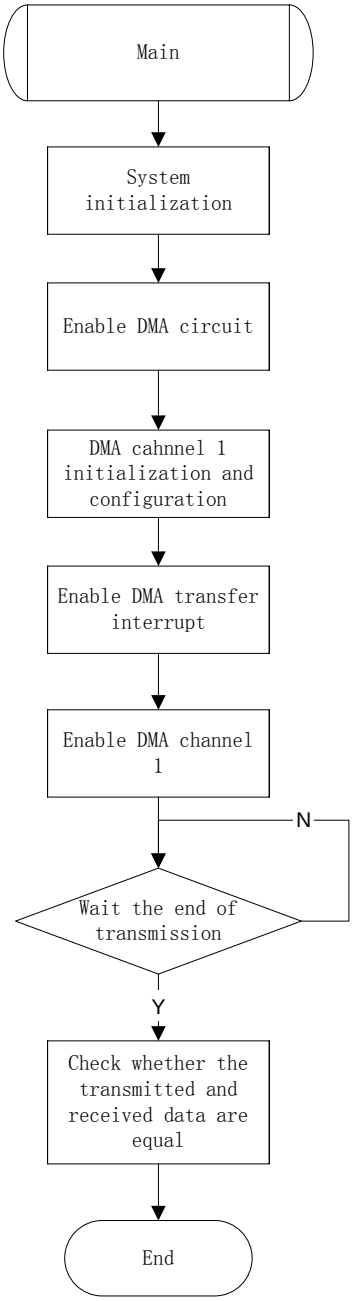
7-5 DMAC

ペリフェラルドライバ (DMAC) を使用したサンプルプログラムです。

このサンプルでは以下を行います。

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

- **フローチャート:**



- **サンプルプログラムのコードと説明**

DMAC_InitTypeDef 構造体を作成し、構造体メンバに初期値を設定します。

```
DMAC_InitTypeDef DMAC_InitStruct;

DMAC_InitStruct.TxDirection = DMAC_MEMORY_TO_MEMORY;
DMAC_InitStruct.SrcAddr = (uint32_t) SRC_Buffer;
DMAC_InitStruct.DstAddr = (uint32_t) DST_Buffer;
```

```
DMAC_InitStruct.SrcIncrementState = ENABLE;
DMAC_InitStruct.DstIncrementState = ENABLE;
DMAC_InitStruct.SrcBitWidth = DMAC_WORD;
DMAC_InitStruct.DstBitWidth = DMAC_WORD;
DMAC_InitStruct.SrcBurstSize = DMAC_4_BEATS;
DMAC_InitStruct.DstBurstSize = DMAC_4_BEATS;
DMAC_InitStruct.TxSize = BUFFER_SIZE;
DMAC_InitStruct.TxINT = ENABLE;
```

DMAC の許可と初期設定を行います。その後、DMA 転送割り込みを許可し、DMAC 転送を開始します。

```
DMAC_Enable();
DMAC_Init(myChannel, &DMAC_InitStruct);
DMAC_TxINTConfig(myChannel, DMAC_INT_TX_END, ENABLE);
DMAC_SetDMAChannel(myChannel, ENABLE);
```

DMAC 転送が終わるまで待ちます。

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

DMAC 転送終了時に ISR の DMA 転送終了フラグがセットされます。

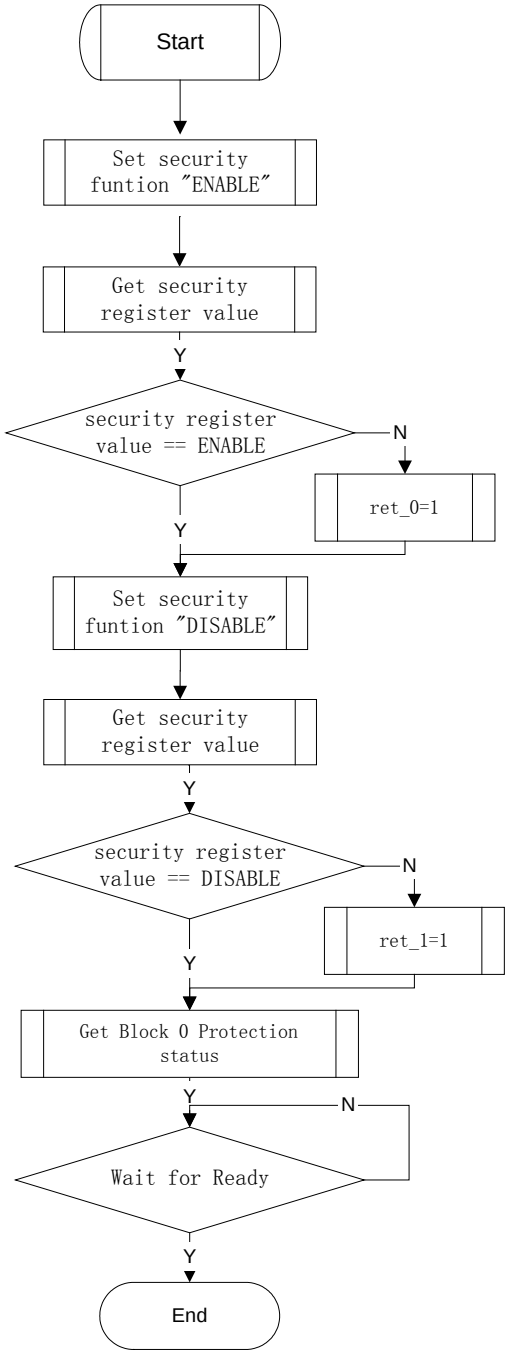
7-6 FLASH

ペリフェラルドライバ(FLASH)を使用したサンプルプログラムです。

このサンプルでは以下を行います。

1. Flash レジスタのリード／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**

以下はサンプルコードです。

```
#include "tmpm36x_fc.h"

int main(void)
{
	FunctionalState tmp_value_sec0, tmp_value_sec1, tmp_value_block;
	uint32_t ret_0 = 0U, ret_1 = 0U,ret_2 = 0U;

	while (1) {
		/* Set security function "ENABLE" */
```

```
        FC_SetSecurityBit(ENABLE);

        /* Get security register value */
        tmp_value_sec0 = FC_GetSecurityBit();
        if (tmp_value_sec0 != ENABLE) {
            ret_0 = 1U;
        }

        /* Set security function "DISABLE" */
        FC_SetSecurityBit(DISABLE);

        /* Get security register value */
        tmp_value_sec1 = FC_GetSecurityBit();
        if (tmp_value_sec1 != DISABLE) {
            ret_1 = 1U;
        }

        /* Get Block 0 Protection status */
        tmp_value_block = FC_GetBlockProtectState(FC_BLOCK_0);

        if (tmp_value_block != DISABLE) {
            ret_2 = 1U;
        }

        /* Wait for Ready */
        while (FC_GetBusyState() != DONE) {
        /* Do nothing */
        };
    }
}
```

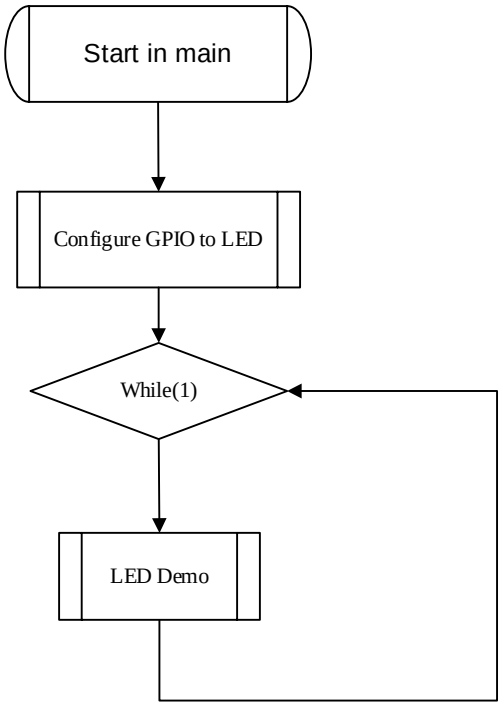
7-7 GPIO

ペリフェラルドライバ(GPIO)を用いたサンプルプログラムです。

このサンプルでは以下を行います。

1. GPIO の初期化
2. GPIO へのデータ書き込み

- **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、API GPIO_Init()関数をコールして、GPIO を LED 用に設定します。GPIO_InitTypeDef 構造体を作成し、構造体メンバに初期値を設定します。

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

その後、API GPIO_Init()関数をコールして、LED 用に初期化します。

```
GPIO_Init(GPIO_PM, GPIO_BIT_4, &led_io);
```

while ループにて、LED を On/Off させます。
API GPIO_WriteData()関数をコールして、GPIO_DATA レジスタに 1 をライトすることで LED を ON します。

```
GPIO_WriteDataBit(GPIO_PM, GPIO_BIT_4, GPIO_BIT_VALUE_1);
```

API GPIO_WriteData()関数をコールして、GPIO_DATA レジスタに 0 をライトすることで LED を OFF します。

```
GPIO_WriteDataBit(GPIO_PM, GPIO_BIT_4, GPIO_BIT_VALUE_0);
```

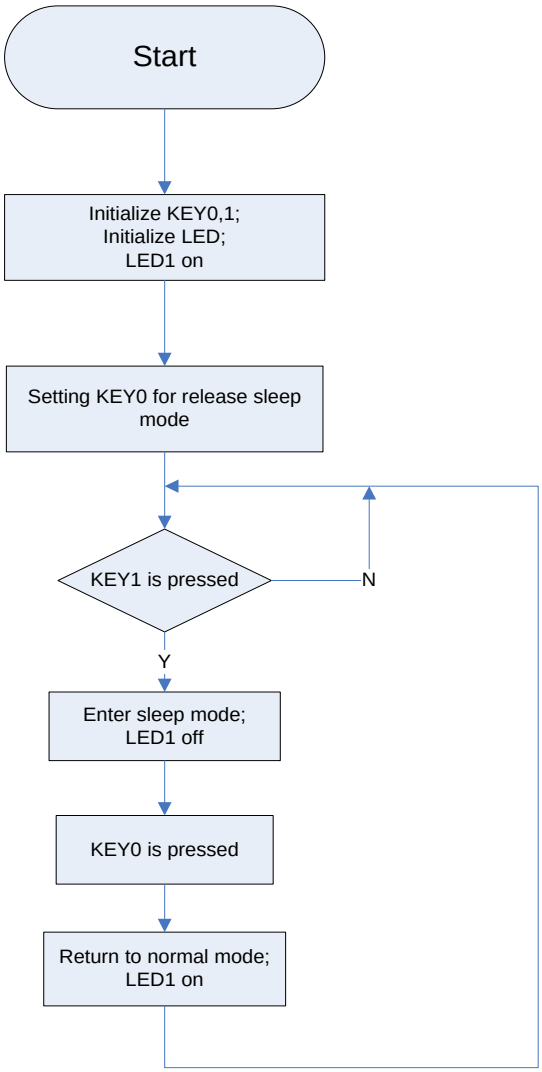
7-8 KWUP

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

このサンプルでは以下を行います。

1. KWUP 設定
2. 低電力モードへの移行と解除

- **フローチャート:**



- **サンプルプログラムのコードと説明**

最初に KWUP 用の初期値を設定し、KWUP 用の GPIO を設定します。

```
KWUP_SettingTypeDef keySetting;
KWUP_PortStatus portStatus;
SystemInit();

/* set PJ4,PJ5 to Key0,1 */
TSB_PJ_FR2_PJ4F2 = 1U;
TSB_PJ_PUP_PJ4UP = 1U;
TSB_PJ_IE_PJ4IE = 1U;

TSB_PJ_FR2_PJ5F2 = 1U;
TSB_PJ_PUP_PJ5UP = 1U;
TSB_PJ_IE_PJ5IE = 1U;

/* Set PM4 to Output Port */
LED_Configuration();
LED_On(LED1);
```

```
/* enable KWUP interrupt */
NVIC_ClearPendingIRQ(INTKWUP_IRQn);
NVIC_EnableIRQ(INTKWUP_IRQn);

keySetting.KeyN = KWUP_INPUT_0;
keySetting.PullUpCtrl = KWUP_PUP_CTRL_BY_DYNAMIC;
keySetting.ActiveState = KWUP_ACTIVE_BY_RISING_EDGE;
keySetting.INTNewState = ENABLE;
KWUP_SetConfig(&keySetting);

KWUP_SetPullUpConfig(KWUP_CYCLES_4_FS, KWUP_CYCLES_256_FS);

KWUP_ClearINTReq();
```

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

```
/* enable low power mode release interrupt to INTKWUP0 */
TSB_CG->IMCGF = 0x1000U;
TSB_CG->IMCGF += 0x010U;

do {
    portStatus = KWUP_GetPortStatus();

    if (portStatus.Bit.Key1 == 0U) {        /* press key(PJ5) */
        LED_Off(LED1);
        /* enter low power mode */
        __WFI();

        /* press KEY(PJ4) to release low power mode */

        LED_On(LED1);
    }
}
while (1);
```

7-9 RC

ペリフェラルドライバ(RC)を使用したサンプルプログラムです。

以下の例を含みます。

1. RC 制御レジスタのリード／ライト

- **サンプルプログラムのコードと説明**

以下はサンプルコードです。

```
#include "tmpm36x_rc.h"

int main(void)
{
    RC_RAMWait tmp_value;
    uint32_t ret_0 = 0U;
    while (1) {
        /* Set the value of RCWAIT register */
        RC_SetRAMWait(RC_RAMWAIT_1);
```

```
            /* Get the value of RCWAIT register */
            tmp_value = RC_GetRAMWait();
            if (tmp_value != RC_RAMWAIT_1) {
                ret_0 = 1U;
            }
        }
}
```

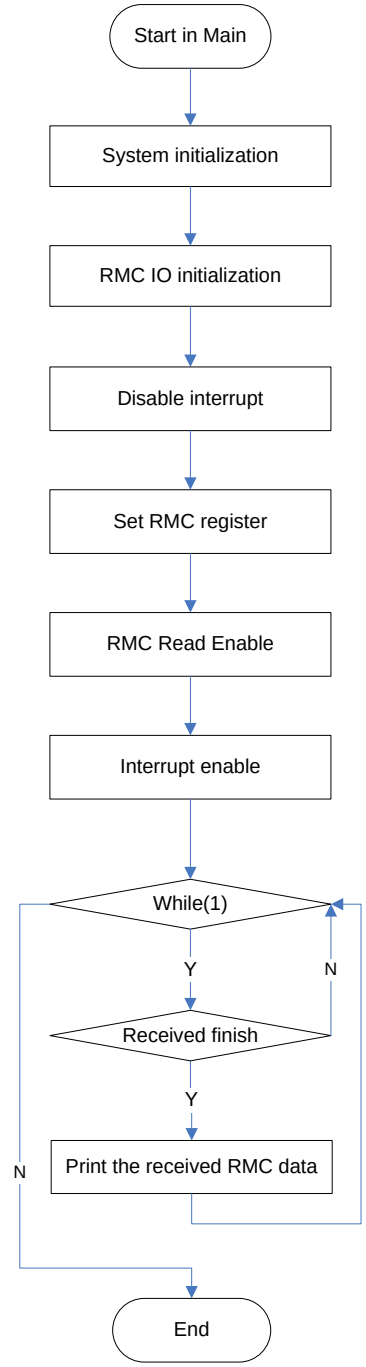
7-10 RMC

ペリフェラルドライバ (RMC, GPIO)のサンプルプログラムです。

このサンプルでは以下を行います。

1. RMC IO 初期化
2. RMC データの受信

- **フローチャート:**



- **サンプルプログラムのコードと説明**

まず、myRMC 構造を作成し、全てのデータ項目を設定します。

```
RMC_InitTypeDef myRMC;

myRMC.LeaderPara.MaxCycle = RMC_MAX_CYCLE;
myRMC.LeaderPara.MinCycle = RMC_MIN_CYCLE;
myRMC.LeaderPara.MaxLowWidth = RMC_MAX_LOW_WIDTH;
myRMC.LeaderPara.MinLowWidth = RMC_MIN_LOW_WIDTH;
myRMC.LeaderPara.LeaderDetectionState = ENABLE;
myRMC.LeaderPara.LeaderINTState = DISABLE;
```

```
myRMC.FallingEdgeINTState = DISABLE;
myRMC.SignalRxMethod = RMC_RX_IN_CYCLE_METHOD;
myRMC.LowWidth = RMC_TRG_LOW_WIDTH;
myRMC.MaxDataBitCycle = RMC_TRG_MAX_DATA_BIT_CYCLE;
myRMC.LargerThreshold = RMC_LARGER_THRESHOLD;
myRMC.SmallerThreshold = RMC_SMALLER_THRESHOLD;
myRMC.InputSignalReversedState = DISABLE;
myRMC.NoiseCancellationTime = RMC_NOISE_CANCELLATION_TIME;
```

次に、RMC チャネル 0 の許可と初期化を行います。

```
RMC_Enable(TSB_RMC0);
```

そして、システムがアイドル状態の場合、RMC を禁止します。

```
RMC_Init(TSB_RMC0, &myRMC);
```

指定の RMC0 チャネルの受信を許可します。

```
RMC_SetRxCtrl(TSB_RMC0, ENABLE);
```

割り込み INTRMCRX_IRQHandler()で、指定の RMC0 チャネルの割り込み要因を取得します。

```
RMC_INTFactor myRMC_INTFactor;
myRMC_INTFactor = RMC_GetINTFactor(TSB_RMC0);
```

指定の RMC0 チャネルのリーダー検出結果を取得します。

```
RMC_LeaderDetection myRMC_LeaderDetection;
myRMC_LeaderDetection = RMC_GetLeader(TSB_RMC0);
```

指定の RMC0 チャネルからの受信データを取得します。

```
RMC_RxDataTypeDef myRMC_RxDataDef;
myRMC_RxDataDef = RMC_GetRxData(TSB_RMC0);
```

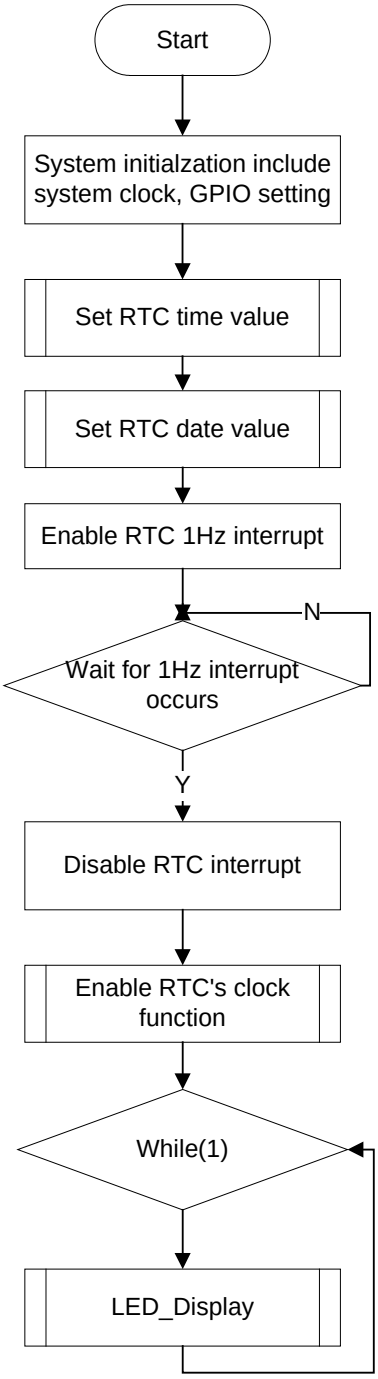
7-11 RTC

ペリフェラルドライバ (RTC, CG) のサンプルプログラムです。

このサンプルでは以下を行います。

1 RTC の時間設定
2 RTC の時間取得

- **フローチャート:**



- **サンプルプログラムのコードと説明**

RTC の初期化で、RTC_DateTypeDef 構造体と RTC_TimeTypeDef 構造体を作成し、全データ項目を設定します。

```
RTC_DateTypeDef DateStruct;
RTC_TimeTypeDef TimeStruct;
DateStruct.LeapYear = RTC_LEAP_YEAR_2;
DateStruct.Year = (uint8_t)10;
DateStruct.Month = (uint8_t)12;
DateStruct.Date = (uint8_t)31;
```

```
DateStruct.Day = RTC_FRI;
TimeStruct.HourMode = RTC_12_HOUR_MODE;
TimeStruct.Hour = (uint8_t)11;
TimeStruct.AmPm = RTC_PM_MODE;
TimeStruct.Min = (uint8_t)59;
TimeStruct.Sec = (uint8_t)50;
```

RTC の時間と日付の値を設定し、RTC(1Hz)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。

```
RTC_SetTimeValue(&TimeStruct);
RTC_SetDateValue(&DateStruct);
__disable_irq();
/* enable RTC interrupt */
NVIC_ClearPendingIRQ(INTRTC_IRQn);
TSB_CG->IMCGF = 0x00000020;
TSB_CG->IMCGF = 0x00000021;
NVIC_EnableIRQ(INTRTC_IRQn);
/* Enable 1Hz interrupt */
RTC_SetAlarmOutput(RTC_PULSE_1_HZ);
/* Enable RTCINT */
RTC_SetRTCINT(ENABLE);
__enable_irq();
```

RTC 割り込みの発生を待ちます。

```
/* waiting for RTC register set finish */
while(fRTCSetting_ok != 1);
fRTCSetting_ok = 0;
```

RTC 割り込みを禁止し、RTC クロックを許可します。

```
/* Disable RTCINT */
__disable_irq();
RTC_SetRTCINT(DISABLE);
NVIC_DisableIRQ(INTRTC_IRQn);
__enable_irq();
/* Enable RTC Clock function */
RTC_EnableClock();
```

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。

```
uint8_t Year = 0U;
uint8_t Month = 0U;
uint8_t Date = 0U;
uint8_t Day = 0U;
Year = RTC_GetYear();
Month = RTC_GetMonth();
Date = RTC_GetDate(RTC_CLOCK_MODE);
Day = RTC_GetDay(RTC_CLOCK_MODE);
```

または 1 つの API 関数をコールします。

```
RTC_DateTypeDef DateStruct;
RTC_GetDateValue(&DateStruct);
```

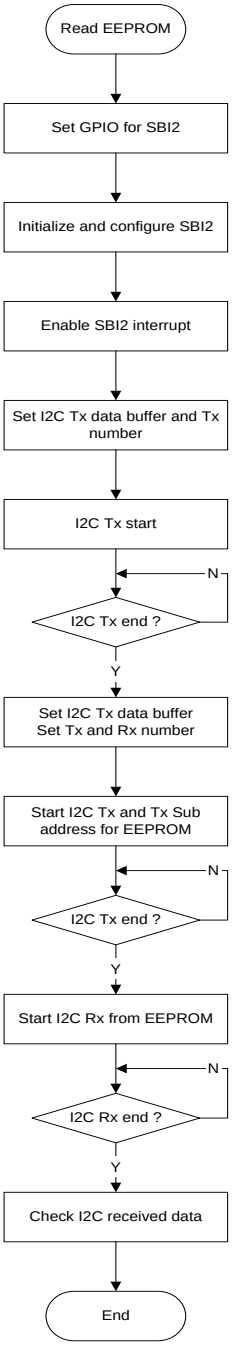
7-12 SBI

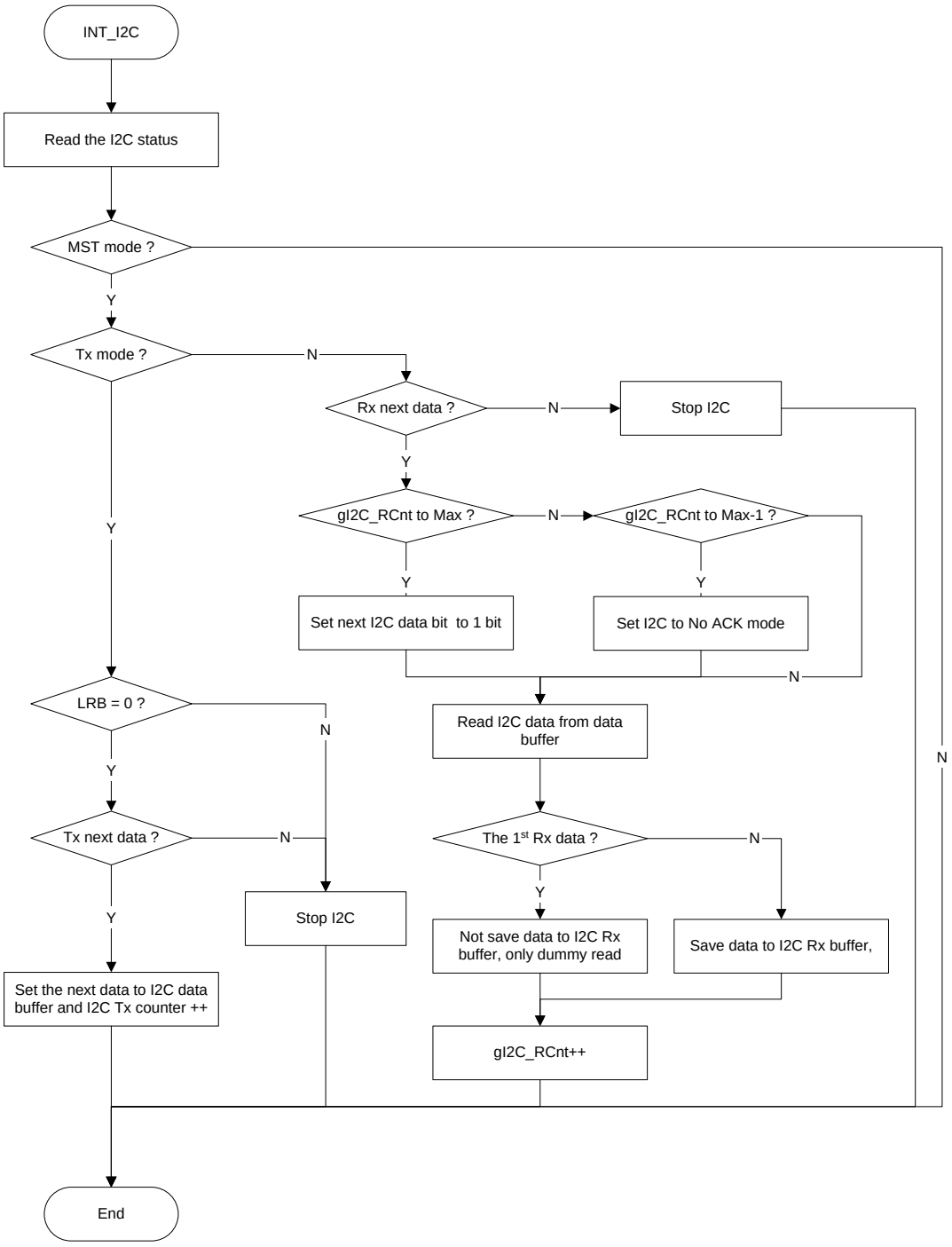
ペリフェラルドライバ(SBI, GPIO)を使用したサンプルプログラムです。

このサンプルでは以下を行います。

1. SBI 設定
2. SBI スレーブによるデータ送信
3. SBI スレーブによるデータ受信

- **フローチャート:**





- **サンプルプログラムのコードと説明**

まず、SBI_InitI2CTypeDef 構造体を作成し、構造体メンバに初期値を設定します。

```
myI2C.I2CSelfAddr = SELF_ADDR;
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2C.I2CACKState = ENABLE;
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
```

その後、SBI2 を許可します。

```
SBI_Enable(TSB_SBI2);
SBI_SWReset(TSB_SBI2);
SBI_InitI2C(TSB_SBI2, &myI2C);
```

```
    NVIC_EnableIRQ(INTSBI2_IRQn);
```

以下はライト動作処理です。
I2C の送信データバッファとカウンタを初期化します。I2C バスがフリーとなっていることを確認し、スレーブアドレスと転送先を設定します。その後スタートコンディションを送信します。

```
        do{
            i2c_state = SBI_GetI2CState(TSB_SBI2);
        } while (i2c_state.Bit.BusState);;
        SBI_SetSendData(TSB_SBI2, SLAVEADDR | SBI_I2C_SEND);
        SBI_GenerateI2CStart(TSB_SBI2);
```

次に I2C 割り込みにて I2C バス状態をリードし、次のデータを送信します。I2C データ転送終了を確認し、I2C を終了します。

```
  SBIx = TSB_SBI2;
  sbi_sr = SBI_GetI2CState(SBIx)
  ……
  SBI_SetSendData(SBIx, gI2C_TxData[gI2C_WCnt]);
  ……
  SBI_GenerateI2CStop(SBIx);
```

I2C 送信バッファのデータは EEPROM にライトされています。

以下はリード動作処理です。
I2C の送信データバッファと送受信カウンタを初期化します。I2C バスがフリーとなっていることを確認し、スレーブアドレスと転送先を設定します。その後スタートコンディションを送信します。

```
        do{
            i2c_state = SBI_GetI2CState(TSB_SBI2);
        } while (i2c_state.Bit.BusState);
        SBI_SetSendData(TSB_SBI2, SLAVEADDR | SBI_I2C_SEND);
        SBI_GenerateI2CStart(TSB_SBI2);
```

次に I2C 割り込みにて I2C バス状態をリードし、次のデータを送信します。I2C データ転送終了を確認し、I2C を終了します。(EEPROM のサブアドレスはこの処理で転送されます)

```
  SBIx = TSB_SBI2;
  sbi_sr = SBI_GetI2CState(SBIx)
  ……
  SBI_SetSendData(SBIx, gI2C_TxData[gI2C_WCnt]);
  ……
  SBI_GenerateI2CStop(SBIx);
```

EEPROM のサブアドレスをセットした後、EEPROM のデータをリードします。まずスレーブアドレスを送信するため I2C を開始した後、I2C データバッファレジスタをダミーリードし、端子を開放します。

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

その後、I2C 割り込み込みにて I2C バス状態とデータバッファをリードし、次の I2C データをセットします。

```
  SBIx = TSB_SBI2;
  sbi_sr = SBI_GetI2CState(SBIx)
  ……
  tmp = SBI_GetReceiveData(SBIx);
  ……
  SBI_SetI2CBitNum(SBIx, SBI_I2C_DATA_LEN_1);
  ……
```

```
SBI_SetI2CACK(TSB_SBI2,DISABLE);
```

EEPROM からのデータはリードしており、I2C 受信バッファに格納されています。

補足: データ受信終了後の ACK モードには"No-ACK"がセットされています。次の I2C 処理のために ACK モードと他のパラメータを再設定してください。

EEPROM のライトアドレスとリードアドレスは同じはずであるため、送信データと受信データの比較を行い、送信確認を行います。

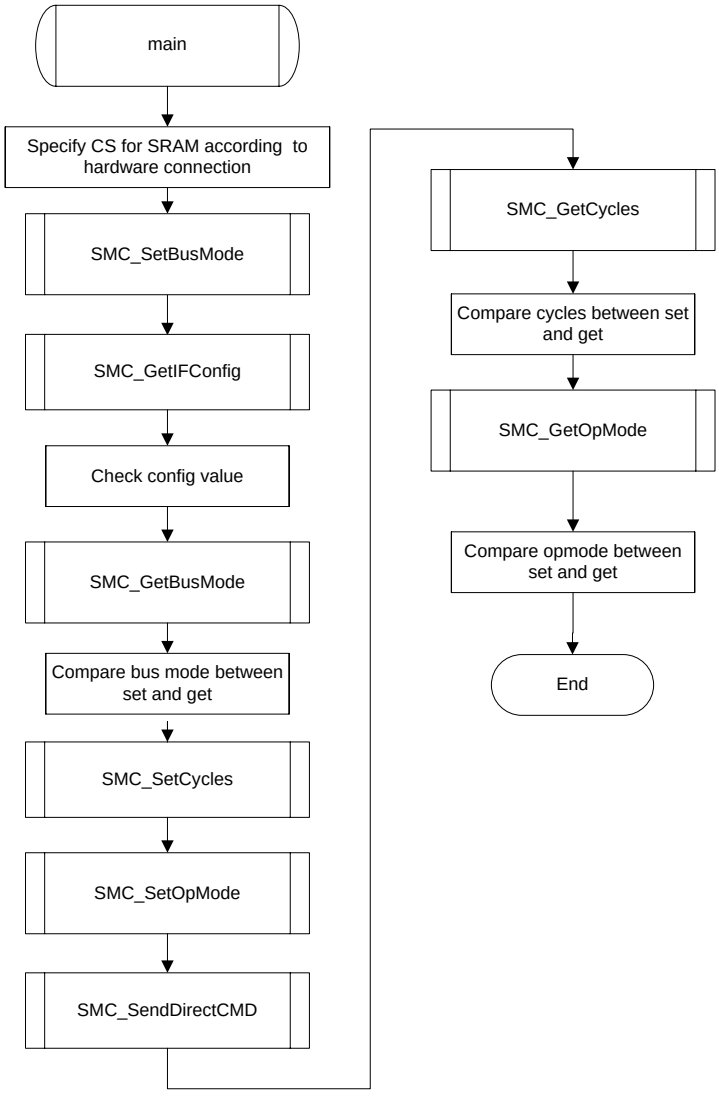
7-13 SMC

ペリフェラルドライバ(SMC)を使用したサンプルプログラムです。

このサンプルでは以下を行います。

1. SMC の設定

- **フローチャート:**



- **サンプルプログラムのコードと説明**

スタティックメモリ: 16 ビット SRAM、マルチプレクスバスモード

まずバスモードを設定します。

```
    uint8_t bus_w = 0xFFU;
    bus_w = SMC_BUS_MULTIPLEX;
    SMC_SetBusMode(bus_w);
```

次に外部インタフェースの設定状態を取得します。

```
    SMC_GetIFConfig(&config);
```

その後、SRAM の AC スペックに合わせた動作モード、サイクルを設定し、スタティックメモリへダイレクトコマンドを送信します。

```
    /* Set cycles time according to AC timing of SRAM datasheet, base clock: SMCCLK */
    cycles_w.RC_Time = SMC_READ_CYCLE_TIME_5;
    cycles_w.WC_Time = SMC_WRITE_CYCLE_TIME_5;
    cycles_w.CEOE_Time = SMC_CEOE_DELAY_CYCLE_TIME_1;
    cycles_w.WP_Time = SMC_WE_PULSE_CYCLE_TIME_1;
    cycles_w.PC_Time = SMC_PAGE_CYCLE_TIME_5;
    cycles_w.TR_Time = SMC_TURN_AROUND_CYCLE_TIME_1;
    SMC_SetCycles(&cycles_w);

    opmode_w.BusWidth = SMC_DATA_BUS_16BIT;
    opmode_w.ReadBurstLen = SMC_READ_BURST_4BEAT;
    opmode_w.ALE = ENABLE;          /* ALE must be enabled in multiplex bus mode */

    /* Make sure SMC_SetCycles and SMC_SetOpMode are performed before
  SMC_SendDirectCMD */
    cmd.CmdType = SMC_CMD_UPDATEREGS;
    cmd.ChipSelect = SMC_CS2;
    SMC_SendDirectCMD(&cmd);
```

そして、サイクルと動作モードを取得し、ベリファイします。

```
    SMC_GetCycles(chip, &cycles_r);
    SMC_GetOpMode(chip, &opmode_r);
```

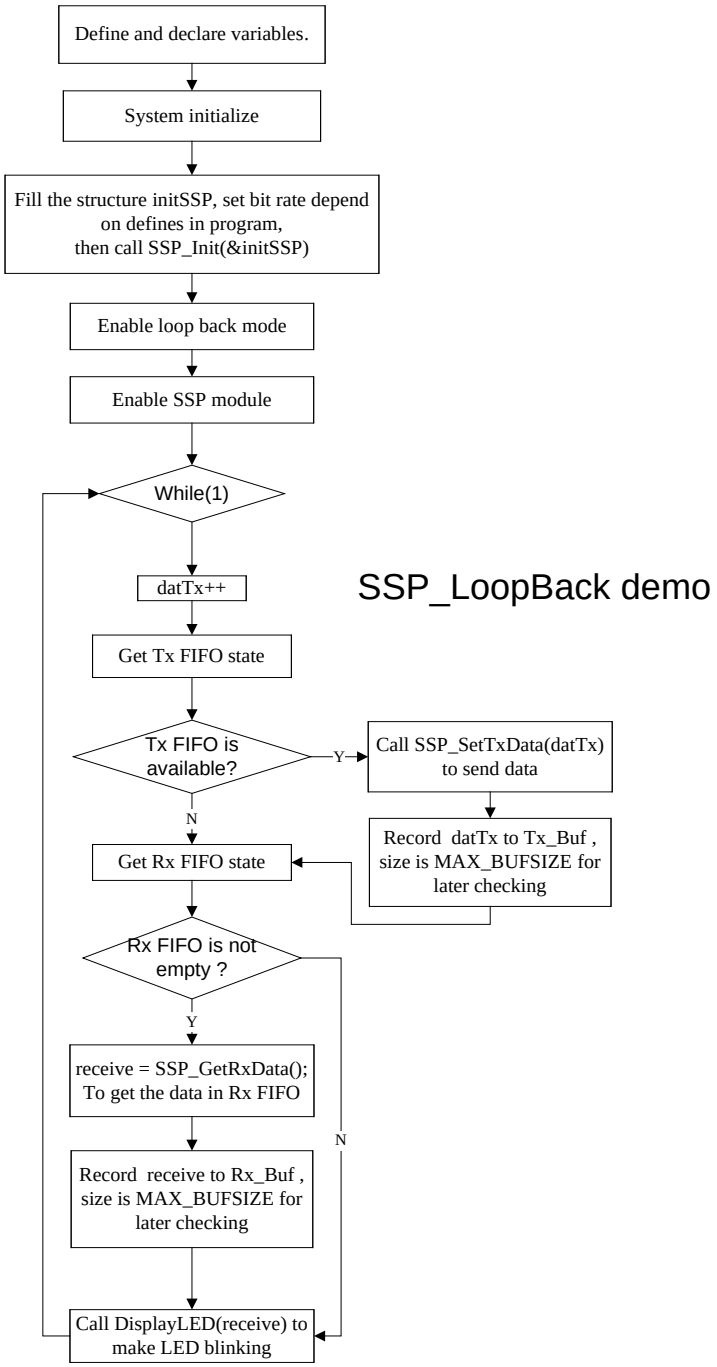
7-14 SSP

ペリフェラルドライバ (SSP, GPIO) のプログラムサンプルです。

このサンプル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**

まず SSP_InitTypeDef 構造体と他の変数を準備します。

```
SSP_InitTypeDef initSSP;

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

uint16_t receive = 0U;
uint16_t Rx_Buf[MAX_BUFSIZE];
```

```
    uint16_t Tx_Buf[MAX_BUFSIZE];
```

システムを初期化し、LED の設定を行います。

```
    SystemInit();

     /* Enable the LED on board to display some info */
    LED_Configuration();
```

SSP_InitTypeDef 構造体メンバに初期値を設定します。

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

      /* default is to run at maximum bit rate */
      initSSP.PreScale = 2U;
      initSSP.ClkRate = 1U;

     /* define BITRATE_MIN to run at minimum bit rate */
     /*     BitRate = fSYS / (PreScale x (1 + ClkRate)) */
  #ifdef BITRATE_MIN
      initSSP.PreScale = 254U;
      initSSP.ClkRate = 255U;
  #endif

      initSSP.ClkPolarity = SSP_POLARITY_LOW;
      initSSP.ClkPhase = SSP_PHASE_FIRST_EDGE;
      initSSP.DataSize = 16U;
      initSSP.Mode = SSP_MASTER;
```

上記設定後、SSP_Init()関数をコールして、SSP を初期化します。

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

ループバックモードを許可します。

```
    SSP_SetLoopBackMode(ENABLE);
```

SSP 動作を許可します。

```
    SSP_Enable();
```

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

```
    while (1) {

      datTx++;

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

      /* check if there is data arrived */
```

```
        fifoState = SSP_GetFIFOState(SSP_RX);
        if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
            receive = SSP_GetRxData();
            if (cntRx < MAX_BUFSIZE) {
                Rx_Buf[cntRx] = receive;
                cntRx++;
            } else {
                /* Place a break point here to check if receive data is right. */
                __NOP();
            }

        } else {
            /* do nothing */
        }

        DisplayLED(receive);
    }
```

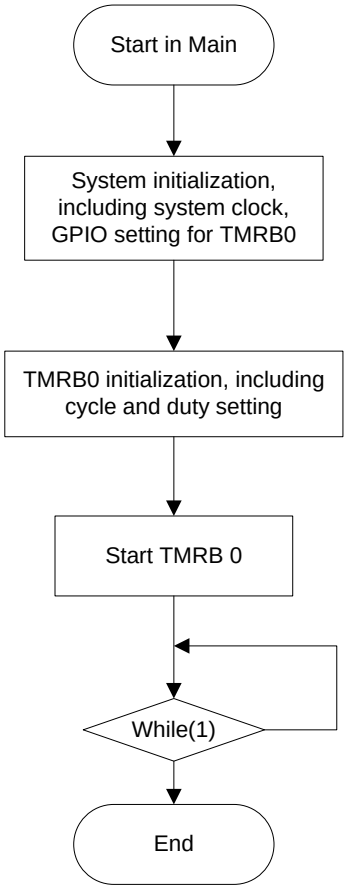
7-15 TMRB

ペリフェラルドライバ (TMRB, GPIO) のプログラムサンプルです。

このサンプル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 TMRB_InitTypeDef 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。

```
TMRB_InitTypeDef myTB;
myTB.Mode = TMRB_INTERVAL_TIMER;
myTB.ClkDiv = TMRB_CLK_DIV_8;
myTB.TrailingTiming = TMRB_1ms;          /* Specific value depends on system clock */
myTB.UpCntCtrl = TMRB_AUTO_CLEAR;
myTB.LeadingTiming = TMRB_1ms / 2;      /* Specific value depends on system clock */
```

その後、TMRB0 動作の許可、及び初期化を行います。

```
TMRB_Enable(TSB_TB0);
TMRB_Init(TSB_TB0, &myTB);
```

そして、INTTB0 割り込みを許可し、TMRB0 動作を開始します。

```
TMRB_SetRunState(TSB_TB0, TMRB_RUN);
```

7-16 SIO/UART

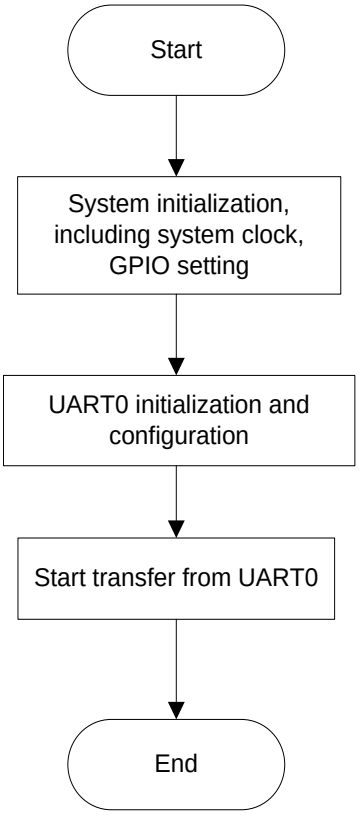
7-16-1 例: UART 出力

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

このサンプルでは以下を行います。

1. UART 設定と初期化
2. UART 送信制御
3. データ送信に UART0 の TX 割り込みを使用

- **フローチャート:**



- **サンプルプログラムのコードと説明**

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

```
    UART_InitTypeDef myUART;
myUART.BaudRate = 115200;
    myUART.DataBits = UART_DATA_BITS_8;
    myUART.StopBits = UART_STOP_BITS_1;
    myUART.Parity = UART_NO_PARITY;
    myUART.Mode = UART_ENABLE_RX | UART_ENABLE_TX;
    myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
```

UART を許可し、UART の初期化を行います。

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

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

送信データは TxBuffer に文字列として確認できます。
残りの処理は UART0 送信割り込みルーチンです。

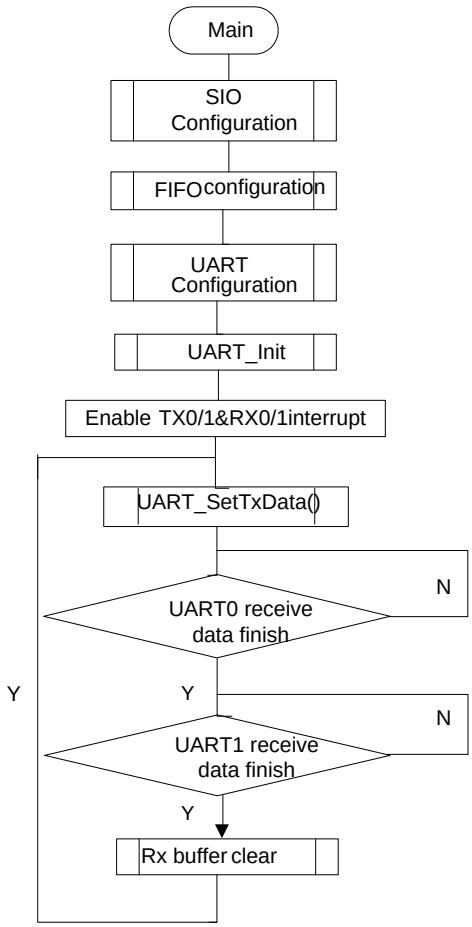
7-16-2　例: UART FIFO

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。

このサンプル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 UART1 に設定します。

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
    if (SCx == TSB_SC0) {
        TSB_PE->CR |= GPIO_BIT_4;
        TSB_PE->FR2 |= GPIO_BIT_4;
        TSB_PE->FR2 |= GPIO_BIT_5;
        TSB_PE->IE |= GPIO_BIT_5;
    } else if (SCx == TSB_SC1) {
        TSB_PL->CR |= GPIO_BIT_4;
        TSB_PL->FR1 |= GPIO_BIT_4;
        TSB_PL->FR1 |= GPIO_BIT_5;
```

```
            TSB_PL->IE |= GPIO_BIT_5;
        }
    }
```

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);
```

UART0/1 の有効化と初期設定を行います。

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

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

FIFO の初期設定を行います。

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

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

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

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

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

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

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

    UART_RxFIFOClear(UART0);
    UART_RxFIFOClear(UART1);

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

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

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

上記設定を行い、その後、UART0/1 の送信割り込みを有効にします。

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

NVIC_EnableIRQ(INTTX1_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);
	}
}
```

UART1 の送信割り込みルーチン:

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

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

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);
	}
}
```

UART1 の受信割り込みルーチン:

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

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

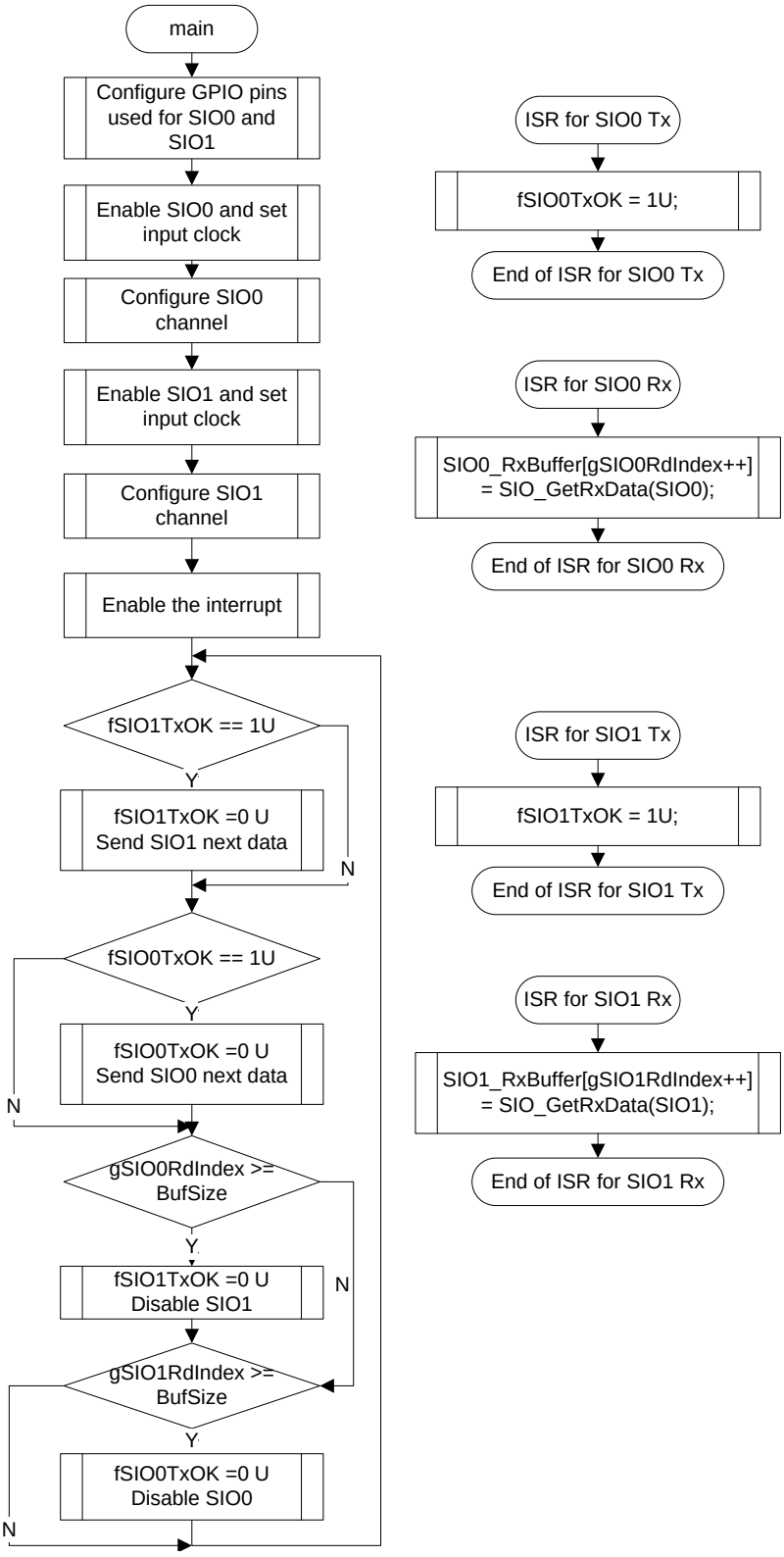
7-16-3　例: SIO

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。

このサンプル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.IntervalTime = SIO_SINT_TIME_SCLK_1;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_T1;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;
```

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.IntervalTime = SIO_SINT_TIME_SCLK_1;
SIO1_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO1_Init.TransferDir = SIO_LSB_FRIST;
SIO1_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO1_Init.DoubleBuffer = SIO_WBUF_ENABLE;

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

SIO 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 */
        }
    }
```

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);
}
```

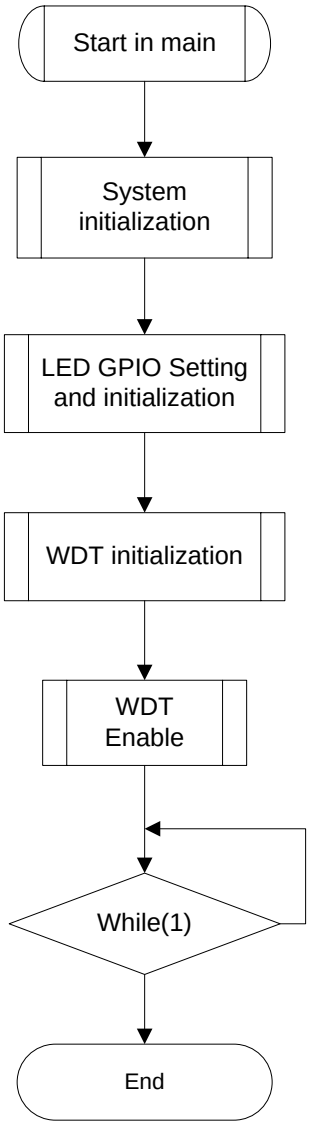
7-17 WDT

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

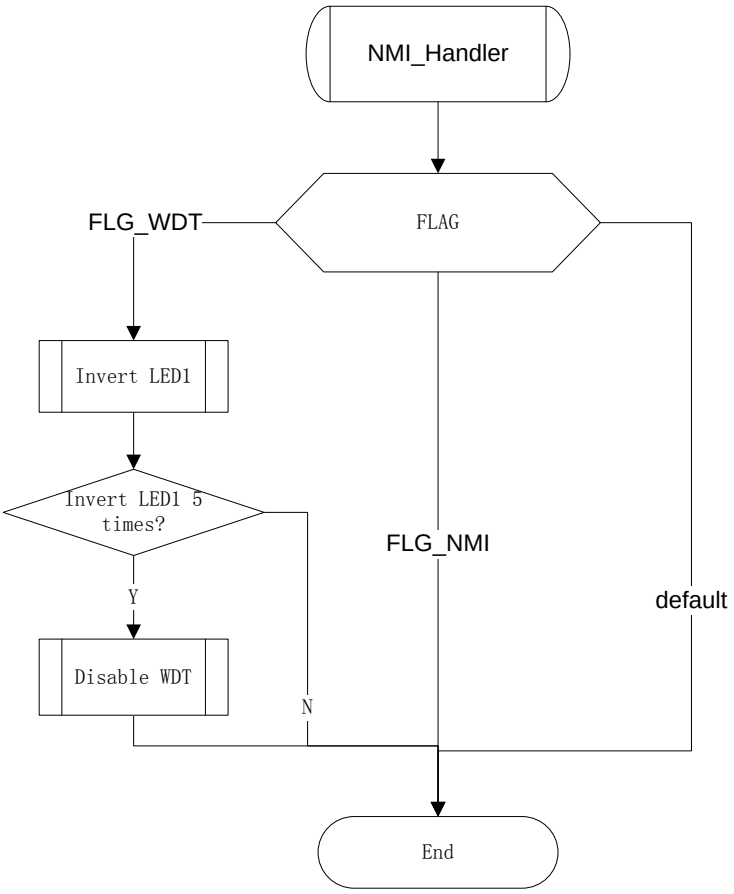
このサンプルでは以下を行います。

1. WDT カウントオーバーフローによる NMI 割り込みの発生
2. LED1 の点滅
3. LED1 を 5 回点滅させた後、WDT を禁止します。

- **フローチャート**
  [Main 関数]



[NMI_Handler 関数]



- **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();
```

NMI 割り込みの発生を待ちます。

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