TOSHIBA

TX03 Peripheral Driver Usage Example (TMPM330)

Ver 1
Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M330UE-01E

RESTRICTIONS ON PRODUCT USE

- DO NOT USE THIS SOFTWARE WITHOUT THE SOFTWARE LISENCE AGREEMENT.

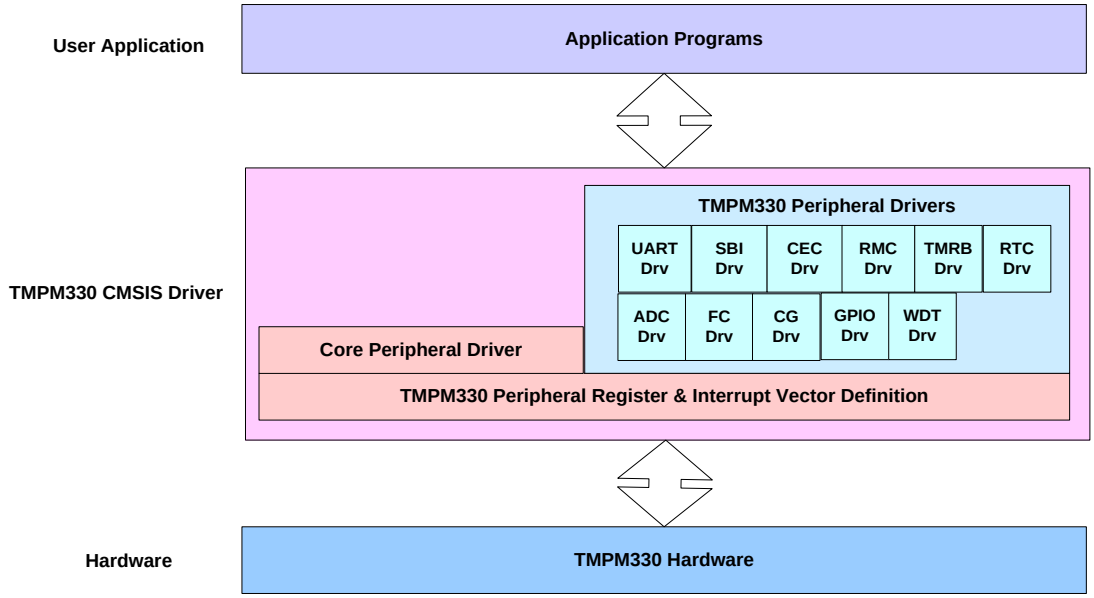
Index

1 General description

The example programs described hereafter are specially designed for TOSHIBA TMPM330 MCU. The programs can be executed individually each to show the main MCU functions. Users can utilize some portions of the programs and integrate them into users' own programs to perform customized functions.

2 Overview

User application utilizes TX03 peripherial driver as following.



3 Build-in hardware usage

Hardware	Channel	Use presence, use
CG	Clock Gear	Used for CG demo
	PLL	PLL on (4x)
Standby mode		Use SLEEP only.
SysTick		Unused
Watch dog timer (WDT)		WDT function
External interrupt (INT)	INT0	Unused
	INT1	Unused
	INT2	Unused
	INT3	Unused
	INT4	Unused
	INT5	Unused
	INT6	Unused
	INT7	Unused

Hardware	Channel	Use presence, use
SIO	SIO0	UART0 ,1,2 communication control (transmission)
	SIO1	UART0, 1 communication control (reception /transmission)
	SIO2	UART0, 2 communication control (reception)
Serial bus (SBI)	SBI0	Slave mode
	SBI1	Unused
	SBI2	Read and write EEPROM data / Master mode
CEC		CEC reception and transmission function
Remote control signal processor (RMC)	RMC0	Unused
	RMC1	Remote controller reception
16-bits timer	TMRB0	Use for PPG output
	TMRB1	Unused
	TMRB2	Unused
	TMRB3	Use for Get frequency
	TMRB4	Unused
	TMRB5	Use for Interval timer
	TMRB6	Unused
	TMRB7	Unused
	TMRB8	Use for Get frequency
	TMRB9	Unused
RTC		Count the second, and display with the year month day hour minite second
10-bit A/D converter	AIN0	Use for ADC data read
	AIN1	Unused
	AIN2	Unused
	AIN3	Unused
	AIN4	Unused
	AIN5	Unused
	AIN6	Unused
	AIN7	Unused
	AIN8	Unused
	AIN9	Unused
	AIN10	Unused
	AIN11	Unused

4 Pin Usage

The example programs are tested on the IAR TMPM330-SK Evaluation board except RMC and CEC that are tested on M330 demo board.

Following is pin usage for example programs.

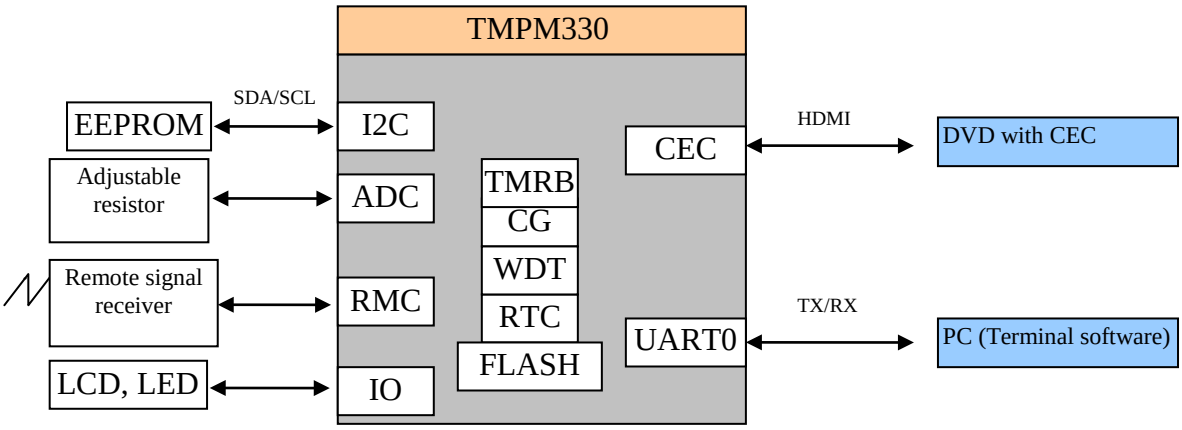
No	Name	Use
1	PD6, AN10	Unused
2	PD7, AN11	Unused
3	AVSS	GND
4	VREFH	+3.3V
5	AVCC	+3.3V
6	PG3, INT4	LED4
7	PK2, TB9OUT	Unused
8	PJ5, TB7OUT	Unused
9	PH4, TB2IN0	Unused
10	PH5, TB2IN1	Unused
11	PG7, TB8OUT	Unused
12	TEST2	Unused
13	DVSS	GND
14	DVCC	+3.3V
15	PG4, SO2, SDA2	SDA2
16	PG5, SI2, SCL2	SCL2
17	PG6, SCK2	Unused
18	TEST1	Unused
19	PF7, INT5	LCD DB7
20	PE0, TXD0	SIO0 TX
21	PE1, RXD0	SIO0 RX
22	PE2, SCLK0, CTS0	SIO0 SCLK0
23	PE4, TXD1	SIO1 TX
24	PE5, RXD1	SIO1 TX
25	PE6, SCLK1, CTS1	SIO1 SCLK1
26	PG0, SO0, SDA0	SDA0/LED1
27	PG1, SI0, SCL0	SCL0/LED2
28	PG2, SCK0	LED3
29	PB3	Unused
30	PH0, TB0IN0, BOOT	Unused
31	PH1, TB0IN1	LCD_E
32	PH2, TB1IN0	LCD_RW
33	PF0, TXD2	SIO2 TX
34	PF1, RXD2	SIO2 RX
35	PF2, SCLK2, CTS2	LCD_BL
36	PH3, TB1IN1	LCD_RS
37	PB4	KEY1
38	PI0, TB0OUT	PPG output
39	PJ6, INT6	Unused
40	PI1, TB1OUT	Unused
41	PB5	KEY2
42	PI2, TB2OUT	Unused
43	PB6	KEY3
44	PF4, SO1, SDA1	LCD_DB4
45	PF5, SI1, SCL1	LCD_DB5
46	PF6, SCK1	LCD_DB6
47	PB7	KEY4
48	PI3, TB3OUT	Unused

No	Name	Use
49	PJ1, INT1	LED6
50	PK0, CEC	CEC (*)
51	PK1, SCOUT, ALARM	Unused
52	PI4, TB4OUT	Unused
53	PI5, TB5OUT	Unused
54	PB0, TDO, SWV	TDO
55	PA0, TMS, SWDIO	TMS
56	PA1, TCK, SWCLK	TCK
57	TEST3	Unused
58	PJ7, INT7	Unused
59	PB1, TDI	TDI
60	PB2, TRST	TRST
61	PF3, RXIN1	RMC1 (*)
62	DVCC	+3.3V
63	DVSS	GND
64	PA2, TRACECLK	Unused
65	PA3, TRACEDATA0	Unused
66	PA4, TRACEDATA1	Unused
67	PA5, TRACEDATA2	Unused
68	PA6, TRACEDATA3	Unused
69	PA7	Unused
70	PJ0, INT0	LED5
71	CVCC	+3.3V
72	X2	Connect 10MHz oscillator
73	CVSS	GND
74	X1	Connect 10MHz oscillator
75	REGVSS	GND
76	REGVCC	+3.3V
77	XT1	Connect 32.768kHz oscillator
78	XT2	Connect 32.768kHz oscillator
79	PI6, TB4IN0	Unused
80	NMI	Unused
81	MODE	GND
82	RESET	RESET
83	PI7, TB4IN1	Unused
84	PH6, TB3IN0	Source clock input for frequency measure
85	PH7, TB3IN1	Unused
86	PJ2, INT2	LED7
87	PJ3, INT3	LED8
88	PJ4, TB6OUT	Unused
89	PE3, RXIN0	Unused
90	TEST4	Unused
91	PC0, AN0	AN0
92	PC1, AN1	Unused
93	PC2, AN2	Unused
94	PC3, AN3	Unused
95	PD0, AN4, TB5IN0	Unused
96	PD1, AN5, TB5IN1	Unused
97	PD2, AN6, TB6IN0	Unused
98	PD3, AN7, TB6IN1	Unused
99	PD4, AN8	Unused
100	PD5, AN9	Unused

* Pins used for CEC and RMC example programs on M330 Demo board.

5 Development Environment

Following is development environment:



1. Hardware board:
   1) IAR TMPM330-SK Evaluation board
   2) TOSHIBA TMPM330 Demo board

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

6 Functions

This chapter will focus on the functions demonstrated in driver usage example.

6-1 Operation mode

There are five operation modes for TMPM330: NORMAL, SLOW, IDLE, SLEEP and STOP mode.

- **NORMAL mode:**

Both high-speed clock and low-speed clock operate in NORMAL mode.
System clock use the high speed clock (fc).

- **SLOW mode:**

This mode is to operate the CPU core and the peripheral hardware by using the low-speed clock with high-speed clock stopped. The SLOW mode reduces power consumption compared to the NORMAL mode. This mode allows only the following peripheral functions to operate: I/O ports, real-time clock (RTC), CEC and remote control signal preprocessor (RMC).

Note: The sample program of SLOW mode is in CG example.

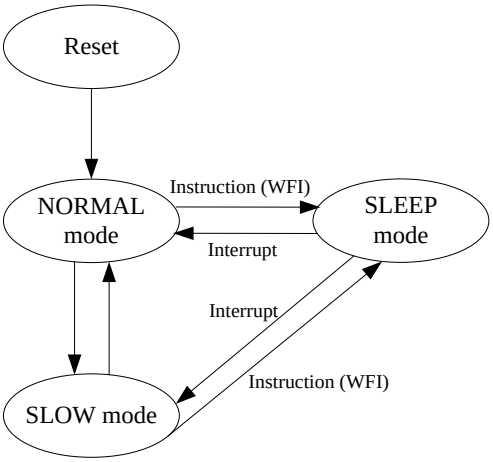
IDLE, SLEEP and STOP mode are low power mode.
To shift to lower power mode, in system control register STBYCR0<STBY2:0>, select the IDLE, SLEEP or STOP mode, and run the WFI (Wait For Interrupt) command.

Note: Only SLEEP mode is demostrated in driver example.

- **SLEEP mode:**

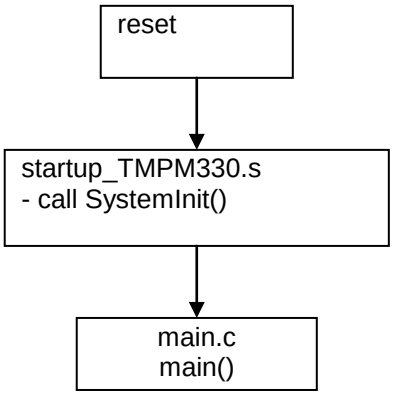
The internal low-speed oscillator (fs), real time clock, CEC (only for reception) and RMC operate. By releasing the SLEEP mode, the device returns to the preceding mode of the SLEEP mode and starts operation.
RTC interrupt, CEC interrupt, RMC interrupt and the extern interrupt can release the SLEEP mode.

Note: The sample program of SLEEP mode is integrated into RMC and CEC example.



6-2 START UP

Start up is running between reset and main function.



6-3 ADC

Sample the value of ADC and display the value on LCD.

Uses interrupt to check if the ADC is finished.

Display the ADC data on LCD.

After Power On or Reset, LCD displays:

```
ADC Value: 25
****
```

Note: First line displays ADC value (0-100).

Second line displays a percentage bar.

6-4 CEC

This function receives the data from the CEC line. Send Header/opercode/upper 8bit of operand to PC HyperTerminal in Hex format.
This function intends to send CEC message onto CEC line, and send the Header/opercode/upper 8bit of operand of the message to PC HyperTerminal in Hex format at the same time.
For TMPM330 demo board, self-logical address = E.

Press the PC keyboard in HyperTerminal or press remote key to send responding CEC message. Following are the supported CEC messages and the trigger key.

PC keyboard:	Remote key	CEC message
1	[1]	Power (pass through)
2	[2]	System standby
3	[3]	Play (pass through)
4	[4]	Stop (pass through)

Power (pass through), the CEC data sample:

CEC_Send: XX 44 40
CEC_Send: XX 45

System standby, the CEC data sample:

CEC_Send: EF 36

Play (pass through), the CEC data sample:

CEC_Send: XX 44 44
CEC_Send: XX 45

Stop (pass through), the CEC data sample:

CEC_Send: XX 44 45
CEC_Send: XX 45

* **Note:** The send address data is XX (according to customer's test environment), and the receive data is related to test environment.

If CPU is in [Sleep] mode when receiving the CEC message, CPU changes to [Normal] mode firstly, and then processes the CEC message.

6-5 CG

This function demonstrates how to switch between operation modes.

NORMAL mode → SLOW mode→ SLEEP mode

6-6 FLASH

This function demonstrates the feature of flash APIs such as erase and writes operation.

The demo runs in Normal mode and User Boot Mode of Single chip mode.

－Reset procedure:

Includes Mode judgement, Programming routine (flash APIs), Copy routine

・Mode judgment routine: judge to enter User Boot Mode or Normal Mode

・Copy routine: copy programming routine(flash APIs) from flash to RAM

・Programming routine: runs at RAM and swap Program A and Program B code in flash

－Program A/B(Reset procedure) are programmed in flash block in advance

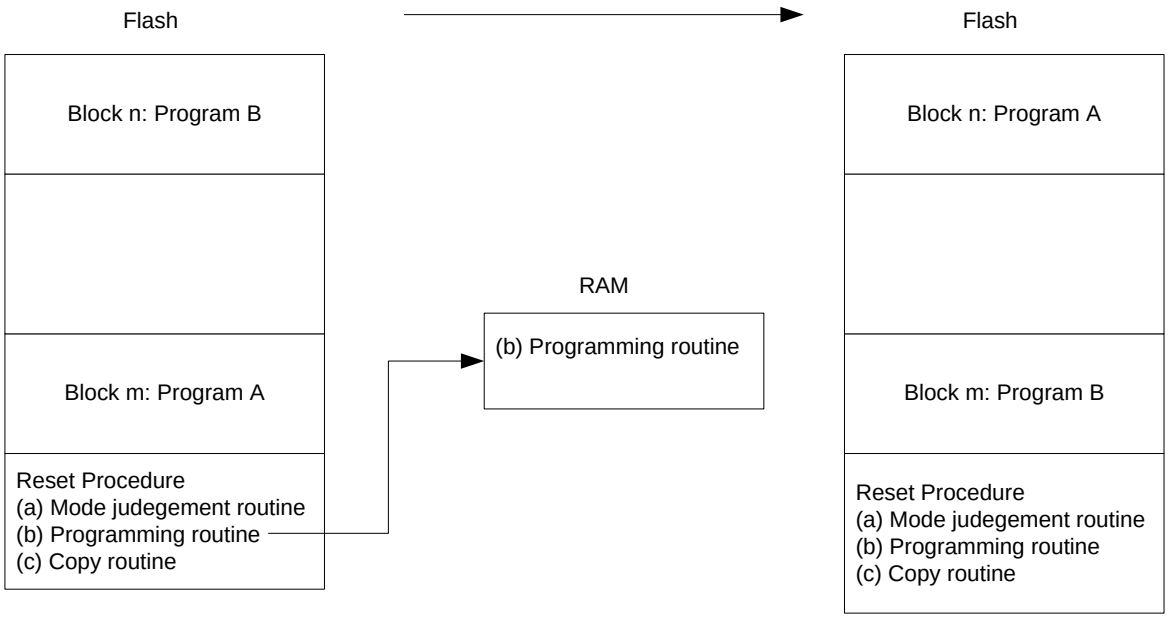
－Program A/B（A: LED1~4 blink, B: LED 5~8 blink）

By default, the program A will run firstly

－KEY1 is used for mode judgment in Reset procedure

KEY1 pressed → User boot mode

KEY1 un-pressed → Normal mode



- **Demo sequence**

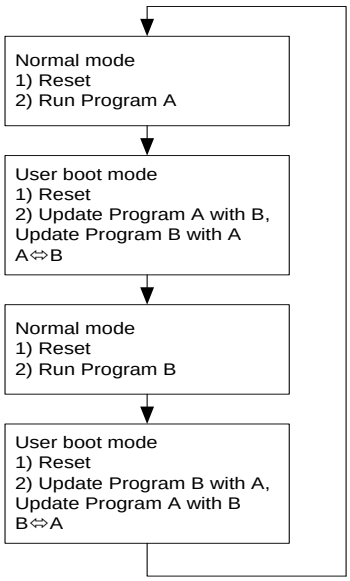
(1) Power on
The demo board runs Reset Procedure.
Then initial program A stored in flash runs.
LED1~4 blink, and LCD displays “A is running ……”.

(2) Press B1 button (RESET) while pressing KEY1 button, and release KEY1 until LCD displays “User Boot Mode”.
The demo board runs Reset Procedure: Programming routine is copied to RAM by Copy routine.
Then run Programming routine to swap program A and B in flash.
During this period, LCD displays information such as “RAM transferring ……”, “FLASH burning ……” and “Finished Restart ……”.

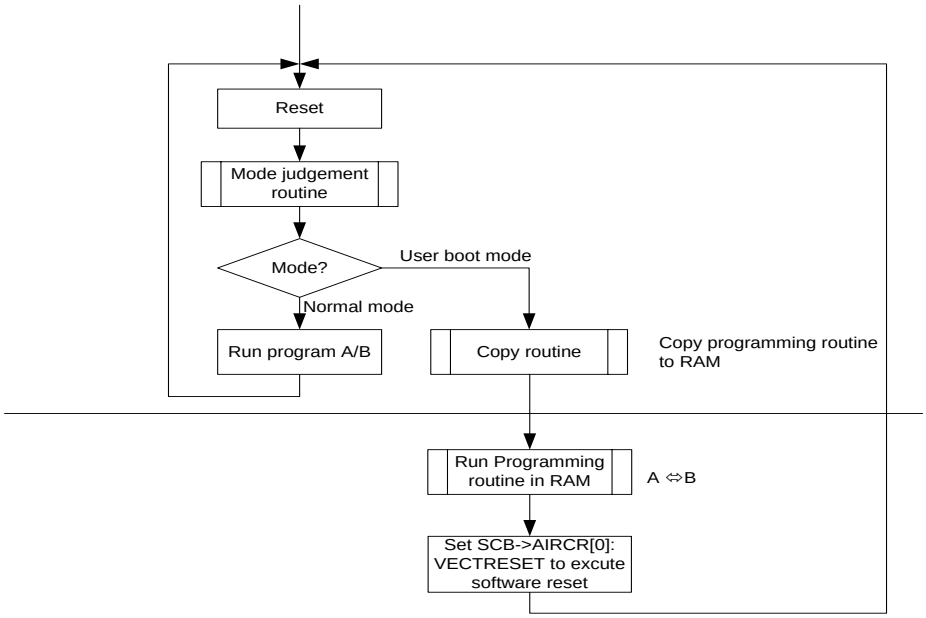
(3) After LCD has displayed “Finished Restart ……”, the demo will reset by software automatically.
Then program B stored in flash runs.
LED5~8 blink, and LCD displays “B is running ……”.

(4) Press B1 button (RESET) while pressing KEY1 button again, and release KEY1 until LCD display information “User Boot Mode”.
Same as step (2).

(5) After LCD has displayed “Finished Restart ……”, the demo will reset by software automatically.
Then program A stored in flash runs.
LED1~4 blink, and LCD displays “A is running ……”.



- **Demo process flow**



When the demo entering user boot mode, LCD will display as below:

User Boot Mode

(Reset Procedure)

While RAM transferring or flash programming, LCD displays the current processing.

LCD display sample:

RAM transferring ……

FLASH burning ……

Finished Restart ……

(Copy routine) (Programming routine) (Programming routine)

6-7 GPIO

This function configures GPIO to make LED and KEY work.

6-8 RMC

6-8-1 RMC receiving

This application intends to receive the Infrared remote signal and decode it. Decoded data will be displayed on PC through terminal software. Customer code or address code and command code will be displayed in Hex format.

- PC display format:

TMPM330 RMC Demo

RMC_1: XX XX

RMC_1: YY YY

Format	Terminal soft display
1	RMC_1：
2	RMC_2：
3	RMC_3：
4	RMC_4：
5	RMC_5：

6-8-2 Power mode change

Change the CPU operation mode. Only 2 modes are supported. NORMAL and SLEEP. Press remote controller power key to switch the power mode. And also maybe set the peripheral power according to power mode.

Current mode	Action (remote key)	Operation	Terminal display
NORMAL	[9] [CHDOWN] [VOL DOWN]	NORMAL→SLEEP	Now, Going to Sleep
SLEEP	[7] [CH UP] [VOL UP]	SLEEP→NORMAL	Wakeup from Sleep

6-9 RTC

Use the internal RTC to display the time and date on LCD.

Display the time and date on LCD and update the date and time.

Initial setting: **2010/10/22 12:50:55**, 24hours format.

Update interval: 1 second.

LCD displays:

2010/10/22 12:50:55

6-10 SBI

6-10-1 EEPROM Read/Write

Read and write the EEPROM data from or to on-board EEPROM.

Read all data and display data on PC through HyperTerminal.

Press K1 to read EEPROM.

HyperTerminal Display: TMPM330 SBI Demo

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

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

…

Finish

LCD display:

TMPM330 EEPROM READ

After read finished:

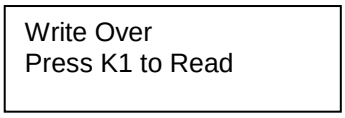
Read Over Press K2 to Write

Press K2 to write EEPROM.

Type a word (max 8 letters) through HyperTerminal, and then program will write the data into EEPROM.

The word written in and read out should be same.

After write finished:

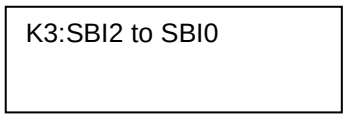


Press K1 to read EEPROM to verify whether write operation succeeds or not.

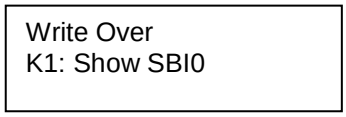
6-10-2 I2C Slave

This function intends to support the I2C bus slave mode.
Connect two I2C buses (SBI0 & SBI2) on IAR TMPM330-SK Evaluation board, one works as I2C master, and another works as I2C slave.
Uses SBI interrupt to handle I2C bus read/write.
Address of I2C slave: Any.
I2C Slave Slave (SBI0) receives “TOSHIBA” from Master (SBI2) and LCD displays it.

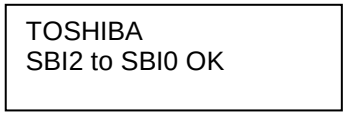
Press K3, SBI2 (master) and SBI0 (slave) demo will start.



When K3 is pressed, the string「TOSHIBA」will be sent from SBI2 (master) to SBI0 (slave).



When K1 is pressed, the string「TOSHIBA」that got from the SBI0 (slave) received buffer will be displayed.



(Attention)
In order to run this example, IAR TMPM330-SK board must be modified.
Connect the PG4/SDA2 and PG0/SDA0.
Connect the PG5/SCL2 and PG1/SCL0.

6-11 SIO

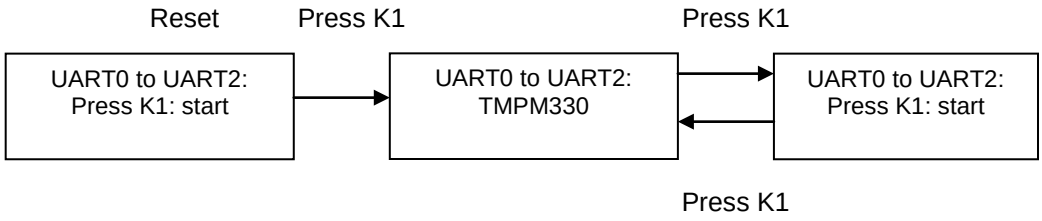
6-11-1 Transmission from UART0 To UART2

In the SIO sample program, UART0 is used to send data and UART2 is used to receive data.

String “TMPM330” is sent from UART0.

UART2 receives the data, and data will be displayed on LCD.

Press K1 to send. After send operation finished, press K1 to restart.

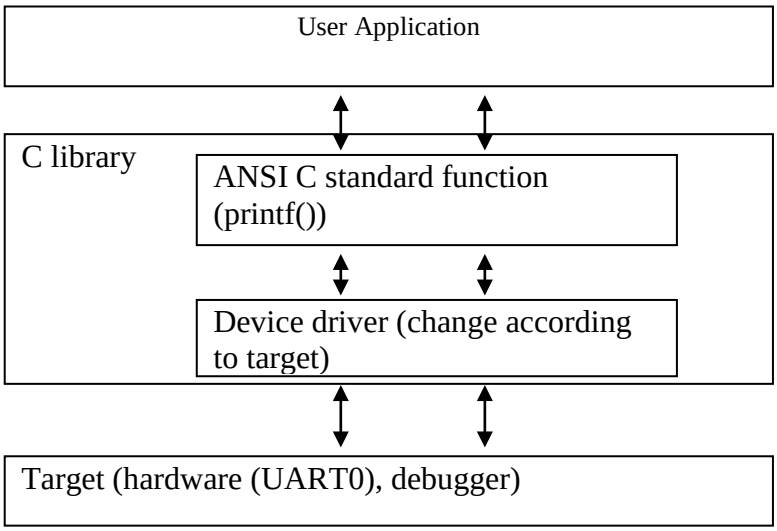


UART Settings:

Communication Channel:		UART0 and UART2
Data length	:	8 bits
Parity bit	:	None
Stop bit	:	1 bit
Baud rate	:	115200bps
Flow control	:	None

6-11-2 Retarget

This function intends to retarget the stdin and stdout of the C libarary.



Stdin and stdout are retargeted to UART0.

Application codes can utilize printf() and getchar() to output to or input from serial port.

Note: getchar() is used in CEC example.

UART Settings:

Communication Channel:		UART0
Data length	:	8 bits
Parity bit	:	None
Stop bit	:	1 bit
Baud rate	:	115200bps
Flow control	:	None

6-11-3 UART FIFO

In this sample program, UART0 sent the data "TMPM3301" to UART1 use FIFO, and at same time UART0 receive the data "TMPM3302" which send from UART1 and also use FIFO.
Set one breakpoint before the function ResetIdx(),when program stop in the breakpoint you can read the RxBuffer = "TMPM3302",and RxBuffer1 = "TMPM3301".

6-11-4 SIO

This demo will show synchronously transfer and receive usage of SIO module in TMPM330
It use the channel SIO0, SIO1 and transfer data synchronously between them.(connect TXD0 with RXD1, connect TXD1 with RXD0, connect sclk0 with sclk1)

6-12 TMRB

6-12-1 Interval Timer

This function implements an interval timer utilizing MCU timer.

Timer trailingtiming is set to 1ms.

Use this timer for LED blinking and the period is 1s (500ms ON and 500ms OFF).

Interval Timer

6-12-2 PPG output

Use MCU timer to output PPG waveform with the leadingtiming value changeable as following:

(10%, 25%, 50%, 75%, 90%)

Power on make timer working in PPG mode and start the PPG output,

K1: Change LeadingTiming

Press K1 to change the leadingtiming:

(10% → 25% →50% → 75% → 90% → 10%)

LCD displays:

PPG Output LeadingTiming: 50%

6-12-3 Frequency measure

Use the MCU timer for measurement of input signal frequency.

Power on the timer will work in frequency measurement mode.

K3: Freq Measure

Press K3 to measure frequency of input signal.

LCD display:

Freq measure f=99999Hz

*__Note:__ PH6/TB3IN0 is the frequency measure input pin, and use pulse generator input any frequency.

6-13 WDT

The watchdog timer cannot be used in the STOP mode, SLEEP mode and SLOW mode where high-speed frequency clock is stopped. Before transition to these modes, the watchdog timer should be disabled.

Watch dog timer is enabled after reset, and is disabled in SystemInit().

The driver example step:

1) Initialize WDT by setting detection time and selecting WDT interrupt as counter overflow operation.
2) There are two demos for WDT.

   DEMO1:

   If timer is overflow then WDT will be cleared, and NMI interrupt is generated.

DEMO2:

WDT clear code will be written to the watchdog timer control register, and watch dog timer counter will be cleared.

7 Software

This software project creates the sample applications based on IAR TMPM330-SK Evaluation board and M330 demo board that will demonstrate the main feature of TMPM330 MCU.

Use current driver software and IAR EWARM or KEIL MDK to implement the sample applications. Create an independent project for each feature.

The workspace structure and project names are defined as following:

IAR EWARM:

```
├─WDT_NMI
    ├─APP
    │   main.c
    │   tmpm330_wdt_int.c
    │
    ├─TX03_CMSIS
    │   │   core_cm3.c
    │   │   core_cm3.h
    │   │   system_TMPM330.c
    │   │   system_TMPM330.h
    │   │   TMPM330.h
    │   │
    │   └─startup
    │       startup_TMPM330.s
    │
    ├─TX03_Periph_Driver
    │   ├─inc
    │   │       tmpm330_wdt.h
    │   │       tx03_common.h
    │   │
    │   └─src
    │           tmpm330_wdt.c
    └─TMPM330-SK
                key.c
                key.h
                lcd.c
                lcd.h
```

led.c
led.h

KEIL MDK:

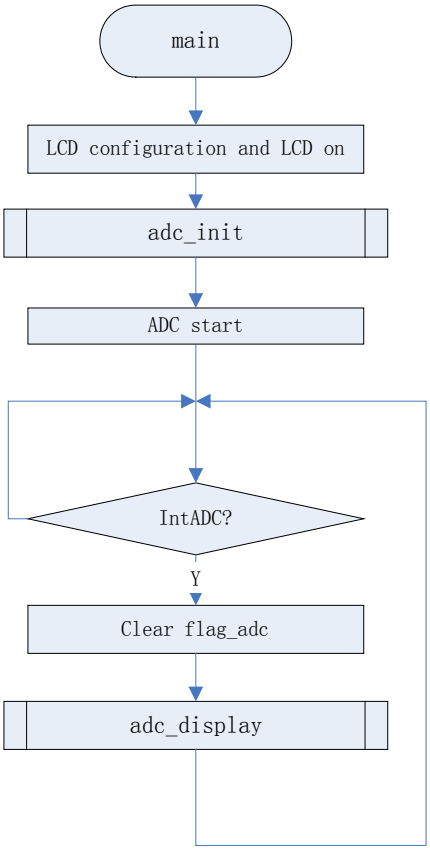
```
├─WDT_NMI
    ├─APP
    │   main.c
    │   tmpm330_wdt_int.c
    │
    ├─TX03_CMSIS
    │   core_cm3.c
    │   system_TMPM330.c
    │   startup_TMPM330.s
    │
    ├─TX03_Periph_Driver
    │   tmpm330_wdt.c
    │
    └─TMPM330-SK
            key.c
            lcd.c
            led.c
```

7-1 ADC

This is a simple example based on the TX03 Peripheral Driver (ADC).

The example includes:
1. ADC initialization.
2. Get ADC value and display it on LCD.

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize the ADC module in ADC_Init() function, the channel is set to ADC0, and interrupt is generated when AD conversion is finished. For example,

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

After initializing, start ADC.

```
ADC_Start();
```

When ADC conversion finished, “fIntADC” will be set in ADC ISR routine.
Then the ADC_Display() function is called when ADC Interrupt is generated. In ADC_Display() function, the API is used to read AD value.

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

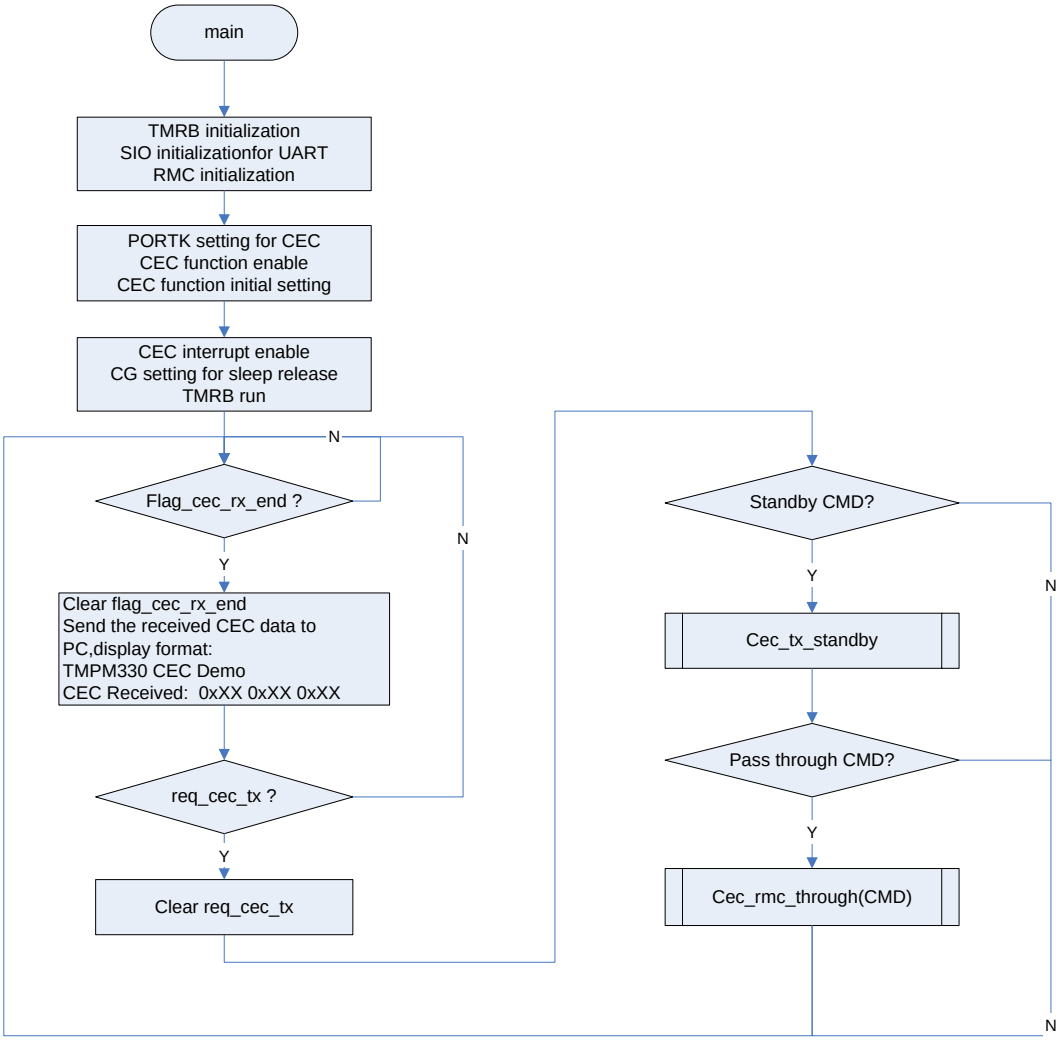
7-2 CEC

7-2-1 Example: CEC TRx

This is a simple example based on the TX03 Peripheral Driver (CEC, RMC, GPIO, UART, TMRB, CG).

The example includes:
1. CEC configuration and CEC initialization
2. CEC command send process
3. CEC data receive process

- **Flowchart**



- **Code and Explanation for the Example**

At first, configure GPIO for CEC.

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

Then enable, initialize and configure CEC channel.

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

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

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

gCEC_Command_Mode = CEC_CMD_MODE_IDLE;
```

Enable INTCECRX and INTCECTX.

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

After the above setting, start CEC Tx and Rx.
Output the CEC data to PC.

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

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

The data transfer is handled in INTCECTX.
The data receive is handled in INTCECRX.

7-3 CG

This is a simple example based on the TX03 Peripheral Driver (CG).

The example includes:
1. Basic setup operation of CG
2. How to switch between normal mode and slow mode

3. How to enable multiple clock circuit

- **Explanation for the Example**

Normal setup for CG (after reset)
The following code is just an example for setting CG in normal mode. It is supposed that the high-speed oscillator is 10MHz.

Example code is as the following:

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

Setup to enter slow mode
Set CG module: normal ->slow mode.
First, switch system clock to fs, then stop high-speed oscillation.

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

Setup to enter sleep mode
Set CG module: slow ->sleep mode.
First set source (RTC interrupt) to exit sleep mode.

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

Enable RTC interrupt, and clear interrupt request.

```
   NVIC_ClearPendingIRQ(INTRTC_IRQn);
   NVIC_EnableIRQ(INTRTC_IRQn);
   CG_ClearINTReq(CG_INT_SRC_RTC);
```

Set up real-time clock and alarm, and generate RTC interrupt after 2 minutes.

Enter sleep mode.

```
   _WFI();
```

* Interrupt is needed to exit sleep mode, and wait 2 minutes to generate RTC interrupt.

Setup to switch from slow mode to normal mode
Set CG module: slow -> normal
First start high-speed oscillator.

```
CG_SetFosc(ENABLE);
```

Then wait 1.953ms for fosc to warm up.

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

Check whether warm up ends or not.

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

Set system clock to clock-gear.

```
CG_SetFsysSrc(CG_FSYS_SRC_FGEAR);
```

Enable multiple clock circuit

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

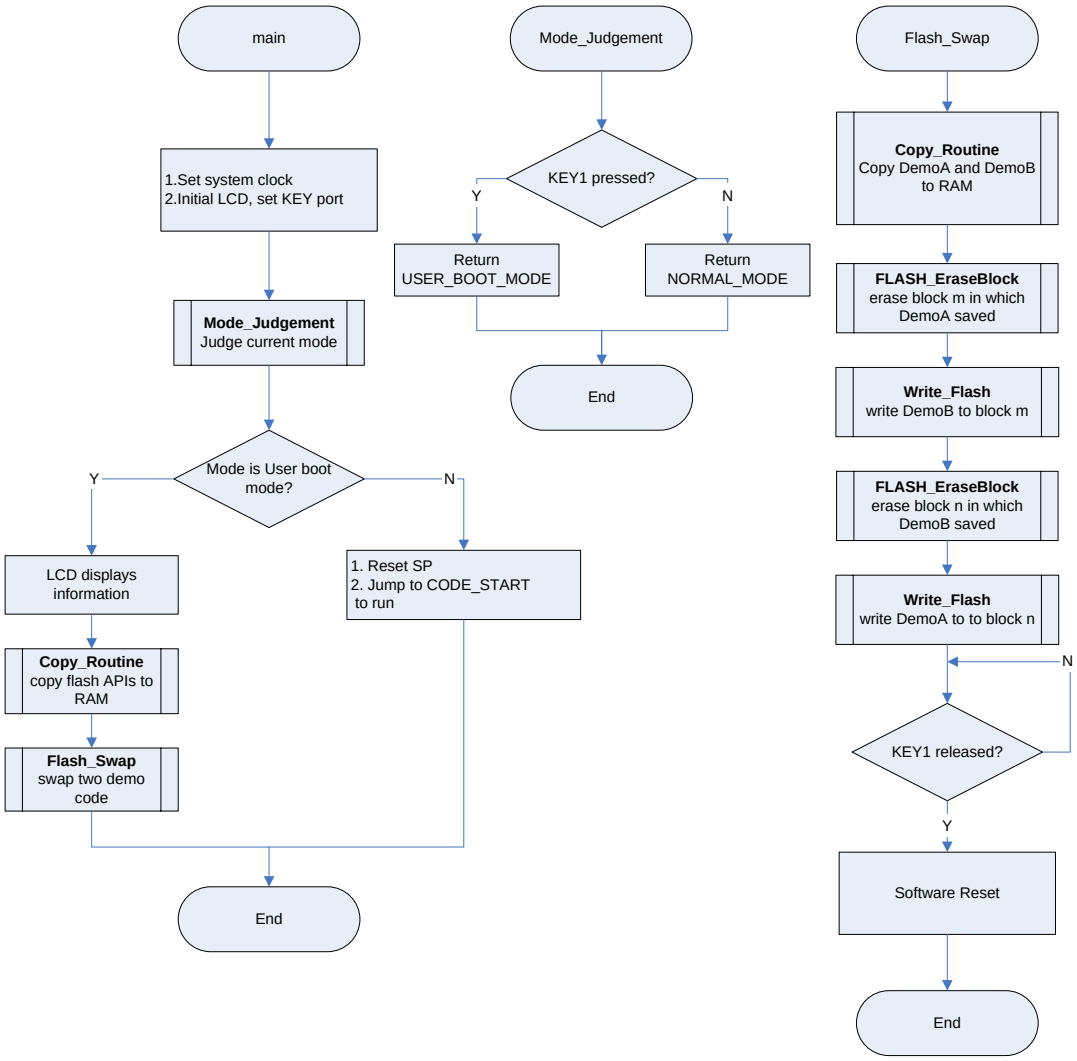
7-4 FLASH

This is a simple example based on the TX03 Peripheral Driver (FLASH, GPIO, and CG).

The example includes:

1. On-board programming method of Flash Memory (Rewrite/Erase)
2. Operation mode: Single chip mode (Normal mode and User boot mode)
3. Use User boot mode to update code in Flash Memory

- **Flowchart**



- **Code and Explanation for the Example**

At first configure system clock and initialize KEY port and LCD. KEY port (GPIO) is used to judge current mode when reset and LCD will display current running routine and Flash operation information.

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

Use function Mode_Judgement() to judge which mode will be entered after reset.

```
uint8_t Mode_Judgement(void)
{
   return (KEY_Get(KEY1)) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

If the KEY isn't pressed when reset, the routine will enter normal mode. Then the routine will reset SP and jump to address "CODE_START" to run. Demo A saved in at address "CODE_START" which belongs to block m, so Demo A will run (LED1~4 blink).

```
    #if defined ( __CC_ARM   )              /* RealView Compiler */
        ResetSP();                          /* reset SP */
```

```
#elif defined ( __ICCARM__ )            /* IAR Compiler */
     asm("MOV R0, #0");                  /* reset SP */
     asm("LDR SP, [r0]");
#endif
     startup = CODE_START;
     startup();                          /* jump to code start address to run */
```

If the KEY is pressed when reset, the routine will enter user boot mode. LCD will display information to user. Then the routine will copy APIs for flash operation from address “FLASH_API_ROM” in Flash memory to address “FLASH_API_RAM” in RAM, because Flash memory cannot erase/program itself when runs the code at the same time.

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

After copying Flash operation APIs to RAM, the routine will jump to function Flash_Swap(), this function has been copied from Flash memory to RAM by using Copy_Routine() above.

In function Flash_Swap(), firstly it will copy Demo A and B from Flash memory to RAM, then call function FLASH_EraseBlock() and Write_Flash() to erase and program Flash memory. The code of Demo A and B will be exchanged in Flash memory.

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

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

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

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

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

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

LCD displays the information to indicate the swap operation has completed. After KEY released, it will execute software reset by using SCB->AIRCR register. Then this demo will run again to enter normal mode, because Demo A and B have been exchanged, the address “CODE_START” has become the start address of Demo B, Demo B will run (LED5~8 blink).

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

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

Flash memory operation function FLASH_EraseBlock() will erase a specified block automatically. The block is specified by parameter “block_addr”. First, this function will check whether the parameter “block_addr” is illegal. Then it will use Flash driver FC_GetBlockProtectState() to check if the specified block is protected.

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

If the block is protected, it will return “ERR_PRO”, otherwise it will send automatic block erase command to erase the block.

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

Then the function will use Flash driver FC_GetBusyState() to monitor when erasing operation finished. At the same time, use a timeout counter to check if the operation is overtime.

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

Function Write_Flash() will call FLASH_WritePage() to write data automatically in one page. The process of this function is basically same as FLASH_EraseBlock() except the automatic page program command.

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

```
    source++;
  }
```

7-5 GPIO

This is a simple example based on the TX03 Peripheral Driver (GPIO).

The example includes:
1. GPIO initialization
2. Write data to GPIO

- **Explanation for the Example**

At first, use GPIO_Init() to configure GPIO to LED. Create a GPIO_InitTypeDef structure, then fill all the data fields. For example,

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

Then call GPIO_Init() function to initialize LED.

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

In the while() process, run the LED demo: LED on and LED off.

Turn on LED by using GPIO_WriteData ().

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

Turn off LED by using GPIO_WriteData ().

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

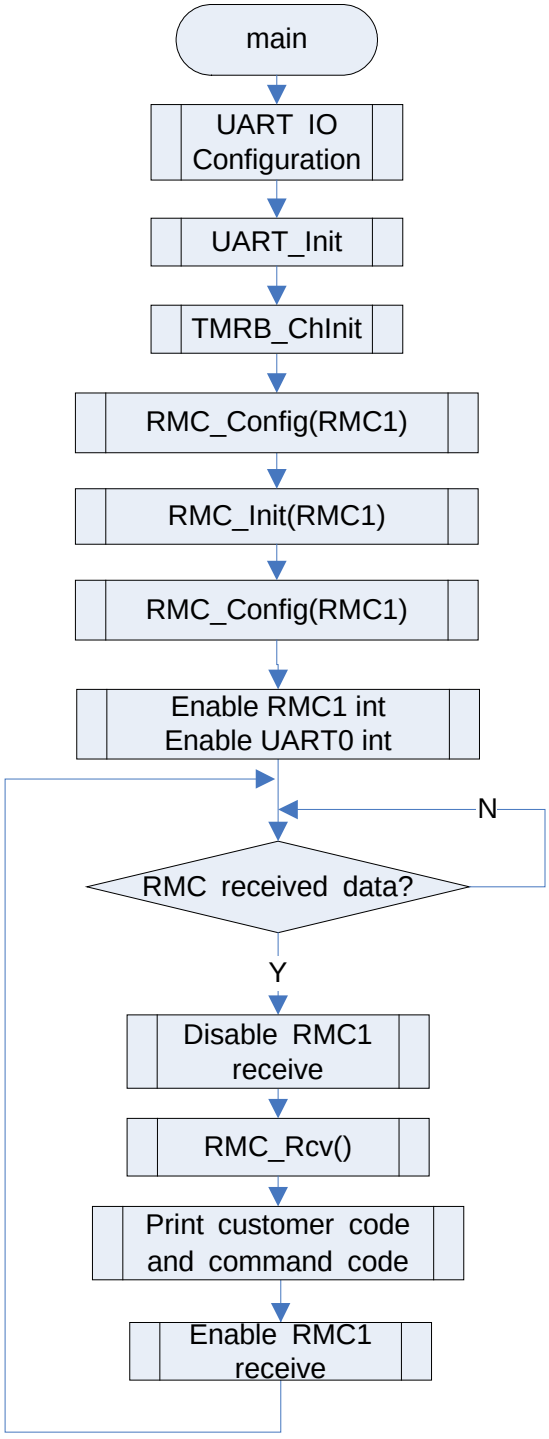
7-6 RMC

7-6-1 Example: RMC receiving

This is a simple example based on the TX03 Peripheral Driver (RMC, GPIO, UART, TMRB).

The example includes:
1. RMC configration and RMC initialization
2. RMC receiving process

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize the UART0, TMRB0 and RMC1 channel. UART0 and TMRB0's GPIO configure and initialize will be introduced in other chapter. Following only introduce the RMC1's GPIO configuration and initialization.

Configure the GPIO and initialize RMC1 channel.

Use GPIO peripheral drivers configure GPIO for RMC1.

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

Create a RMC_InitTypeDef structure and fill all the data fields. For example, RMC register set for TOSHIBA format.

```
    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;
```

Use RMC peripheral drivers to enable, initialize RMC1 channel.

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

After above setting, enable UART0 TX interrupt and RMC1 RX1 interrupt.

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

The process of data receiving will be finished in ISR routine.
RMC ISR routine:

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

    myRMC_LeaderDetection = RMC_GetLeader(TSB_RMC1);

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

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

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

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

```
                /* do nothing */
            }
        }
    }
}
```

Utilize printf() to output data which received by RMC to UART.

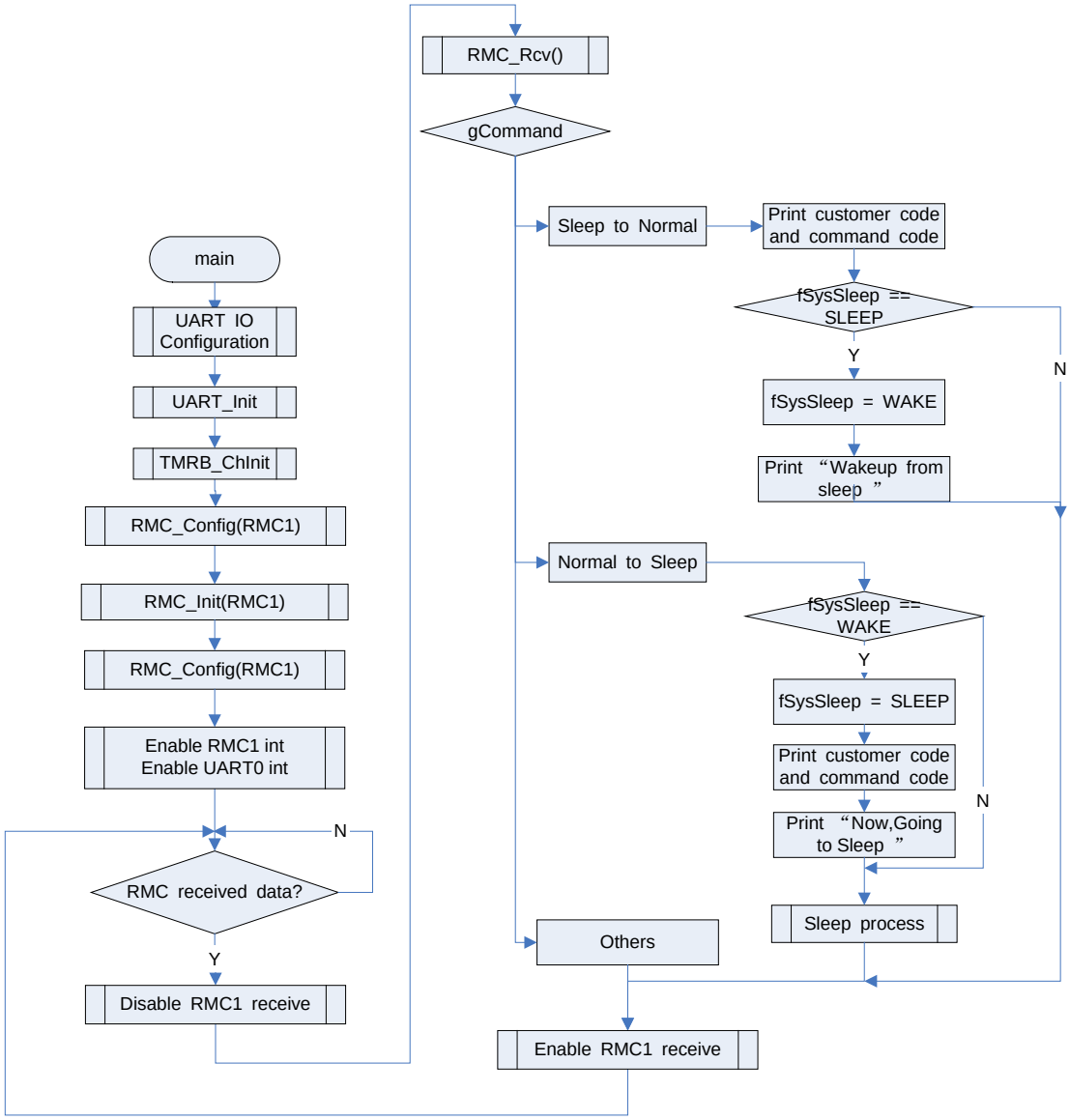
```
    printf("%s: %02x %02x\r\n", ch, addr, cmd);
```

7-6-2 Example: Power mode change

This is a simple example based on the TX03 Peripheral Driver (RMC, GPIO, UART, TMRB, CG).

The example includes:
1. RMC receive routine is same as RMC receiving module above
2. Change MCU CG operation mode from normal mode to sleep mode

- **Flowchart**



- **Code and Explanation for the Example**

RMC receive routine is same as above module. This chapter mainly introduces how to change MCU CG operation mode from normal mode to sleep mode.

Set source (RMC1 interrupt) to exit sleep mode.

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

Change MCU CG operation mode from normal mode to sleep mode.
Enable low-speed oscillator.

```
CG_SetFs(ENABLE);
```

Set warm-up time: wait 1.953ms for fosc to warm up then start warm up.

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

Check whether warm up ends or not.

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

Set warm-up time: wait 3.2768 ms, counter is X1.
Set low power consumption mode sleep.

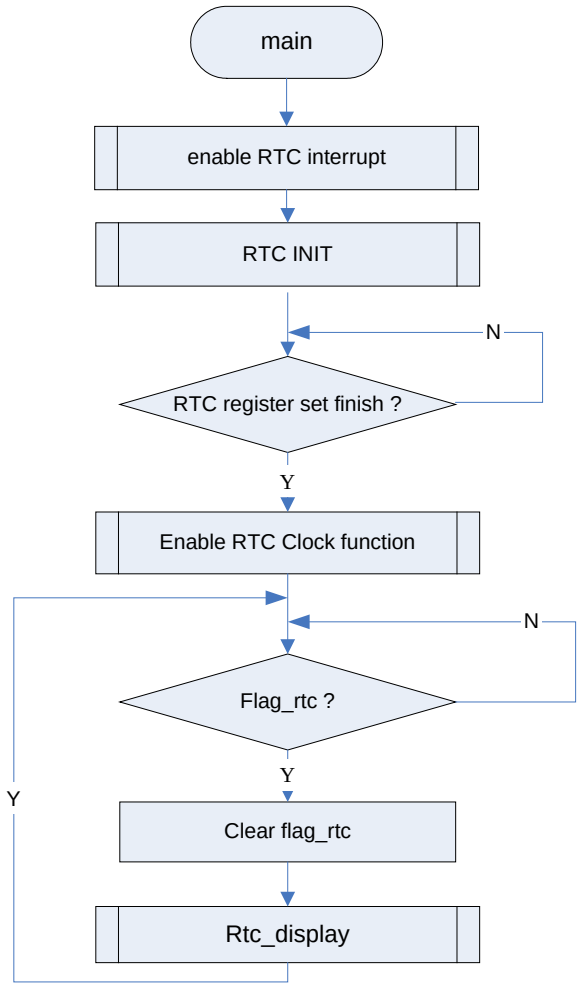
```
CG_SetWarmUpTime(CG_WARM_UP_SRC_X1, CG_WARM_UP_TIME_EXP_15);
CG_SetSTBYMode(CG_STBY_MODE_SLEEP);
```

7-6-3 RTC

This is a simple example based on the TX03 Peripheral Driver (RTC, CG).

The example includes:
1. RTC initialization
2. Get RTC date and time

- **Flowchart**



- **Code and Explanation for the Example**

At first set source (RTC interrupt) to exit sleep mode.

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

Initialize RTC .Create a RTC_DateTypeDef and RTC_TimeTypeDef structures and fill all the data fields. For example, initial setting: 2010/10/22 12:50:55, 24hours format.

```
RTC_DateTypeDef Date_Struct;

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

RTC_TimeTypeDef Time_Struct;

Time_Struct.HourMode = RTC_24_HOUR_MODE;
Time_Struct.Hour = (uint8_t) 12U;
Time_Struct.Min = (uint8_t) 50U;
Time_Struct.Sec = (uint8_t) 55U;
```

Disable clock and alarm function.

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

Reset RTC second counter, enable 1Hz interrupt, enable RTCINT.

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

Set RTC time and date value

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

After the setting above, enable RTC interrupt. Wait for 1second for RTC set finished, and then enable RTC Clock function.

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

In RTC ISR routine, the RTC interrupt will happen every second. Then RTC interrupt request will be cleared.

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

After interrupt, RTC date and time value will be sent to LCD.
Following is shown how to get RTC date and time value.

```
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();
```

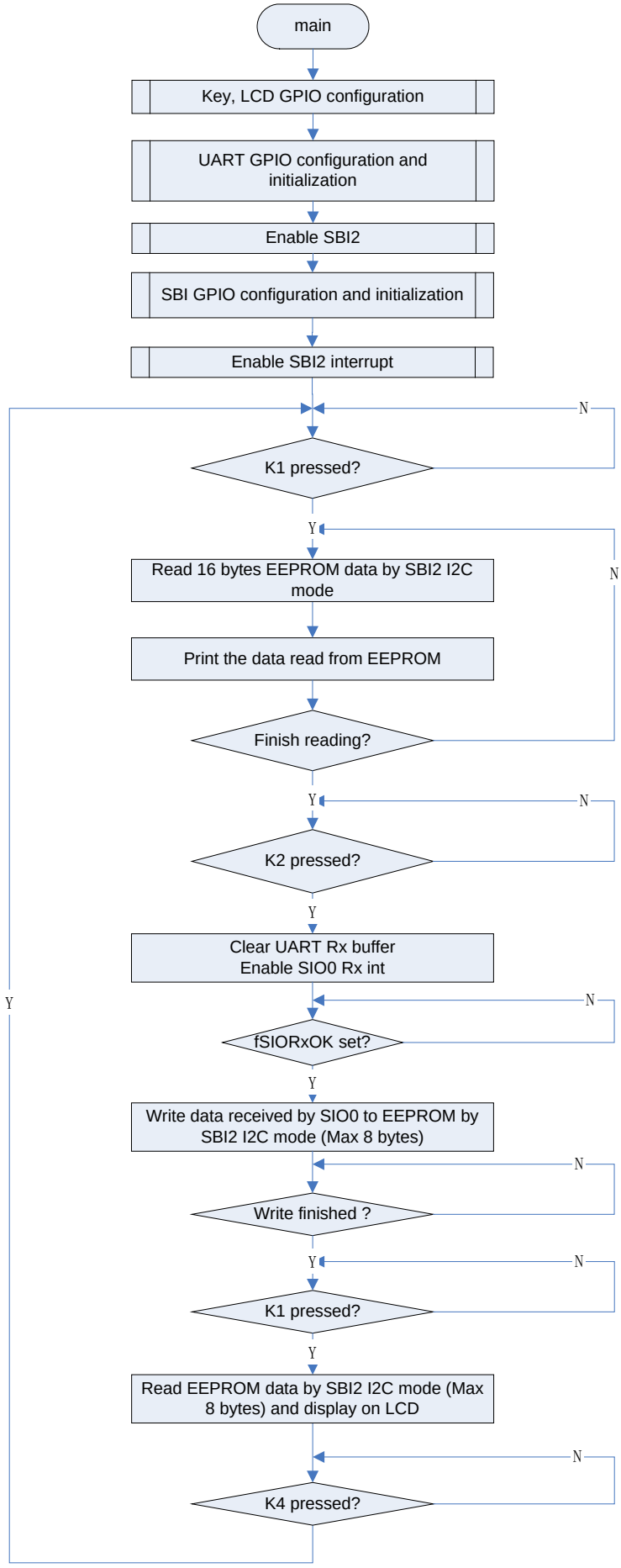
7-7 SBI

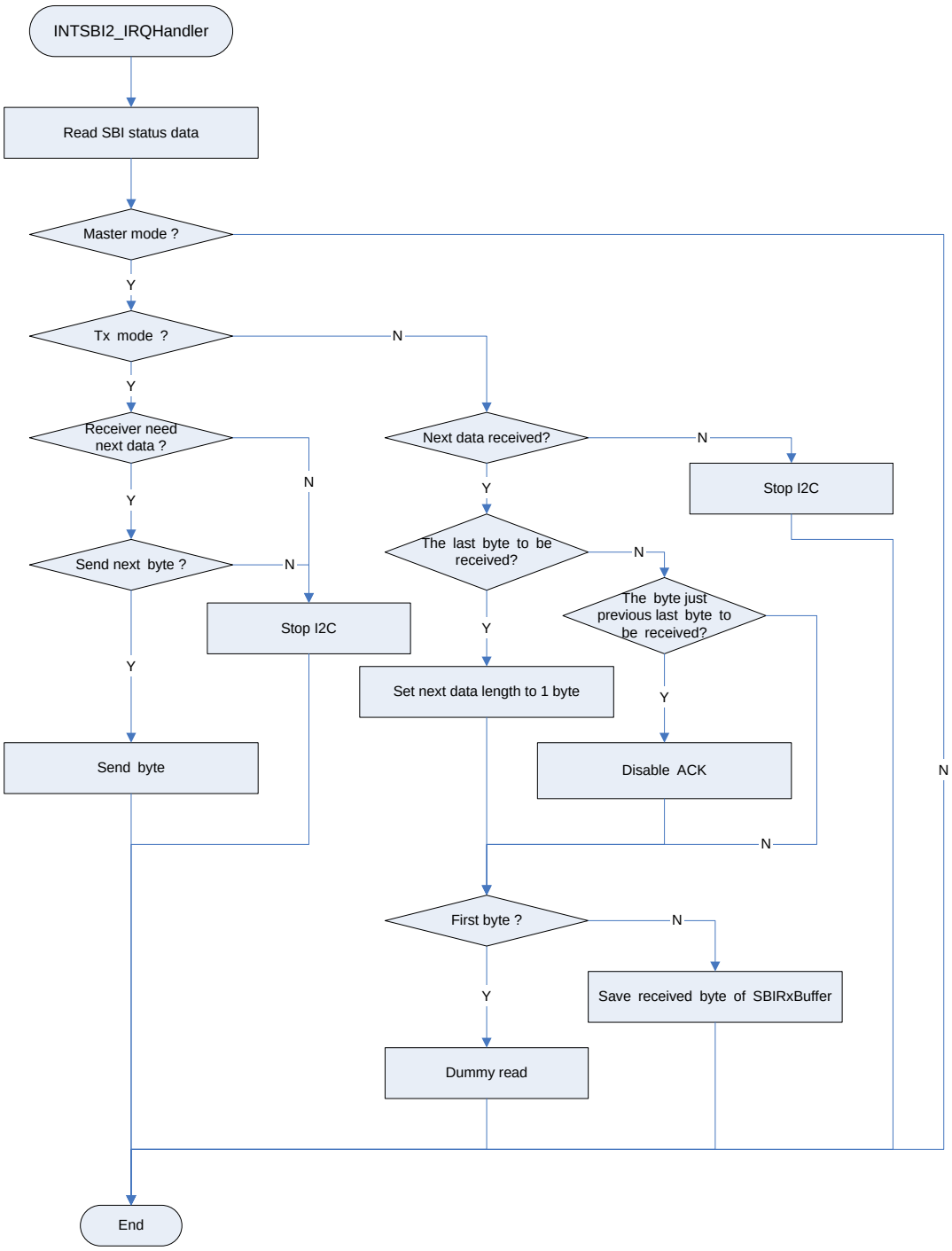
7-7-1 Example: EEPROM Read/Write

This is a simple example based on the TX03 Peripheral Driver (SBI, GPIO, UART).

The example includes:
1. SBI configuration and I2C initialization
2. I2C send data process

- **Flowchart**


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



- **Code and Explanation for the Example**

At first, configure GPIO for SBI2 I2C mode.

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

Then enable, initialize and configure SBI2 channel and enable INTSBI2.

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

After above setting, start EEPROM read.
Set I2C buffer, and then start I2C transmission to send EEPROM device address and the address to be read.
Send EEPROM device address: Set the deviece address to SBI Tx buffer and direction is “SBI_I2C_SEND”. Use SBI_GenerateI2CStart(TSB_SBI2) function to start I2C process.

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

The read address of EEPROM is sent in INTSBI2.

After sending EEPROM device address and read address, then start to send data to EEPROM device address to be read, direction is “SBI_I2C_RECEIVE”.

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

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

The EEPROM read process is started, and the I2C data receive is handled in INTSBI2 function.

After EEPROM data read finished, start to write data that is received from UART to EEPROM.
Initialize the SBI Tx buffer and length, check if the I2C bus is free or not, then set EEPROM address and direction “SBI_I2C_SEND”. Use SBI_GenerateI2CStart(TSB_SBI2) function to start I2C process.

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

The EEPROM write process is started, and write the remain data of gI2C_TxData[] in INTSBI2 function.

In INTSBI2 function, I2C master send and slave receive process are handled according to I2C bus state.
During I2C master sending, SBI_SetSendData() is used to send next data, SBI_GenerateI2CStop() is used to stop I2C when I2C process is finished.
During I2C slave receiving, SBI_GetReceiveData() is used to read I2C data received in SBI data buffer, SBI_SetI2CACK() is used to not generate ACK after next data received, SBI_SetI2CBitNum() is used to set only 1 clock for I2C stop, SBI_GenerateI2CStop() is used to stop I2C when I2C process is finished, at the same time, SBI_SetI2CACK() is used to re-set I2C with ACK mode.

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

   SBIx = TSB_SBI2;
   sbi_sr = SBI_GetI2CState(SBIx);

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

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

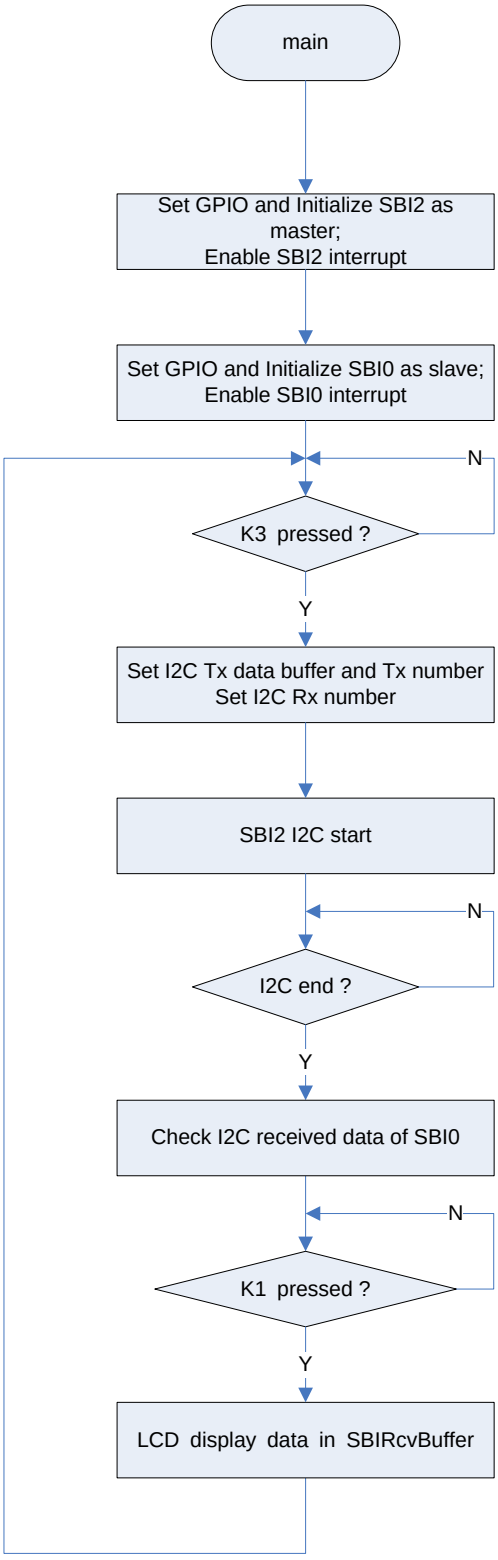
7-7-2 Example: I2C Slave

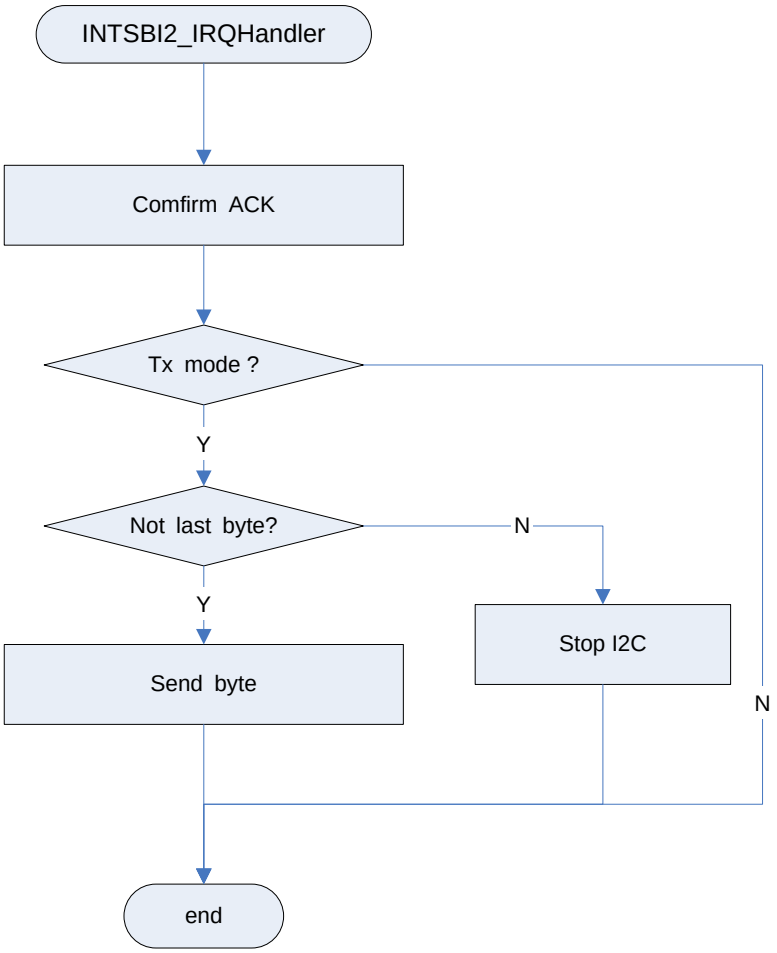
This is a simple example based on the TX03 Peripheral Driver (SBI, GPIO).

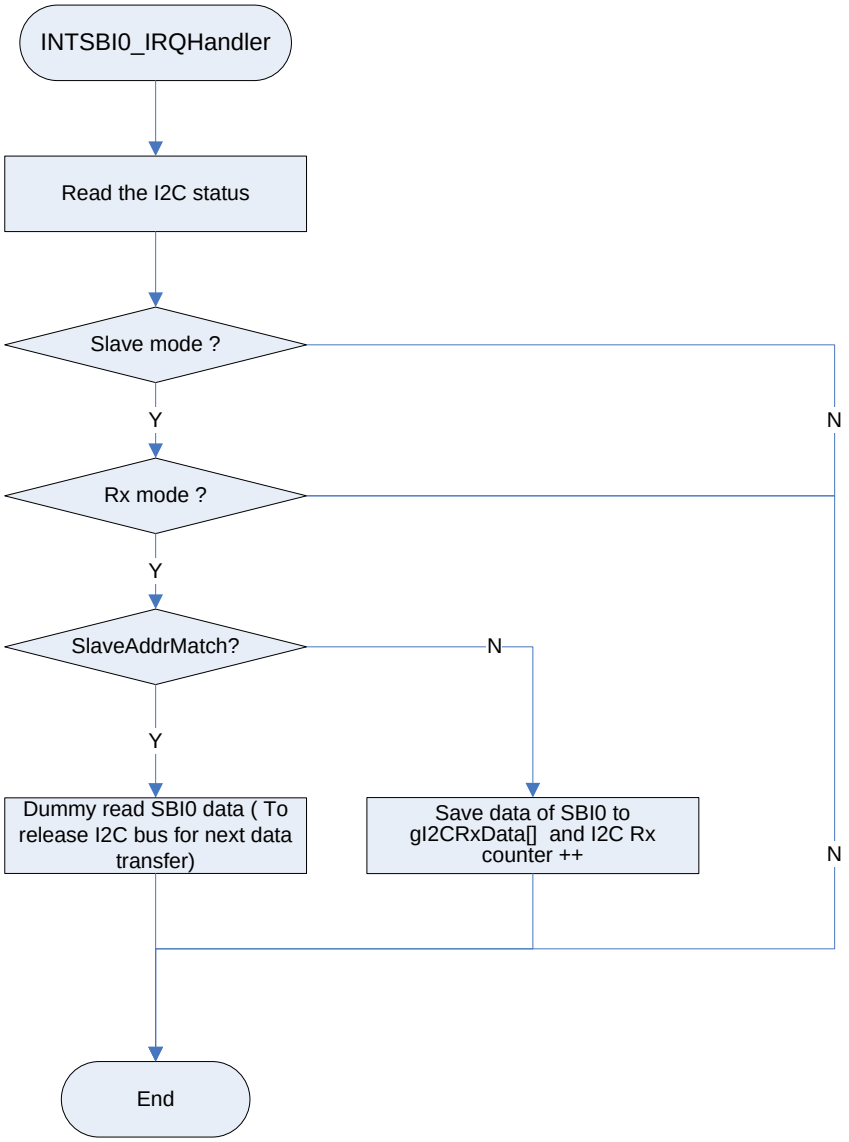
The example includes:

1. SBI configuration and I2C initialization
2. I2C master send data process
3. I2C slave receive data process

- **Flowchart**


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


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



- **Code and Explanation for the Example**

At first, configure GPIO for SBI2 I2C mode.

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

Configure GPIO for SBI0 I2C mode.

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

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

Then enable, initialize and configure SBI2 channel and enable INTSBI2.

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

Initialize and configure SBI0 channel and enable INTSBI0.

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

After the above setting, start I2C send.
Clear I2C Rx buffer, initialize the SBI Tx buffer and length, and clear Rx buffer.

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

Check if the I2C bus is free or not, SBI_SetSendData() is used to set data “SLAVE_ADDR”.and direction is “SBI_I2C_SEND” to SBI data buffer; then SBI_GenerateI2CStart(TSB_SBI2) is used to to start I2C process.

```
    /* Check I2C bus state and start TRx */
case MODE_SBI_I2C_START:
    i2c_state = SBI_GetI2CState(TSB_SBI2);
    if (!i2c_state.Bit.BusState) {
        SBI_SetSendData(TSB_SBI2, SLAVE_ADDR | SBI_I2C_SEND);
        SBI_GenerateI2CStart(TSB_SBI2);
        gSBIMode = MODE_SBI_I2C_TRX;
    } else {
        /* Do nothing */
    }
```

```
    break;
```

The data transfer is handled in INTSBI2.

The data receive is handled in INTSBI0.

In INTSBI2 function, I2C master send process is handled according to I2C bus state. During I2C master sending, SBI_SetSendData() is used to send next data, SBI_GenerateI2CStop() is used to stop I2C when I2C process is finished.

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

  SBIx = TSB_SBI2;
  sbi_sr = SBI_GetI2CState(SBIx);

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

In INTSBI0 function, I2C slave receive process is handled according to I2C bus state, SBI_GetReceiveData() is used to read received data in SBI buffer, I2C stop condition is controlled by master.

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

  SBIy = TSB_SBI0;
  sbi0_sr = SBI_GetI2CState(SBIy);

  if (!sbi0_sr.Bit.MasterSlave) {     /* Slave mode */
    if (!sbi0_sr.Bit.TRx) { /* Rx Mode */
      if (sbi0_sr.Bit.SlaveAddrMatch) {
        /* First read is dummy read for Slave address recognize */
        tmp = SBI_GetReceiveData(SBIy);
        gI2CRCnt = 0U;
      } else {
        /* Read I2C received data and save to I2C_RxData buffer */
        tmp = SBI_GetReceiveData(SBIy);
        gI2CRxData[gI2CRCnt] = tmp;
        gI2CRCnt++;
```

```
        }
      } else {              /* Tx Mode */
        /* Do nothing */
      }
    } else {                /* Master mode */
      /* Do nothing */
    }
}
```

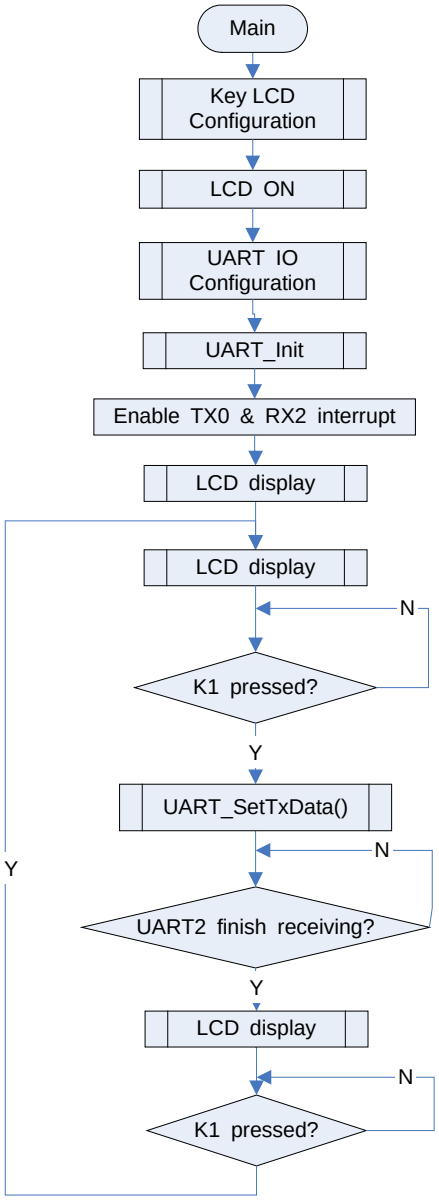
7-8 SIO

7-8-1 Example: Transmission from UART0 to UART2

This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:
1. UART configuration and initialization
2. UART TRX process
3. Use UART0 TX interrupt to send data
4. Use UART2 RX interrupt to receive data that from UART0

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART.
Use GPIO peripheral drivers configure GPIO for UART0.

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

Configure GPIO for UART2.

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

Create a UART_InitTypeDef structure and fill all the data fields. For example,

```
UART_InitTypeDef myUART;
```

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

Use UART peripheral drivers to enable and initialize UART0/2 channels.

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

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

After above setting, enable UART0 TX interrupt and UART2 RX interrupt.

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

Then start sending data. Here TxBuf is a character array.

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

The rest process of data flow is finished in ISR of UART0 TX interrupt routine and UART2 RX interrupts routine.

UART0 TX interrupt routine:

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

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

UART2 RX interrupt routine:

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

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

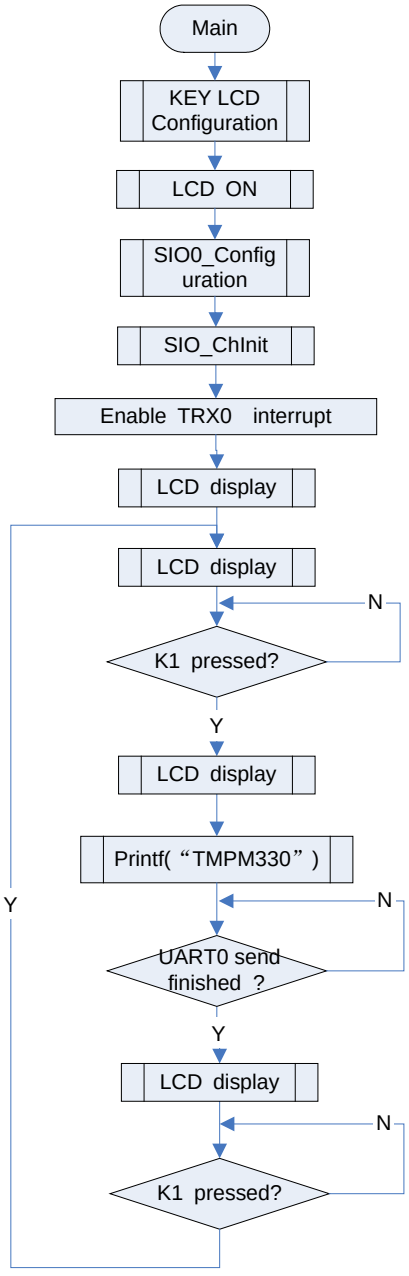
7-8-2 Example: Retarget

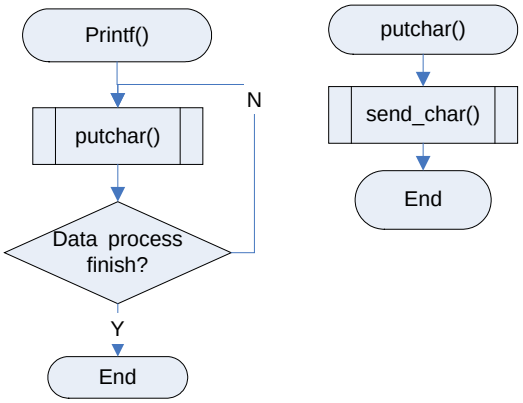
This is a simple example based on the Peripheral Driver (UART , GPIO).

The example includes:

1. UART configuration and initialization
2. UART TX process
3. Use UART0 TX interrupt to send data
4. Retarget printf() to UART0

- **Flowchart**


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



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART0.
Use GPIO peripheral drivers configure GPIO for UART0.

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

Create a UART_InitTypeDef structure and fill all the data fields. For example,

```
UART_InitTypeDef myUART;

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

Use UART peripheral drivers enable and initialize UART0.

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

After above setting, enable UART0 TX interrupt.

```
  NVIC_EnableIRQ(INTTX0_IRQn);
```

Use printf() to output to UART0. Here TxBuf is a character array.

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

Function printf() will call putchar() for IAR Compiler and fputc() for RealView Compiler to output data to UART0.

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

#elif defined ( __ICCARM__ )      /*IAR Compiler */
   int putchar(int ch)
   #endif
{
```

```
    return (send_char(ch));
}

uint8_t send_char(uint8_t ch)
{
    while (gSIORdIndex != gSIOWrIndex) {        /* wait for finishing sending */
        /* do nothing */
    }
    gSIOTxBuffer[gSIOWrIndex++] = ch;   /* fill TxBuffer */
    if (fSIO0_INT == CLEAR) {   /* if SIO0 INT disable, enable it */
        fSIO0_INT = SET;        /* set SIO0 INT flag */
        UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);
        NVIC_EnableIRQ(INTTX0_IRQn);
    }

    return ch;
}
```

The rest process of data flow is finished in ISR of UART0 TX interrupt.

UART0 TX interrupt routine:

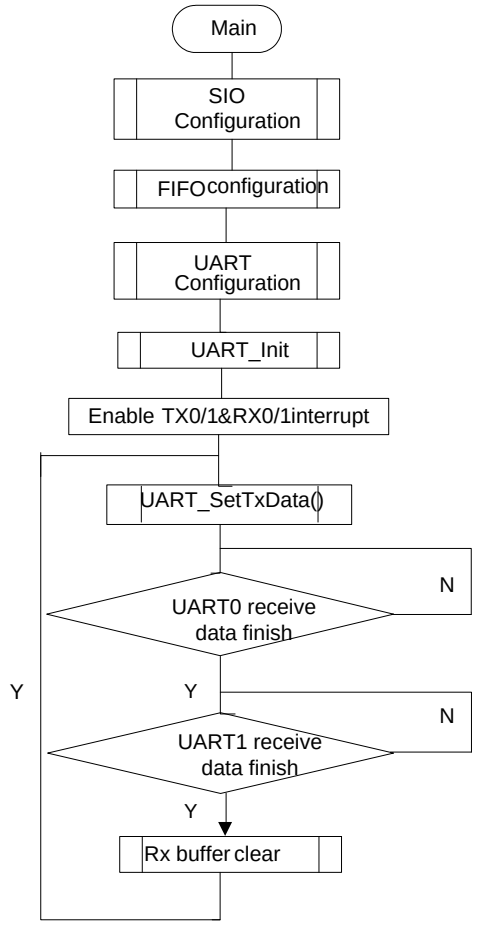
```
void INTTX0_IRQHandler(void)
{
    if (gSIORdIndex < gSIOWrIndex) {    /* buffer is not empty */
         UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);    /* send data */
        fSIO0_INT = SET;        /* SIO0 INT is enable */
    } else {
        /* disable SIO0 INT */
        fSIO0_INT = CLEAR;
        NVIC_DisableIRQ(INTTX0_IRQn);
        fSIOTxOK = YES;
    }
    if (gSIORdIndex >= gSIOWrIndex) {   /* reset buffer index */
        gSIOWrIndex = CLEAR;
        gSIORdIndex = CLEAR;
    } else {
        /* do nothing */
    }
}
```

7-8-3 Example: UART FIFO

This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:
1.UART and FIFO configuration and initialization.
2.UART send and receive data use FIFO process.

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART.

Use GPIO peripheral drivers configure GPIO for UART0 and UART1.

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

Create a UART_InitTypeDef structure and fill all the data fields. For example,

```
   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);
```

Use UART peripheral drivers to enable and initialize UART0/1 channels.

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

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

FIFO configuration.

```
   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);
```

After above setting, enable UART0/1 interrupt.

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

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

The rest process of data flow is finished in ISR of UART0/1 RX and TX interrupt routine

UART0 TX interrupt routine:

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

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

UART1 TX interrupt routine:

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

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

UART0 RX interrupt routine:

```
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 RX interrupt routine:

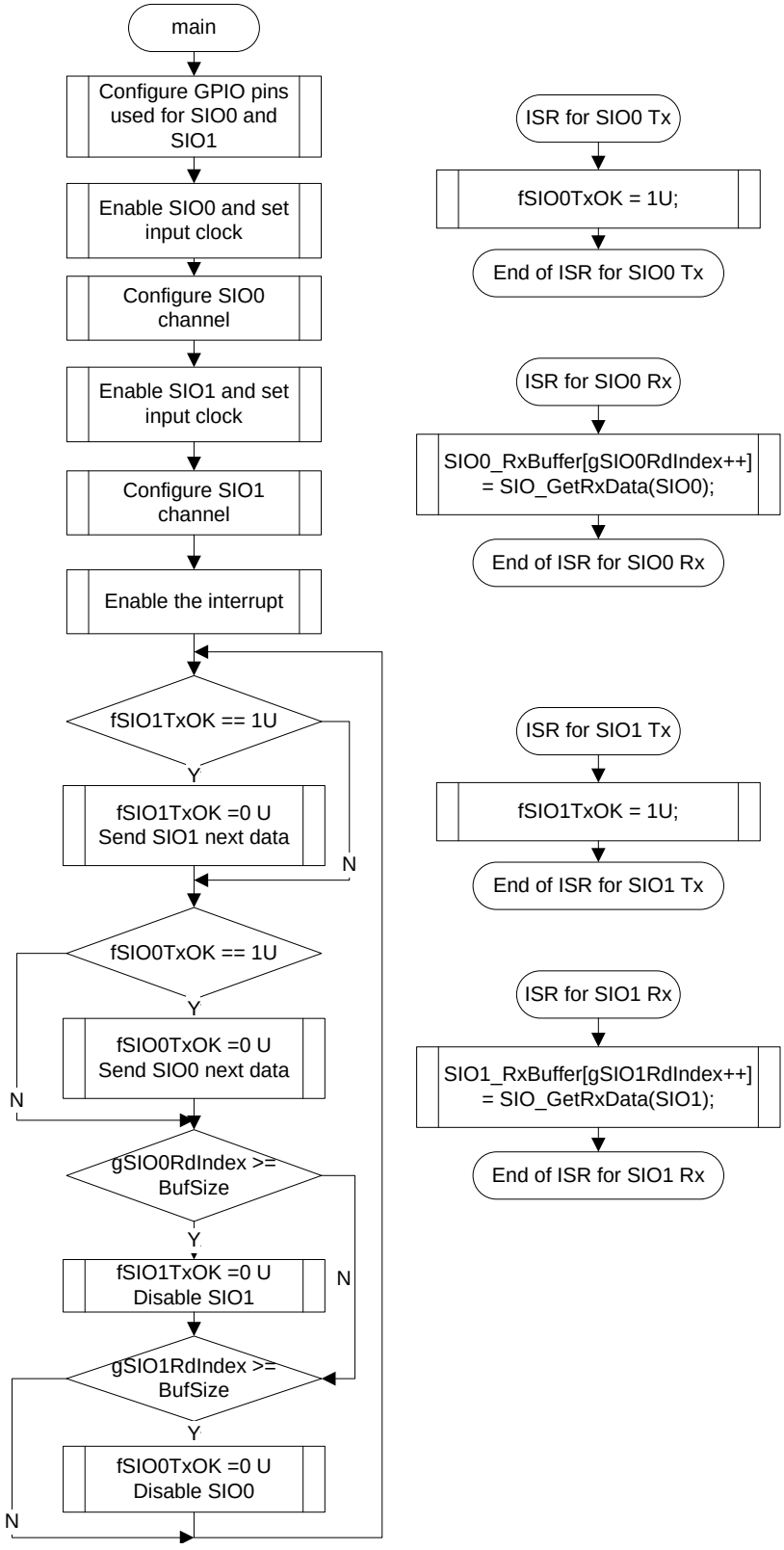
```
void INTRX1_IRQHandler(void)
{
   volatile UART_Err err;

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

7-8-4 Example: SIO

This is a simple example based on the TX03 Peripheral Driver (SIO).

The example includes:
1.Basic setup operation of SIO
2.The data transfer between SIO0 and SIO1
3.SIO interrupt of Tx and Rx

- **Flowchart**



- **Code and Explanation for the Example**

  At first, configure the GPIO pins for SIO by setting the CR, FR1, IE register of proper port.

Then, enable SIO0 configure the input clock and SIO0 initialization structure.

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

/*initialize the SIO0 struct */
SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_1;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_T4;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;

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

Enable SIO1 configure the input clock and SIO1 initialization structure.

```
/*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);
```

Enable the interrupt of SIO TX, RX.

```
    /* 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);
```

After all the basic configure, Enter the data transfer routine .

```
    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 */
    }   }
```

In ISR of SIO0 Tx, set transfer is ok.

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

In ISR of SIO0 Rx, get the receive data from RX buffer

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

In ISR of SIO1 Tx, set transfer is ok.

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

In ISR of SIO1 Rx, get the receive data from RX buffer.

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

7-9 TMRB

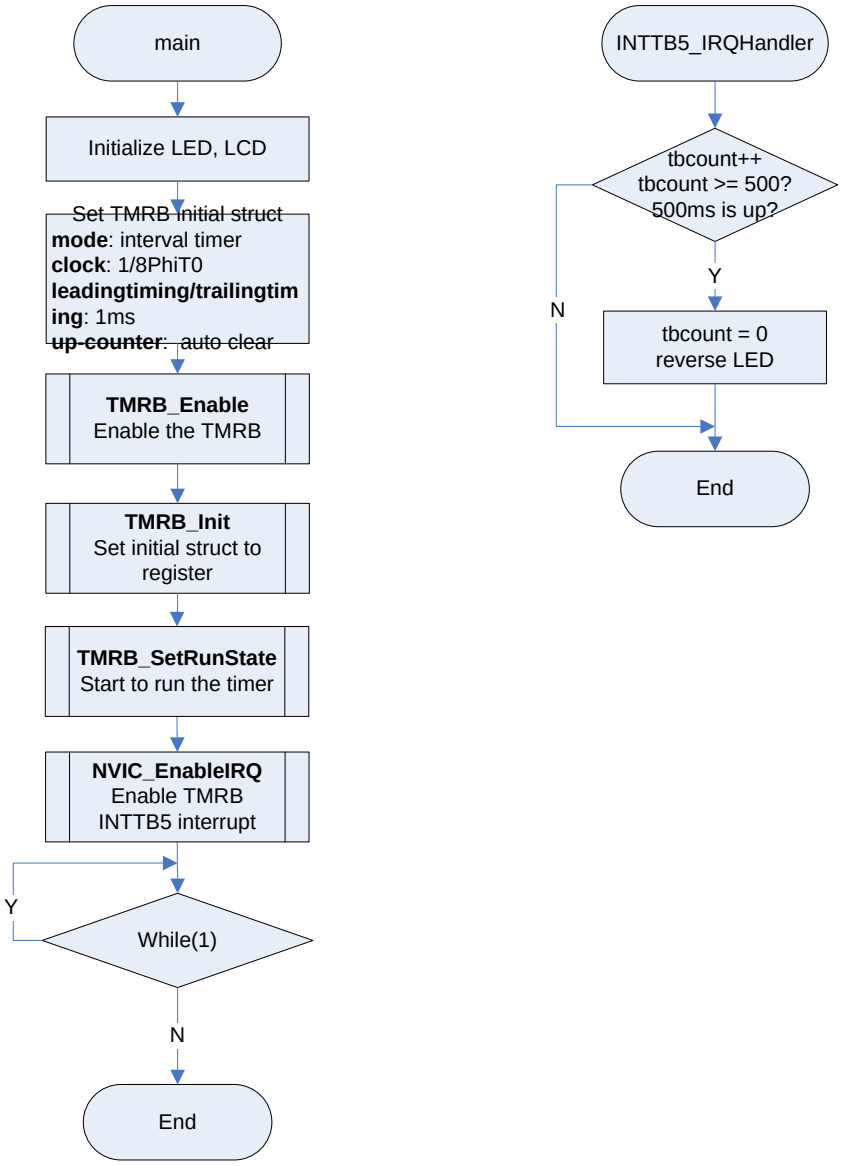
7-9-1 Example: Interval Timer

This is a simple example based on the TX03 Peripheral Driver (TMRB, GPIO).

The example includes:

1. GPIO configuration and TMRB5 initialization
2. Interval Timer for 1ms
3. LED ON/OFF for 1Hz

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize LED channel on SK board and turn on LED.

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

Prepare TMRB initialization structure, and then fill TMRB mode, clock, up-counter clear method, trailingtiming and leadingtiming. This demo will set trailingtiming and leadingtiming to 1ms, macro TMRB_1ms equals 0x1388, because fphiT0 = fsys = fc = 10MHz * PLL = 40MHz, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us, 1ms/0.2us = 5000 = 0x1388 (Please see CG part for more detail information about the clock setting)

```
TMRB_InitTypeDef myTMRB;

myTMRB.Mode = TMRB_INTERVAL_TIMER;    /* internal timer */
```

```
    myTMRB.ClkDiv = TMRB_CLK_DIV_8;            /* 1/8PhiT0 */
    myTMRB.TrailingTiming = TMRB_1ms;                  /* trailingtiming is 1ms */
    myTMRB.UpCntCtrl = TMRB_AUTO_CLEAR;    /* up-counter auto clear */
    myTMRB.LeadingTiming = TMRB_1ms;                   /* leadingtiming time is
1ms */
```

Enable the TMRB module and set the initialization structure to specified registers. Enable INTTB5 interrupt, this interrupt will be triggered every 1ms, in the end, start the TMRB5 to run.

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

Then the main routine will enter “While(1)” to wait the interrupt happens. In interrupt routine, use a counter to count. If 500ms is up, reverse the LED and count again.

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

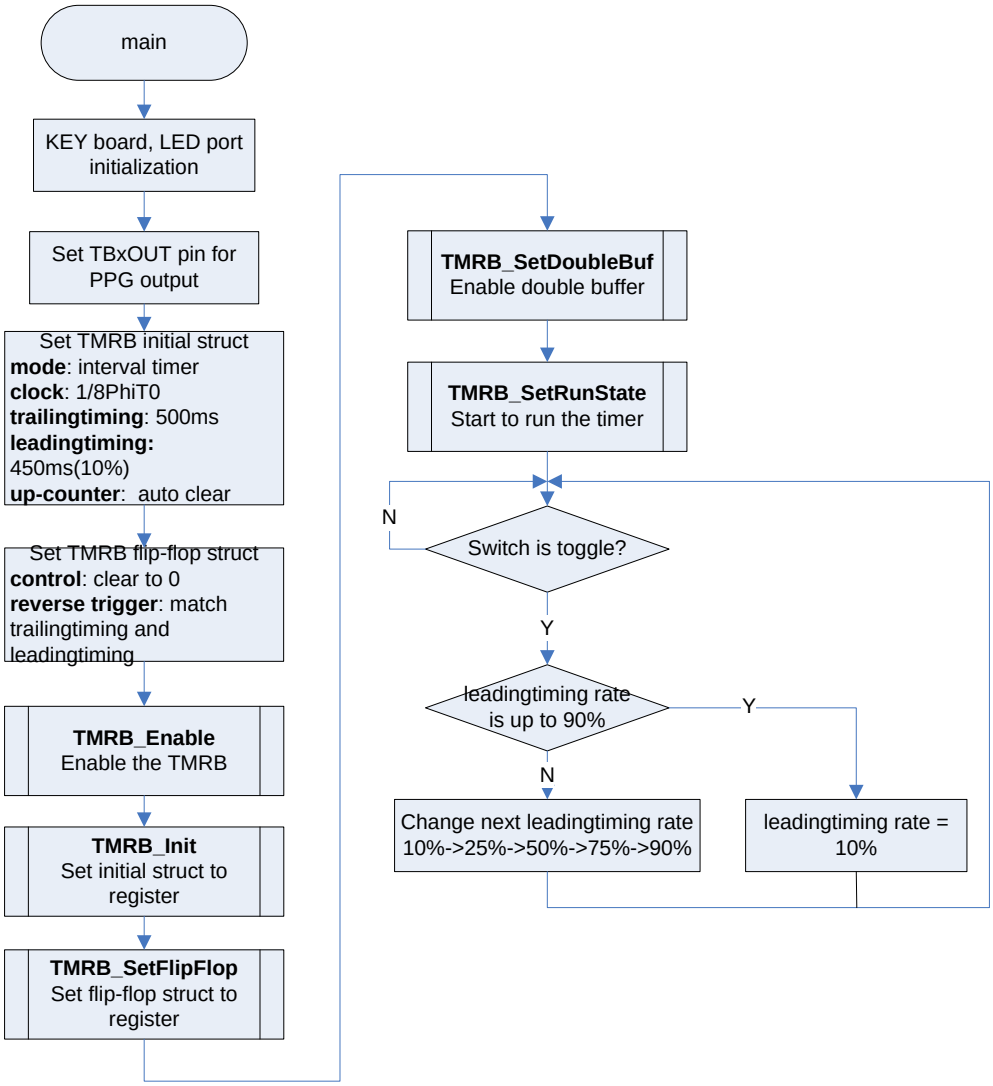
7-9-2 Example: PPG Output

This is a simple example based on the TX03 Peripheral Driver (TMRB, GPIO).

The example includes:

1. GPIO configuration and TMRB0 initialization
2. PPG process setting and start
3. PPG leadingtiming adjustment

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize keyboard and LED, LCD, set PI0 as TB0OUT for PPG output.

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

    GPIO_WriteData(GPIO_PG, tmp);

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

Prepare TMRB initialization structure, and then fill TMRB mode, clock, up-counter clear method, trailingtiming and leadingtiming into it. This demo will set trailingtiming to 500us, macro TMRB0TIME equals 0x09C4, because fphiT0 = fsys = fc = 10MHz * PLL = 40MHz, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us, 500us/0.2us = 2500 = 0x09C4 (Please see CG part for more detail information about the clock setting)

```
    TMRB_InitTypeDef myTMRB;

    myTMRB.Mode = TMRB_INTERVAL_TIMER;     /* internal timer */
    myTMRB.ClkDiv = TMRB_CLK_DIV_8;        /* 1/8PhiT0 */
```

```
    myTMRB.TrailingTiming = TMRB0TIME;                    /* trailingtiming is 500us
*/
    myTMRB.UpCntCtrl = TMRB_AUTO_CLEAR;    /* up-counter auto clear */
    myTMRB.LeadingTiming = LeadingTiming[0];                    /*
leadingtiming, initial value 10% */
```

Set flip-flop initialization structure, and fill flip-flop control, reverse trigger parameter into it. Reverse trigger is set to match leadingtiming and match trailingtiming.

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

Enable the TMRB module and set the initialization structure and flip-flop structure to specified registers. Enable double buffer, and disable capture function. In the end, start the TMRB to run.

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

Wait the button pressed and released.

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

If the button is released, change the leadingtiming according to 10%->25%->50%->75%->90%, then from 90% to 10% again.

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

    TMRB_ChangeLeadingTiming(TSB_TB0, LeadingTiming[Rate]);   /* change
leadingtiming rate */
```

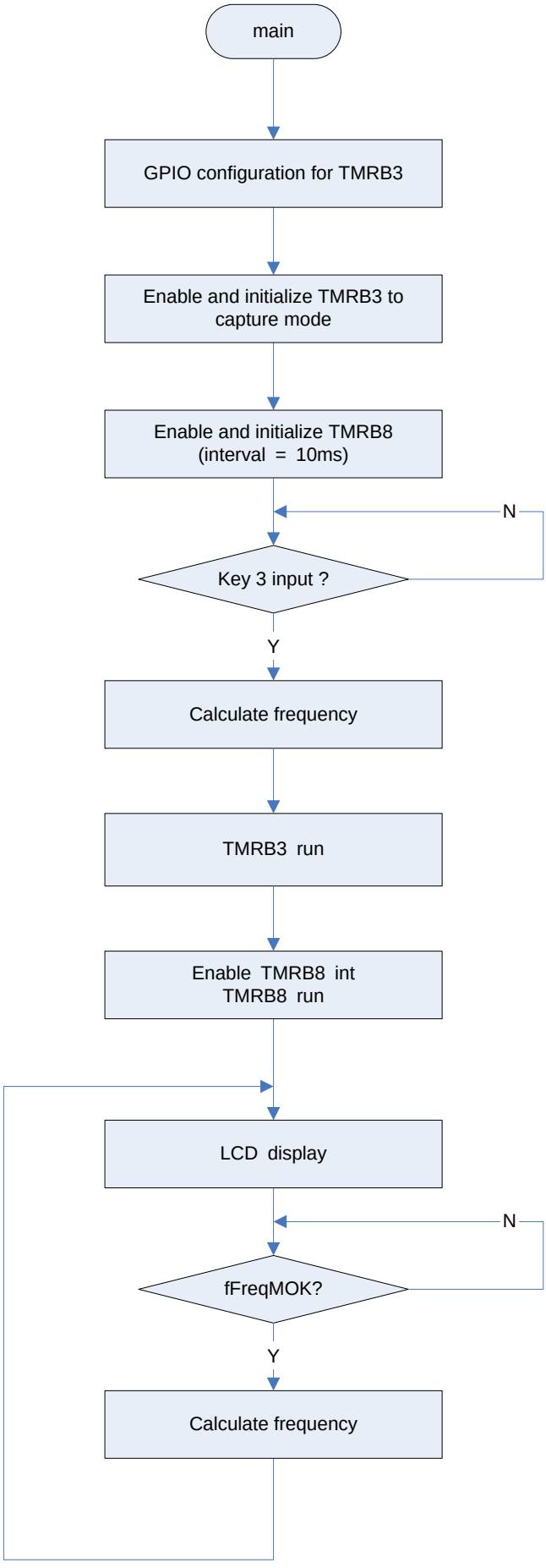
7-9-3 Example: Frequency measure

This is a simple example based on the TX03 Peripheral Driver (TMRB, GPIO).

The example includes:

1. TMRB configuration and TMRB initialization
2. Frequency calculation method

- **Flowchart**


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

- **Code and Explanation for the Example**

At first, configure GPIO for TB3IN0.

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

Then enable, initialize and configure TMRB3 and TMRB8.

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

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

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

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

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

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

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

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

If K3 is inputted, run TMRB3 and TMRB8.

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

If frequency measures mode and finish measuring, calculate average value of frequency.

Measure the frequency in INTTB8, if the mearsure time is matched, output fFreqMOK = true.

```
void INTTB8_IRQHandler(void)
{
   static uint8_t tb8count = 0U;
   TMRB_INTFactor TMRB_Status;
   uint16_t eTMRB_CP0;
   uint16_t eTMRB_CP1;

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

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

7-10 WDT

This is a simple example based on the TX03 Peripheral Driver (WDT, GPIO).

The example includes:

1. WDT initialization.
2. In DEMO1 WDT isn't cleared before timer overflow, NMI interrupt is generated.
3. In DEMO2 WDT is cleared before timer overflow, the LED will blink all the time.

- **Code and Explanation for the Example**

The following code is an example for initializing WDT. Detect time is set to 2^25/fsys, and interrupt will be generated when timer overflow.

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

Initialize WDT and enable it.

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

In DEMO1 Waiting for the NMI interrupt.

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

In DEMO1 When NMI interrupt is occurred the WDT will be disabled and the LED blink will be stopped.

```
WDT_Disable();
```

In DEMO2 WDT will be cleared and the LED will blink all the time.

```
WDT_WriteClearCode();
```