TOSHIBA

TX03 Peripheral Driver Usage Example (TMPM366)

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M366UE-01E

RESTRICTIONS ON PRODUCT USE

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

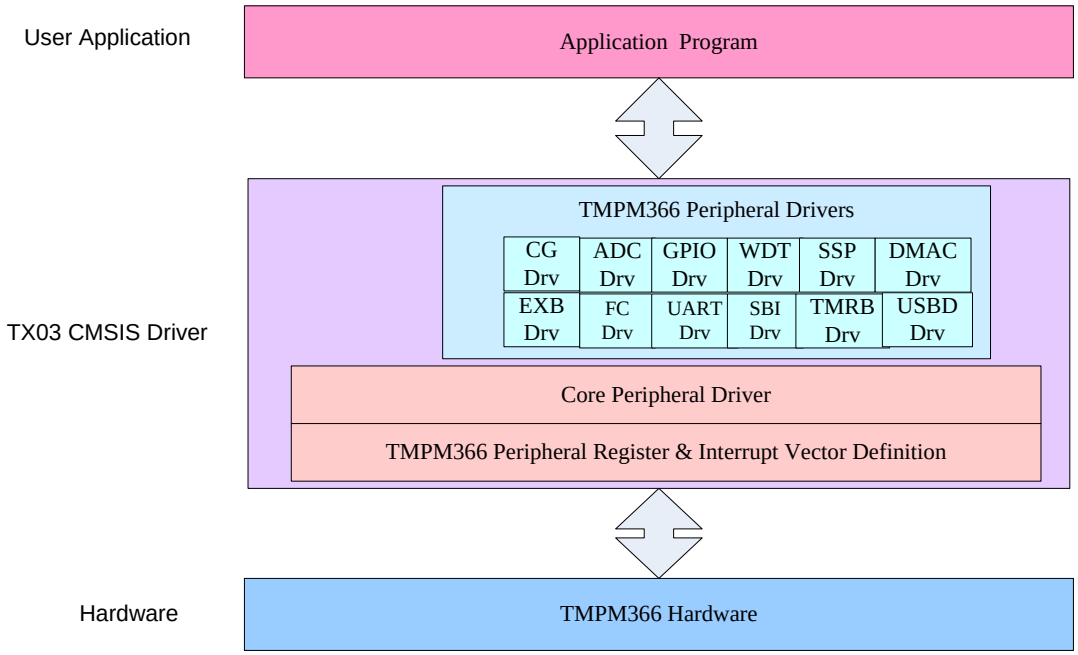
Index

1 General description

The example programs described hereafter are specially designed for TOSHIBA TMPM366 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 peripheral driver as following.



3 Build-in hardware usage

Hardware	Channel	Use presence, use
CG	Clock Gear	Used for CG demo
	PLL	PLL on (8x)
Standby mode	-	Use STOP1 mode
SysTick	-	Unused
Watch dog timer (WDT)	-	Used for WDT demo

Hardware	Channel	Use presence, use
External interrupt (INT)	INT0	Used for releasing STOP1 mode
	INT1	Unused
	INT2	Unused
	INT3	Unused
	INT4	Unused
	INT5	Unused
	INT6	Unused
	INT7	Unused
	INT8	Unused
	INT9	Unused
SIO	SIO0	Used for UART & DMAC demo Tx
	SIO1	Used for UART & DMAC demo Rx
Synchronous serial port (SSP)	SSP0	Used for SSP transfer demo Tx
	SSP1	Used for SSP transfer demo Rx
	SSP2	Unused
Serial bus (SBI)	SBI0	Used for Master Tx
	SBI1	Used for Slave Rx
16-bit timer	TMRB0	Used for TMRB: General Timer
	TMRB1	Unused
	TMRB2	Unused
	TMRB3	Unused
	TMRB4	Used for TMRB: PPG Output
	TMRB5	Unused
	TMRB6	Unused
	TMRB7	Unused
	TMRB8	Unused
	TMRB9	Unused
10-bit A/D converter	AIN0	Unused
	AIN1	Unused
	AIN2	Unused
	AIN3	Unused
	AIN4	Unused
	AIN5	Unused
	AIN6	Unused
	AIN7	Unused
	AIN8	Unused
	AIN9	Unused
	AIN10	Unused
	AIN11	Used for ADC data read
External bus interface (EXB)	CS0	Unused
	CS1	Used for SRAM write/read

4 Pin Usage

The example programs are tested on the IAR TMPM366-SK Board.

Following is pin usage for example programs on IAR TMPM366-SK Board.

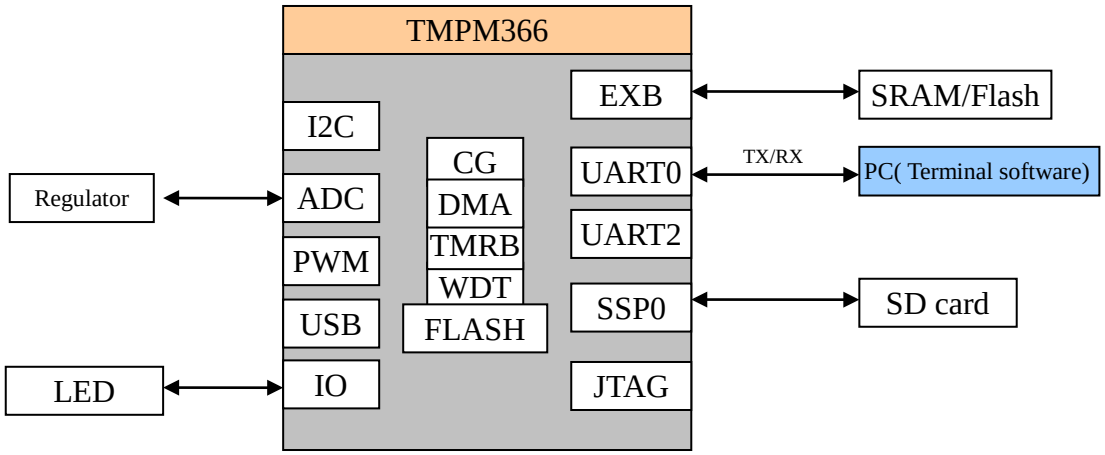
No	Name	Usage
1	AVDD3	+3.3V
2	AVSS	GND
3	AVREFH	+3.3V
4	AVREFL	GND
5	DVSSA	GND
6	DVDD3A	+3.3V
7	TRST, PI7	Unused
8	TDI, PI6	Unused
9	TDO, SWV, PI5	Unused
10	TMS, SWDIO, PI4	Unused
11	TCK, SWCLK, PI3	Unused
12	TRACECLK, PI2	Unused
13	TRACEDATA0, PI1	Unused
14	TRACEDATA1, PI0	Unused
15	TRACEDATA2, PH0	Unused
16	TRACEDATA3, PH1	Unused
17	DCD02, TB4OUT, A10, PH2	PPG Output
18	DSR02, TB5OUT, A9, PH3	Unused
19	DTR02, INT8, A8, PH4	Unused
20	USBPON, INT1, PG5	Unused
21	RTS02, TB4IN1, A7, PG4	Unused
22	RIN02, TB4IN0, A6, INT0, PG3	INT0
23	CTS2, TB3IN1, A5, SCK0, PG2	Unused
24	IRIN, RX02, TB3IN0, A4, SCL0, SI0, PG1	SCL0
25	IROUT, TX02, A3, SDA0, SO0, PG0	SDA0
26	FTEST3	Unused
27	PC0, TXD1, A2, TB2IN0	TXD1
28	PC1, RXD1, A1, TB2IN1	RXD1
29	PC2, SCLK1, A0, TB0OUT, CTS1	Unused
30	PF0, BOOT, TB6OUT	Unused

No	Name	Usage
31	PF1, RD	RD
32	PF2, WR	WR
33	PF3, BELL	BELL
34	PF4, BELH, INT6, TB5IN0	BELH
35	PF5, CS1, INT7, TB5IN1	CS1
36	PF6, CS0	CS0
37	PF7, ALE	ALE
38	DVSSA	GND
39	DVDD3A	+3.3V
40	PA0, AD0, D0	LED1/ AD0
41	PA1, AD1, D1	LED2/ AD1
42	PA2, AD2, D2	LED3/ AD2
43	PA3, AD3, D3	LED4/ AD3
44	PA4, AD4, D4	LED5/ AD4
45	PA5, AD5, D5	LED6/ AD5
46	PA6, AD6, D6	LED7/ AD6
47	PA7, AD7, D7	LED8/ AD7
48	PB0, AD8, D8, SP1DO, A0	SP1DO/ AD8
49	PB1, AD9, D9, SP1DI, A1	SP1DI/ AD9
50	PB2, AD10, D10, SP1CLK, A2	SP1CLK/ AD10
51	PB3, AD11, D11, SP1FSS, A3	SP1FSS/ AD11
52	PB4, AD12, D12, SP2DO, A4	AD12
53	PB5, AD13, D13, SP2DI, A5	AD13
54	PB6, AD14, D14, SP2CLK, A6	AD14
55	PB7, AD15, D15, SP2FSS, A7	AD15
56	DVDD3A	+3.3V
57	DVSSA	GND
58	PD7, SP0FSS, SCOUT	SP0FSS
59	PD6, SP0CLK	SP0CLK
60	PD5, SP0DI	SP0DI
61	PD4, SP0DO	SP0DO
62	DVSS3C	GND
63	D-	Unused
64	D+	Unused
65	DVDD3C	+3.3V
66	PD3, A19, ADTRG	A19
67	PD2, A18, TB9OUT	A18
68	PD1, A17, TB8OUT	PSW1/ A17
69	PD0, A16, TB7OUT	PSW2/ A16

No	Name	Usage
70	PE0, TXD0, A20	TXD0/ A20
71	PE1, RXD0, A21	RXD0/ A21
72	PE2, SCLK0, TB2OUT, CTS0, A22	A22
73	PE3, INT5, A15, TB3OUT, A23	A23
74	MODE	GND
75	RESET	RESET
76	X2	Connect 12MHz oscillator
77	DVSS	GND
78	X1	Connect 12MHz oscillator
79	NMI	Unused
80	A14, SDA1, SO1, PE4	SDA1
81	A13, SCL1, SI1, PE5	SCL1
82	A12, SCK1, PE6	Unused
83	A11, INT4, PE7	Unused
84	DVDD3A	+3.3V
85	DVDD3A	+3.3V
86	DVSSA	GND
87	RVSS	GND
88	RVDD	+3.3V
89	AIN00, PJ0	Unused
90	AIN01, PJ1	Unused
91	AIN02, PJ2	Unused
92	AIN03, PJ3	Unused
93	AIN04, PJ4	Unused
94	AIN05, PJ5	Unused
95	TB0IN0, AIN06, PJ6	Unused
96	TB0IN1, INT9, AIN07, PJ7	Unused
97	TB1IN0, INT2, AIN08, PK0	Unused
98	TB1IN1, INT3, AIN09, PK1	Unused
99	TB6IN0, AIN10, PK2	Unused
100	TB6IN1, AIN11, PK3	AIN11

5 Development Environment

Following is development environment:



1. Hardware board:
   IAR TMPM366-SK Board

2. Development tool:
   - IAR:
     1) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
     2) IDE: IAR Embedded workbench 6.30.1 version
   - KEIL:
     1) U-Link: Realview ULINK2
     2) IDE: KEIL uVision 4.14

6 Functions

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

6-1 Operation mode

There are four operation modes for TMPM366: NORMAL, IDLE, STOP1 and STOP2 mode.

IDLE, STOP1 and STOP2 mode are low power mode.

To shift to lower power mode, in system control register CGSTBYCR<STBY2:0>, select the IDLE, STOP1 or STOP2 mode, and run the WFI (Wait For Interrupt) command.

- **NORMAL mode:**

This mode is to operate the CPU core and the peripheral hardware by using the high-speed clock. It is shifted to the NORMAL mode after reset.

Note: Only STOP1 mode is demonstrated in driver example.

- **STOP1 mode:**

All the internal circuits including the internal oscillator are brought to a stop in STOP1 mode. The STOP1 mode enables to select the pin status by setting the CGSTBYCR<DRVE>.

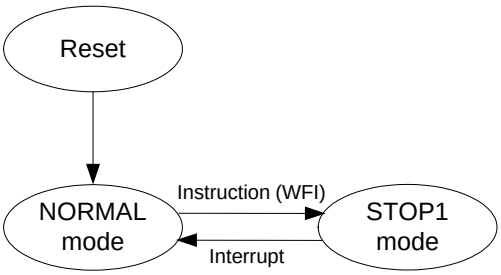
When returning from STOP1 mode, related bits CGPLLSEL<PLLSEL>, CGOSCCR <HWUPSEL>, <OSCSEL>, <XEN2> and <PLLON> are initialized and internal oscillator starts. After elapsed warm-up time, the NORMAL mode is operated.

The warm-up time is needed approximately 3us elapses as stability time for an internal high-speed oscillator and the warm-up time set before entering the STOP1 mode.

When releasing the STOP1 mode by reset, the power-on counter activates a usual reset operation instead of the warm-up counter.

PHCNT interrupt and the extern interrupt can release the STOP1 mode.

Note: The sample program of STOP1 mode is integrated into CG example.



6-2 ADC

6-2-1 ADC Data Read

There is a potentiometer that connected to PK3/AIN11. The voltage on it will be measured and print to stdout. As stdout is retargeted to UART, connect UART0 with a PC and the AD result can be displayed on terminal software.

6-3 CG

6-3-1 Power mode change

Change the CPU operation mode. Only 2 modes are supported, NORMAL and STOP1. Press the key to switch the power mode. LED1 is turned on in the NORMAL mode and turned off in the STOP1 mode.

The following operation is for TMPM366-SK board:

Current Mode	Action (remote key)	Operation	Terminal display
NORMAL	Press [SW1]	NORMAL → STOP1	Now, Going to Stop1
STOP1	Connect PG3(INT0) to VCC(3.3V)	STOP1 → NORMAL	Wakeup from Stop1

6-4 DMAC

6-4-1 Memory to peripheral

This program implements transmission from UART0 to UART1, the string “TOSHIBA” is sent via UART0 to UART1.
The string is transferred from a RAM area to UART0 data register by DMAC, each character of the string is sent via UART0 and received by UART1. After UART1 received the data, it is saved in a character array.

The received value can be checked by RAM variable, the character array in debugger.

Note: In order to run this sample software, the TMPM366-SK Board must be modified:
Connect the TX of UART0 and RX of UART1 via wire

6-5 EXB

6-5-1 SRAM Read/Write

This program implements read/write the external SRAM assembled on TMPM366 evaluation board and EXB is set as 16-bit bus width on multiplex bus. The A.C specifications of SRAM (cycles time) used is for specific SRAM chip. The external SRAM is IS62WV51216BLL with 1Mbyte size.

6-6 FLASH

This program 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 judgment, 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

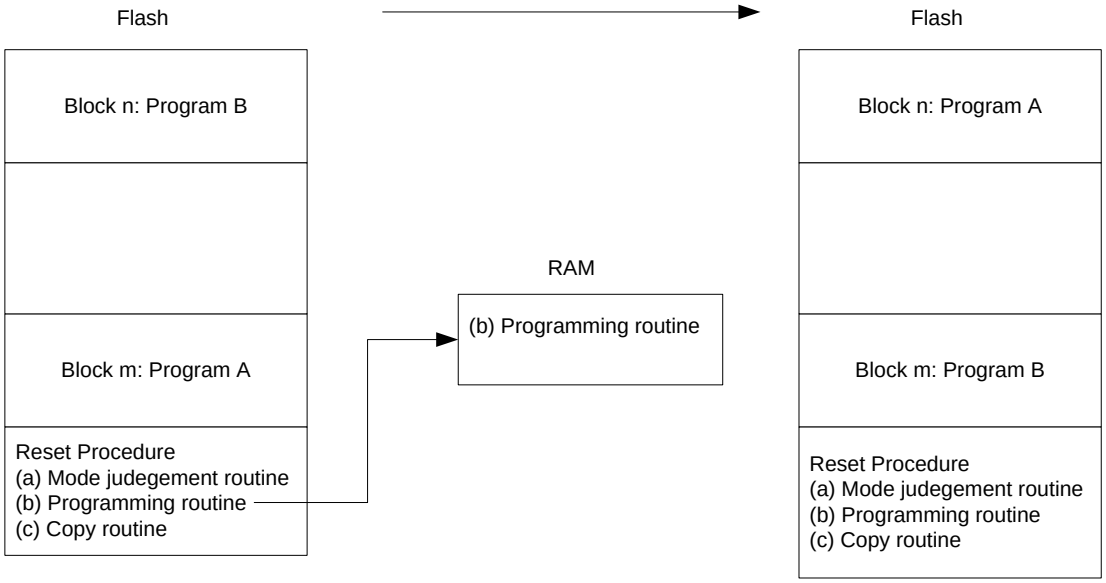
－Program A/B（A: LED 1 is on, B: LED 2 is on)

By default, the program A will run firstly

－Button SW2 is used for mode judgment in Reset procedure

SW2 is released Normal mode

SW2 is pressed User boot mode

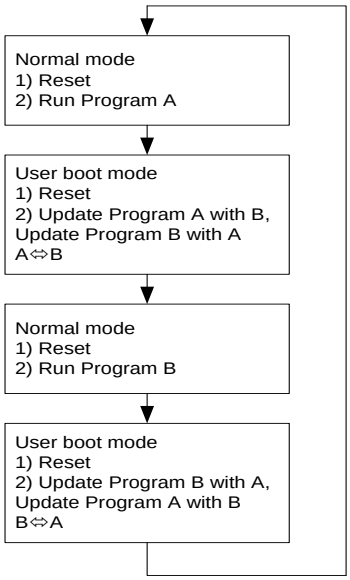


Demo sequence

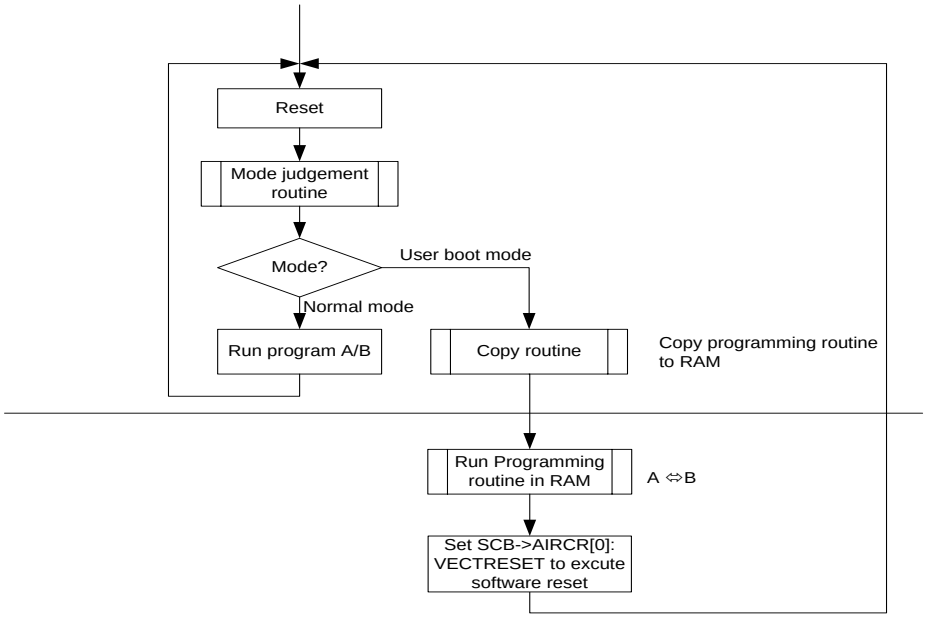
(1) Power on
When SW2 is released, power on or press RESET button. Program A will run at first.
LED 1 is on.

(2) Pressing SW2, then press RESET button, there will be two processes:
Exchange step: led display from right to left “LED4->LED3->LED2”
Exchange of successful step: LED4 and LED3 are on

(3) After exchange of successful, release SW2 and system will reset automatically to run Demo B.
LED2 is on



・**Demo process flow**



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

LED4 is on
(User boot mode)

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

LED display sample:

LED3 is on LED2 is on LED3 and LED4 are on

(RAM transferring) (Swap A and B) (Swap finished, wait for reset)

6-7 FUART

In this program, both Full UART Transmit and Receive FIFO are enabled.
This program sends 64 different data value from Full UART channel 0 TX02 pin and receives data from RX02 pin. When Key SW1 is pressed, this program starts to read data from Receive FIFO. The program doesn't stop reading data until Receive FIFO is empty. After Key SW1 is pressed several times, the program finished reading all the data that RX02 receives. Then the program will compare all the received data with transmitted data.
If received data are same with transmitted data, LED 1,2,3,4,5,6,7,8 turn on for a short time, then turn off, then repeat this process.
If received data are different with transmitted data, LED 1,3,5,7 turn on for a short time, then turn off, then repeat this process.

This program can be run for two times to see the hardware flow control function.
The first time:
Enable RTS and CTS hardware flow control, the received data are same with transmitted data.
The second time:
Disable RTS and CTS hardware flow control, the received data are different with transmitted data.

NOTE: Connect TX02 pin with RX02 pin on TMPM366-SK evaluation board.
Connect RTS pin with CTS pin on TMPM366-SK evaluation board.

6-8 GPIO

This is a simple application based on the Peripheral Driver (GPIO), use GPIO functions to configure GPIO to LED, then turn on the LED, or turn off the LED.

6-9 SBI

This program intends to support the I2C bus slave mode.
Connect two I2C buses (SBI0 & SBI1) on TOSHIBA TMPM366-SK board.
One is working as I2C master, and the other is working as I2C slave.
Use SBI interrupt to handle I2C bus read/write.
Address of I2C slave: 0xB0.
I2C Slave (SBI1) receives "TOSHIBA" from I2C Master (SBI0).

Received results can be checked by RAM variable gI2CRxData[] in debugger.

Note: In order to run this sample software, TOSHIBA TMPM366-SK board must be modified.
Connect the pin PE5(SCL1) and pin PG1(SCL0).
Connect the pin PE4(SDA1) and pin PG0(SDA0).

6-10 SIO/UART

6-10-1 Retarget

This program intends to retarget the stdin and stdout of the C lib.
Stdin and stdout are retargeted to UART0. Then application code can use printf() and getchar() to output to or input from serial port.

6-10-2 UART FIFO

In this sample program, UART0 sent the data "TMPM3661" to UART1 use FIFO, and at same time UART0 receive the data "TMPM3662" 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 = "TMPM3662",and RxBuffer1 = "TMPM3661".

6-10-3 SIO

This demo will show synchronously transfer and receive usage of SIO module in TMPM366
It use the channel SIO0, SIO1 and transfer data synchronously between them.(connect TXD0 with RXD1, connect TXD1 with RXD0, connect sclk0 with sclk1)
Note: For M366-SK board, the E-JP5 and E-JP7 should be no CAP (disconnected) due to they are related with the SCLK.

	SIO pins	Pin Number	Connected PINS
1	SIO0:RXD0	JP1:No.7	JP2:No.38
2	SIO0:TXD0	JP1:No.8	JP2:No.37
3	SIO0:SCLKx/PE2	JP1:No.6	JP2:No.36
4	SIO1:RXD1	JP2:No.37	JP1:No.8
5	SIO1:TXD1	JP2:No.38	JP1:No.7
6	SIO1:SCLKx/PC2	JP2:No.36	JP1:No.6

6-11 SSP

6-11-1 SSP0 to SSP1 via DMAC

For M366, there are 3 SSP channels, SSP0 is set as SPI Master while SSP1 is set as SPI Slave, then connect correspond pins, use DMA to transmit/receive infinite self increased data.

Note: TOSHIBA TMPM366 –SK board must be modified as below:

	SSP0 pins	Note	SSP1 pins
1	PD7/SP0FSS	Connect PD7 to PB3	PB3/SP1FSS
2	PD6/SP0CLK	Connect PD6 to PB2	PB2/SP1CLK
3	PD5/SP0DI	Connect PD5 to PB0	PB1/SP1DI
4	PD4/SP0DO	Connect PD4 to PB1	PB0/SP1DO

* SSP0 to SSP1 via DMAC demo should run on TOSHIBA TMPM366 -SK Board.

6-11-2 SSP0 Self Loop Back

Infinite data is sent and checked if the received data is OK or not. And if transmission is set to lowest speed mode (lowest bit rate), what happens will be checked.

Note: The SD card socket must be left nothing inside.

6-12 TMRB

6-12-1 General Timer

This function implements a general 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).

6-12-2 PPG Output

Use Toggle Switch 1 to change PPG waveform. LeadingTiming can be changed as following: 10%, 25%, 50%, 75%, 90%
Power on to enter PPG mode and starts the PPG output.
Change the leadingtiming every switch changing:
10% → 25% → 50% → 75% → 90% → 10%

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 TMPM366-SK Board which will demonstrate the main feature of TMPM366 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
        │   tmpm366_wdt_int.c
        ├──TX03_CMSIS
        │   │   core_cm3.c
        │   │   core_cm3.h
        │   │   system_TMPM366.c
        │   │   system_TMPM366.h
        │   │   TMPM366.h
        │   │
        │   └──startup
        │       startup_TMPM366.s
```

```
        |
        ├──TX03_Periph_Driver
        |    ├──inc
        |    |    tmpm366_wdt.h
        |    |    tmpm366_gpio.h
        |    |    tx03_common.h
        |    |
        |    └──src
        |         tmpm366_wdt.c
        |         tmpm366_gpio.c
        └──TMPM366-SK
             led.c
             led.h
```

KEIL MDK:

```
    ├──WDT_NMI
        ├──APP
        |   main.c
        |   tmpm366_wdt_int.c
        |
        ├──TX03_CMSIS
        |    core_cm3.c
        |    system_TMPM366.c
        |    startup_TMPM366.s
        |
        ├──TX03_Periph_Driver
        |    tmpm366_wdt.c
        |    tmpm366_gpio.c
        |
        └──TMPM366-SK
             led.c
```

7-1 ADC

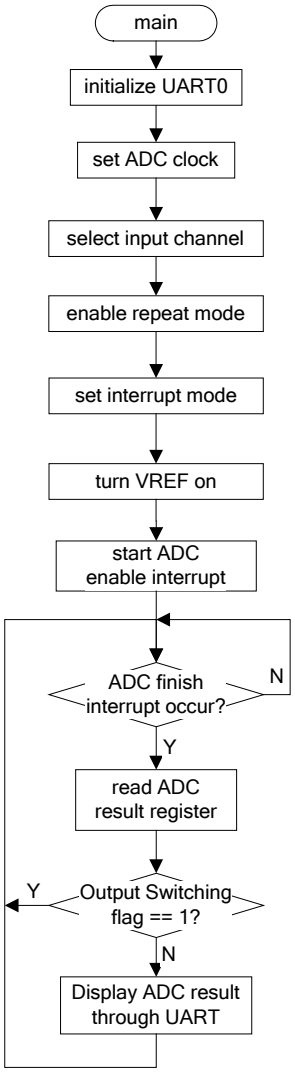
7-1-1 Example: ADC Data Read

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

The example includes:

1. ADC configuration and initialization
2. Start ADC in fixed channel repeat mode and read AD result

- **Flowchart:**



- **Code and Explanation for the Example**

At first, set ADC clock, select input channel, enable repeat mode, set interrupt mode and turn VREF on.

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

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

/* select ADC input channel */
ADC_SetInputChannel(ADC_AN_11);

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

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

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

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

Then start ADC and enable INTAD:

```
ADC_Start();

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

After started ADC, wait for INTAD. When INTAD occurs, call ADC_Display() function.

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

In ADC_Display() function, read corresponding ADREG to get ADC result, and check OutputSwitching flag.

```
ADC_Result result = ADC_GetConvertResult(ADC_REG_00);

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

7-2 CG

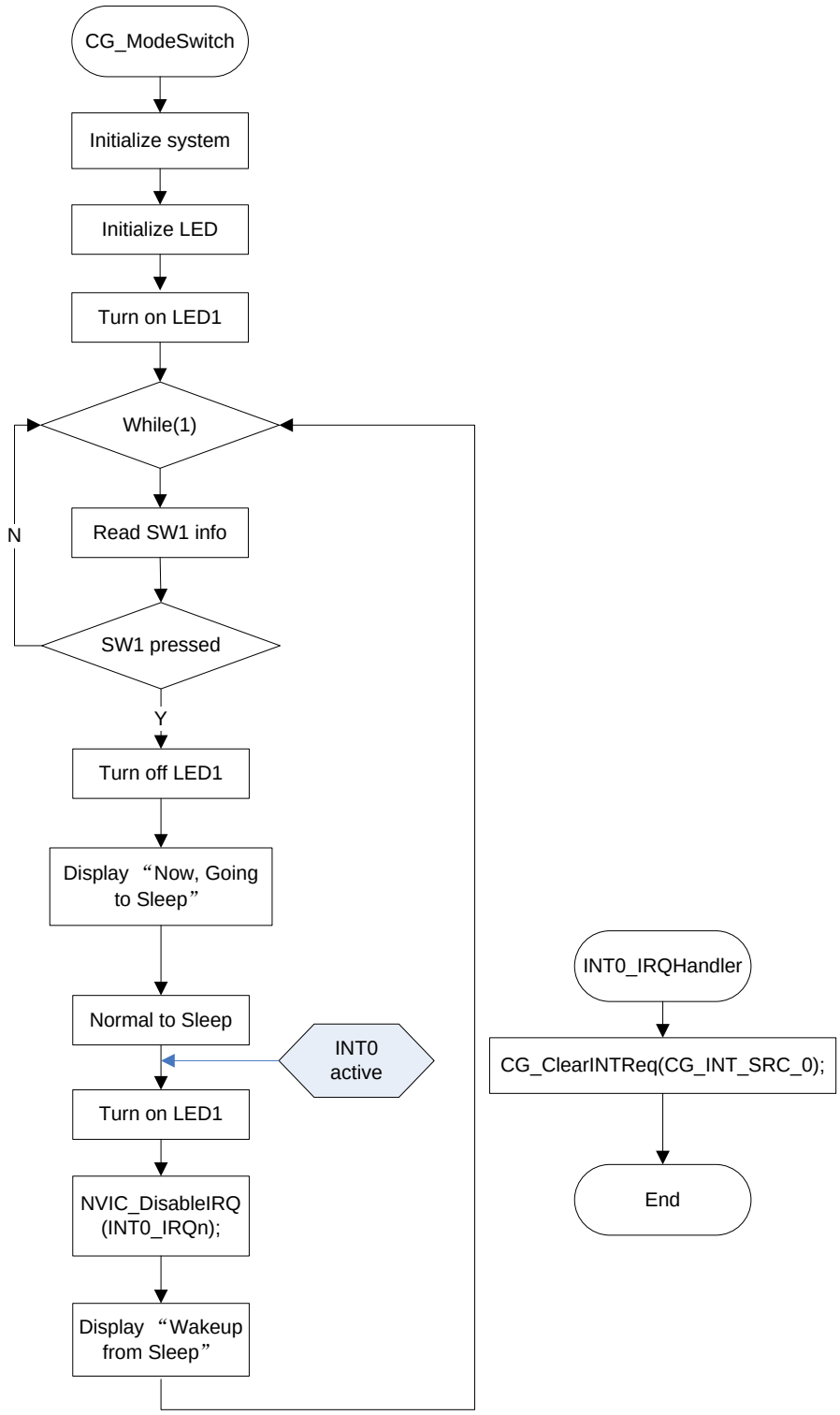
7-2-1 Example: Power mode change

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

The example includes:

1. Basic setup operation of CG
2. How to switch between normal mode and stop1 mode
3. How to enable multiple clock circuit

- **Flowchart**



- **Code and Explanation for the Example**

The following simple example is based on TX03 Peripheral Driver (CG), which will switch between normal mode and stop1 mode.

Normal setup for CG (after reset)

The following code is just an example for setting CG in normal mode. It is supposed that the external high-speed oscillator is 12MHz.

Example code is as the following:

```
    /* Set SCOUT source */
    CG_SetSCOUTSrc(CG_SCOUT_SRC_PHIT0);
    if (CG_GetFoscSrc()==CG_FOSC_OSC_INT){
        /* Switch over from IHOSC to EHOSC*/
        switchFromIHOSCtoEHOSC();
    }
    /* Set up pll and wait 200us for pll to warm up , set fc source to fpll */
    CG_EnableClkMulCircuit();
    /* Set fgear = fc/2 */
    CG_SetFgearLevel(CG_DIVIDE_2);
    /* Set fperiph to fgear */
    CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
    CG_SetPhiT0Level(CG_DIVIDE_64);
    /* Set low power consumption mode stop1 */
    CG_SetSTBYMode(CG_STBY_MODE_STOP1);
    /* Set pin status in stop mode to "active" */
    CG_SetPinStateInStop1Mode(ENABLE);
```

Setup to enter STOP1 mode

Prepare to enter stop1 mode. Set warm up time, wait for warm up is complete. Set INT0 interrupt to wake up system. Enable INT0 interrupt, and clear interrupt request. Finally use __WFI() instruction enter STOP1 mode.

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

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

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

Enable multiple clock circuit

Set PLL and set the source of fc. First set PLL value and enable PLL, then set warm up time and wait warm up time is complete. Finally set fPLL for fc source.

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

        do {
```

```
            st = CG_GetWarmUpState();
        } while (st != DONE);

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

    return retval;
```

7-3 DMAC

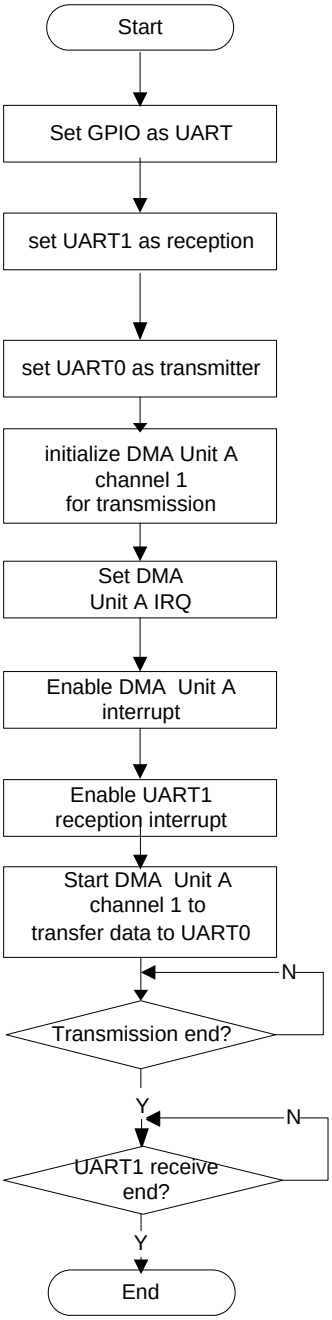
7-3-1 Example: Memory to peripheral

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

The example includes:

1. UART0/1 configuration and DMAC initialization
2. Transfer data from memory to UART0 via DMAC

- **Flowchart:**



- **Code and Explanation for the Example**

The following simple example is based on TX03 Peripheral Driver (DMAC), which will transfer data from memory to UART0.

Firstly set UART0 as transmitter and enable it

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

Configure and initialize DMA Unit A channel1 for transmission

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

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

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

Enable DMA Unit A IRQ

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

Enable DMA Unit A circuit, transfer end interrupt, burst transfer request for UART0 and start channel 1 of DMAC to transfer

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

Wait the end of DMAC transmission

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

Set flag for DMAC transmission end in DMAC ISR

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

7-4 EXB

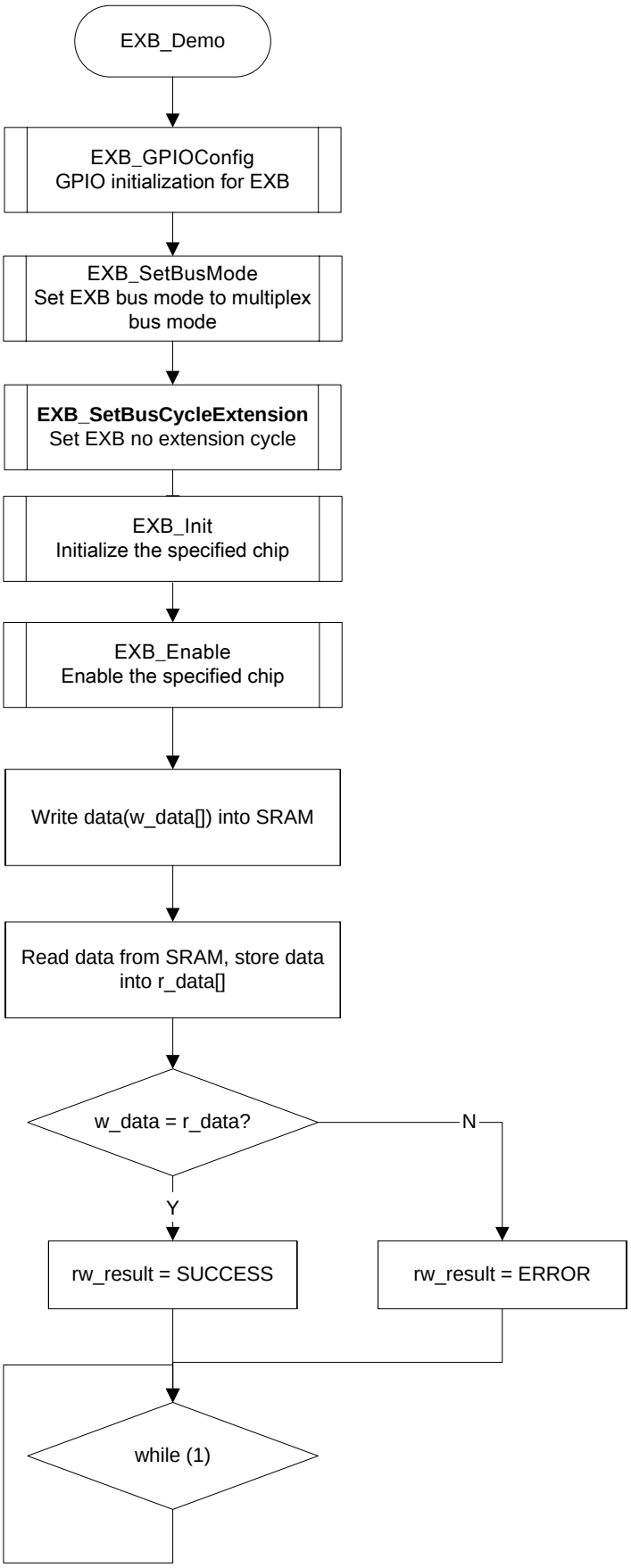
7-4-1 Example: Read/Write SRAM

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

The example includes:

1. Initialization of EXB
2. Read/Write the external SRAM

- **Flowchart:**


EXB_Demo
EXB_GPIOConfig
GPIO initialization for EXB
EXB_SetBusMode
Set EXB bus mode to multiplex bus mode
EXB_SetBusCycleExtension
Set EXB no extension cycle
EXB_Init
Initialize the specified chip
EXB_Enable
Enable the specified chip
Write data(w_data[]) into SRAM
Read data from SRAM, store data into r_data[]
w_data = r_data?
N
Y
rw_result = SUCCESS
rw_result = ERROR
while (1)

- **Code and Explanation for the Example**

Firstly set initial value for EXB.

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

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

	EXB_InitTypeDef InitStruct = { 0U };

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

Configure GPIO for EXB and EXB, then enable the specified chip. Write w_data[] into SRAM, after reading data from SRAM and store the data into r_data[]. Check rw_result to see whether SRAM write/read is sucessful.

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

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

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

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

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

```
    memcpy(r_data, addr, (uint32_t) rw_cnt);

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

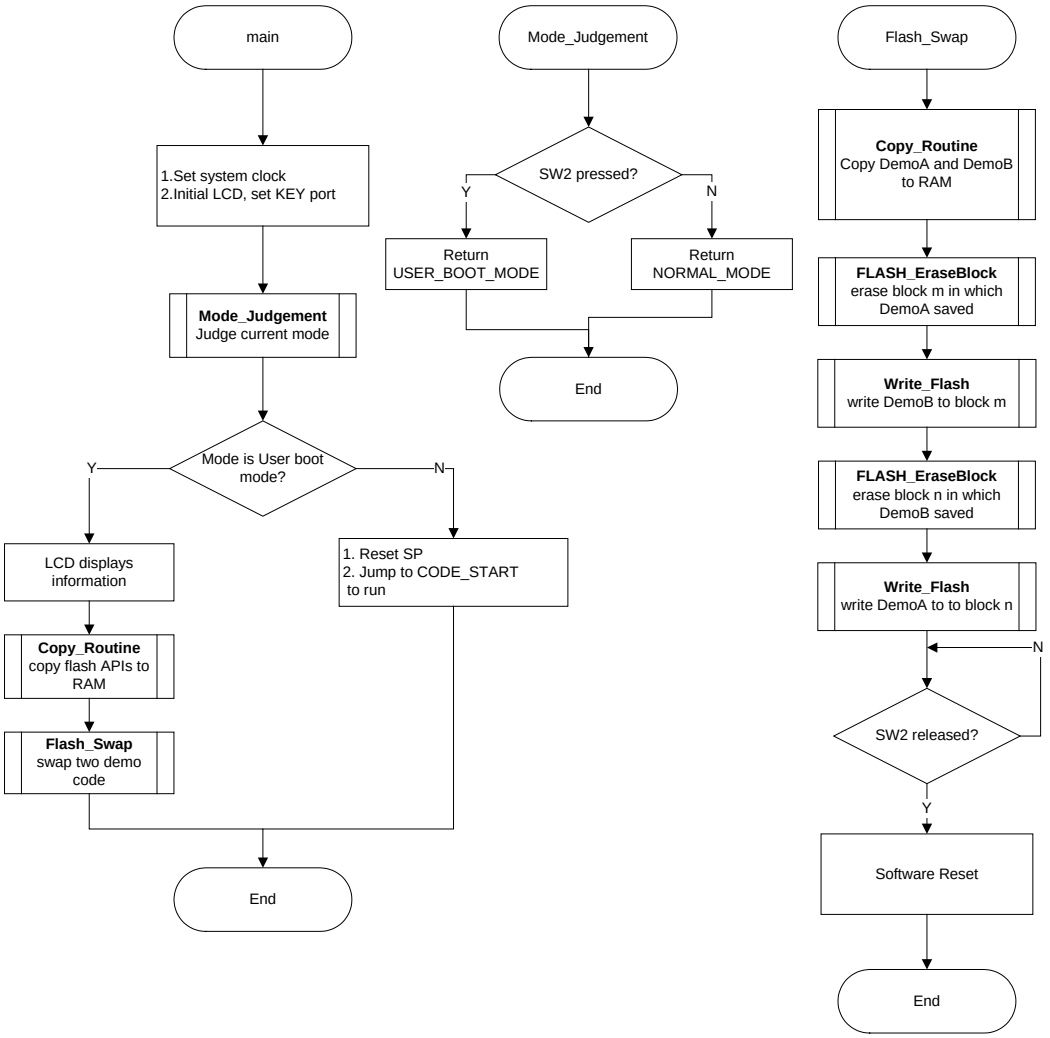
7-5 FLASH

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 SWITCH port and LED. SWITCH port (GPIO) is used to judge current mode when reset and LED will display current operation information.

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

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

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

If SW2 is released 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 is on).

```
#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 SW2 is pressed during reset, the routine will enter user boot mode. LED4 will be on to indicate this mode. 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. LED3 is on to indicate RAM transferring.

```
    Status_Display (0x08U);       /* status 1: enter user boot mode */
    Copy_Routine (FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
                                                 /* copy flash API to RAM */
     Status_Display (0x04U);      /* status 2: RAM transferring */
```

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 FC_EraseBlock () and Write_Flash() to erase and program Flash memory. The code of Demo A and B will be exchanged in Flash memory. LED2 is on to indicate swapping two demos.

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

    Status_Display(0x02U);              /* status 2: swap the demo */

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

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

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

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

LED4 and LED3 will be on to indicate the swap operation has completed. After SWITCH changed to high, 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 (LED2 is on).

```
Status_Display(0x0CU);                    /* status 3: complete and restart */

while (GPIO_ReadDataBit(GPIO_PD, GPIO_BIT_0) == GPIO_BIT_VALUE_0) {
        /* wait for SW2 release */
}
reg_value = SCB->AIRCR;        /* software reset */
reg_value = (uint32_t) 0x05FA0001;
SCB->AIRCR = reg_value;
```

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

```
if (ENABLE == FC_GetBlockProtectState(BlockNum)) {
    retval = FC_ERROR_PROTECTED;
}
```

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

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

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

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

```
*addr1 = (uint32_t) 0x000000AA;          /* bus cycle 1 */
*addr2 = (uint32_t) 0x00000055;          /* bus cycle 2 */
*addr1 = (uint32_t) 0x000000A0;          /* bus cycle 3 */
for (i = 0U; i < FC_PAGE_SIZE; i++) {                  /* bus cycle 4~7 */
    *addr3 = *source;
    source++;
}
```

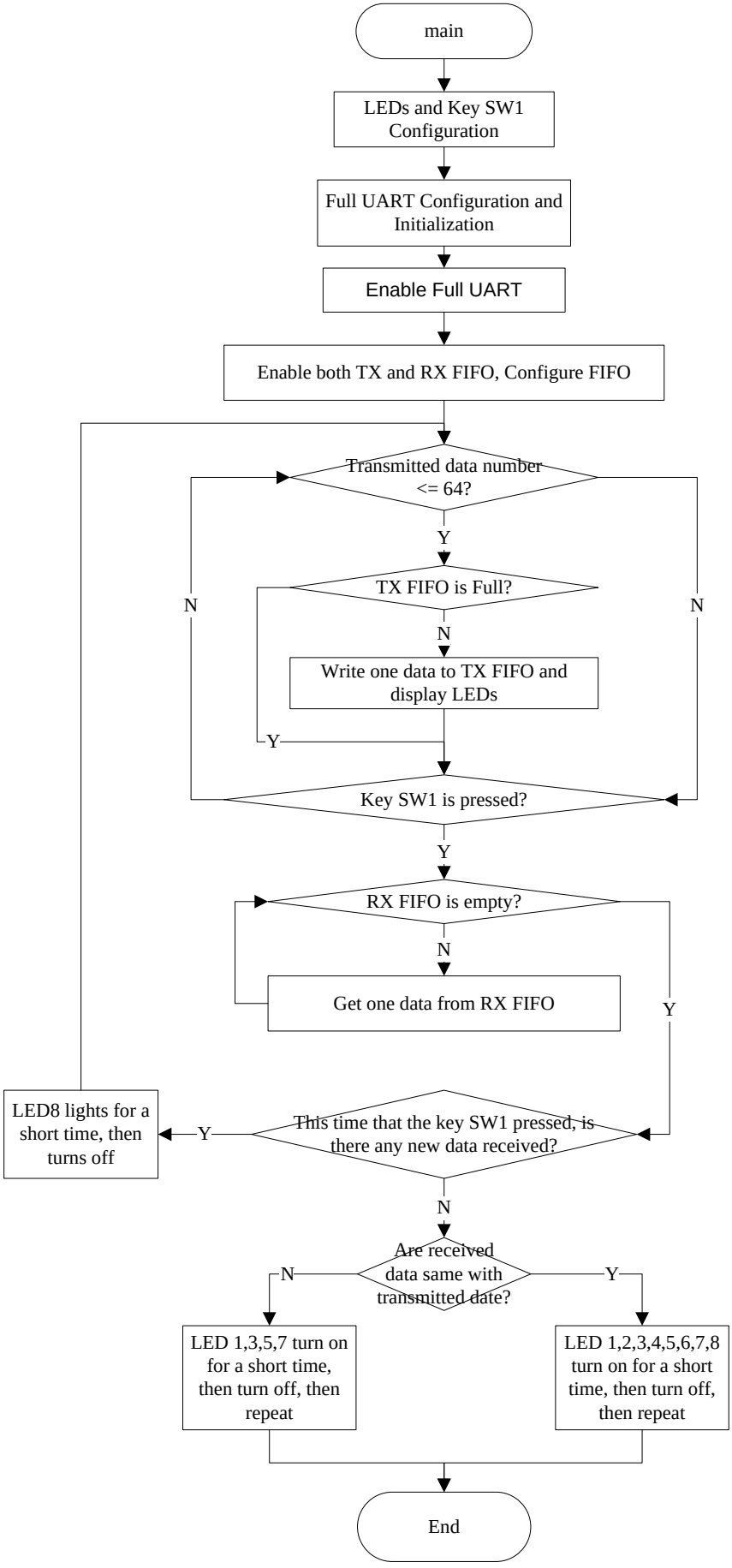
7-6 FUART

7-6-1 Example: LoopBack

This is a simple example based on the TX03 Peripheral Driver (FUART, GPIO).
This program can be run for two times to see the hardware flow control function.
The first time: Enable RTS and CTS hardware flow control
The second time: Disable RTS and CTS hardware flow control
The example includes:

1. LEDs and Key SW1 Initialization, Full UART configuration and initialization.
2. Full UART Transmit data process.
3. Press SW1 to receive data.
4. When receving data is finished, compare the received data with transmitted data.

- **Flowchart**


main
LEDs and Key SW1 Configuration
Full UART Configuration and Initialization
Enable Full UART
Enable both TX and RX FIFO, Configure FIFO
Transmitted data number <= 64?
Y
TX FIFO is Full?
N
N
N
Write one data to TX FIFO and display LEDs
Y
Key SW1 is pressed?
Y
RX FIFO is empty?
N
Get one data from RX FIFO
Y
LED8 lights for a short time, then turns off
Y
This time that the key SW1 pressed, is there any new data received?
N
Are received data same with transmitted date?
N
Y
LED 1,3,5,7 turn on for a short time, then turn off, then repeat
LED 1,2,3,4,5,6,7,8 turn on for a short time, then turn off, then repeat
End

- **Code and Explanation for the Example**

Before You run the program You need to decide whether to use the RTS and CTS flow control. If RUN_NONE_FLOW_CONTROL is undefined, RTS and CTS flow control will be enabled in the program. If RUN_NONE_FLOW_CONTROL is defined, there will be no flow control in the program.

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

At first, the program Initializes LEDs and Key SW1
Use GPIO peripheral drivers configure GPIO for LEDs and Key SW1.

```
    LEDInit();
    LedOff(LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8);
    SW1_Init();
```

Use GPIO peripheral drivers configure GPIO for FUART0.

```
    /* Configure port PG0 to be TX02 */
    GPIO_SetOutput(GPIO_PG, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_0);

    /* Configure port PG1 to be RX02 */
    GPIO_SetInput(GPIO_PG, GPIO_BIT_1);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_1);

    /* Configure port PG2 to be CTS2 */
    GPIO_SetInput(GPIO_PG, GPIO_BIT_2);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_2);
    GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_2, DISABLE);

    /* Configure port PG4 to be RTS02 */
    GPIO_SetOutput(GPIO_PG, GPIO_BIT_4);
    GPIO_EnableFuncReg(GPIO_PG, GPIO_FUNC_REG_4, GPIO_BIT_4);
    GPIO_SetOpenDrain(GPIO_PG, GPIO_BIT_4, DISABLE);
```

Create a FUART_InitTypeDef structure and fill all the data fields, then Initialize FUART0.

```
    FUART_InitTypeDef myFUART;

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

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

    FUART_Init(FUART0, &myFUART);
```

Use FUART peripheral drivers enable FUART0 and configure FIFO.

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

Then FUART0 starts to send data, the Full UART will only send 64 differfent data value, and it will send the data when the transmit FIFO is normal or empty. When each data is sent, the LEDs display the data.

```
    if (cntTx < MAX_BUFSIZE) {
        FIFOStatus = FUART_GetStorageStatus(FUART0, FUART_TX);
        if ((FIFOStatus == FUART_STORAGE_EMPTY)
            || (FIFOStatus == FUART_STORAGE_NORMAL)) {
            FUART_SetTxData(FUART0, Tx_Buf[cntTx]);
            LED_TXDataDisplay(Tx_Buf[cntTx]);
            cntTx++;
        }
    }
```

Each time when Key SW1 is pressed, the program starts to read data from Receive FIFO. If some data are got, the program doesn't stop reading data until the FIFO is empty, and LED8 will light on for a short time to show there's data that has been read. If no any data can be got when SW1 is pressed, the program starts to compare the received data with transmitted data. If received data are same with transmitted data, LED 1,2,3,4,5,6,7,8 will repeat turning on and turning off. If received data are different with transmitted data, LED 1,3,5,7 will repeat turning on and turning off.

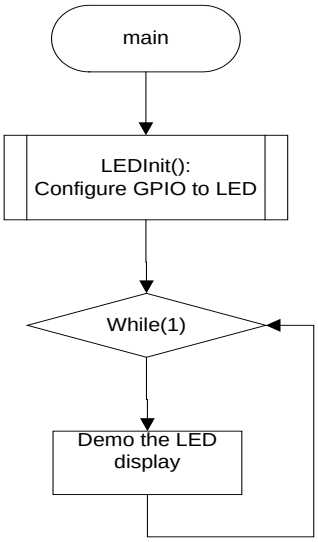
```
if (SW1_DOWN == SW1_Get()) { /* press the key SW1 */
    while (FUART_STORAGE_EMPTY != FUART_GetStorageStatus(FUART0,
FUART_RX)) {
        receive = FUART_GetRxData(FUART0);
        Rx_Buf[cntRx] = receive;
        cntRx++;
    }
    rxlast = rxthis;
    rxthis = cntRx;
    if (rxlast != rxthis) { /* there are some data that has been received    */
        LED_RXDataDisplay(LED8_ON);
        delay(0x180000U);
        LED_RXDataDisplay(LED8_OFF);
    } else {                                        /* receiving data is finished */
        result = Buffercompare(Tx_Buf, Rx_Buf, MAX_BUFSIZE);
        if (result == SAME) {
    /* received data are same with trnsmitted data */
    /* RTS and CTS flow control has worked normally */
            while (1) {
                LedOn(LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8);
                delay(0xFFFFFU);
                LedOff(LED1 | LED2 | LED3 | LED4 | LED5 | LED6 | LED7 | LED8);
                delay(0xFFFFFU);
            }
        } else {
    /* received data are different with trnsmitted data */
    /* RTS and CTS flow control doesn't work */
            while (1) {
                LedOn(LED1 | LED3 | LED5 | LED7);
                delay(0xFFFFFU);
                LedOff(LED1 | LED3 | LED5 | LED7);
                delay(0xFFFFFU);
            }
        }
    }
}
```

7-7 GPIO

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

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

- **Flow chart:**



- **Code and Explanation for the Example:**

At first, Configure GPIO to LED. For example, set the port as output pin, set port data.

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

In the While process, do the LED demo: LED on and LED off, and display Second%10U.

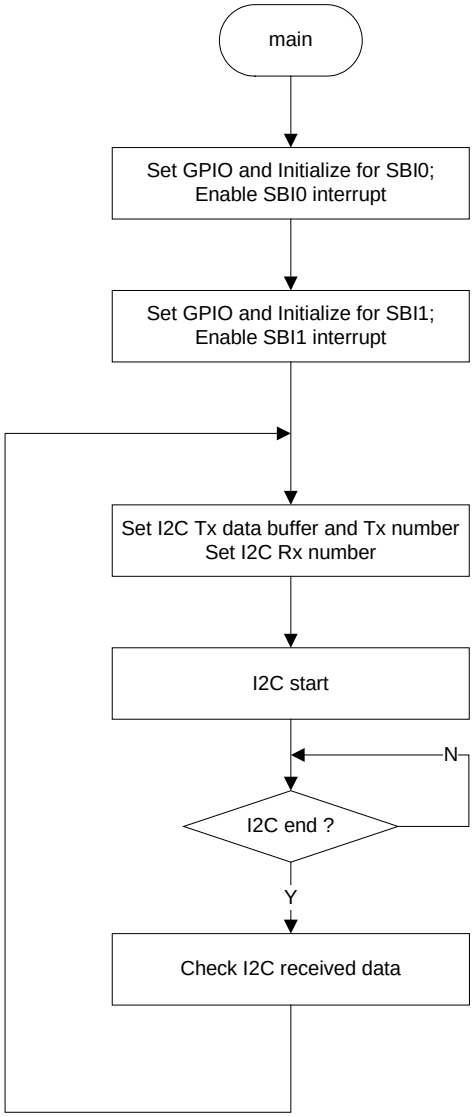
7-8 SBI

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

- **Flow Chart**



- **Code and Explanation for the Example**

At first, configure GPIO for SBI0 and SBI1 I2C mode.

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

Initialize and configure SBI0 channel and enable INTSBI0.

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

Then enable, initialize and configure SBI1 channel and enable INTSBI1.

```
myI2C.I2CSelfAddr = SLAVE_ADDR;
```

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

After the above setting, start I2C read.
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_SBI0) is used to to start I2C process.

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

The data transfer is handled in INTSBI0.
The data receive is handled in INTSBI1.

In INTSBI0 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 INTSBI0_IRQHandler(void)
{
    TSB_SBI_TypeDef *SBIx;
    SBI_I2CState sbi_sr;

    SBIx = TSB_SBI0;
    sbi_sr = SBI_GetI2CState(SBIx);

    if (sbi_sr.Bit.MasterSlave) {           /* Master mode */
        if (sbi_sr.Bit.TRx) {     /* Tx mode */
            if (sbi_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further data. */
```

```
                SBI_GenerateI2CStop(SBIx);
            } else {                /* LRB=0: the receiver requires further data. */
                if (gI2CWCnt <= gI2CTxDataLen) {
                    SBI_SetSendData(SBIx, gI2CTxData[gI2CWCnt]);
                                                                    /* Send next data */
                    gI2CWCnt++;
                } else {            /* I2C data send finished. */
                    SBI_GenerateI2CStop(SBIx);    /* Stop I2C */
                }
            }
        } else {                    /* Rx Mode */
            /* Do nothing */
        }
    } else {                        /* Slave mode */
        /* Do nothing */
    }
}
```

In INTSBI1 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 INTSBI1_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI_TypeDef *SBIy;
    SBI_I2CState sbi1_sr;

    SBIy = TSB_SBI1;
    sbi1_sr = SBI_GetI2CState(SBIy);

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

7-9 SIO

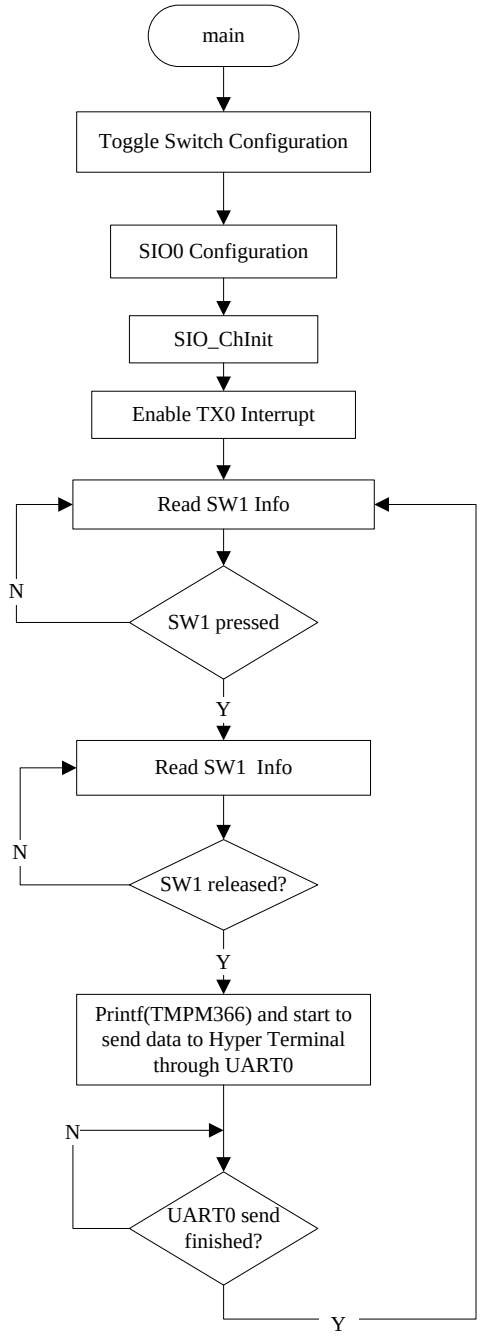
7-9-1 Example: Retarget

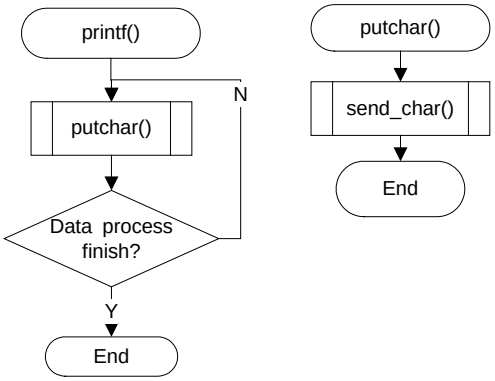
This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:

5. UART configuration and initialization.
6. UART TX process.
7. Use UART0 TX interrupt to send data.
8. Retarget printf() to UART0.

- **Flowchart**


main
Toggle Switch Configuration
SIO0 Configuration
SIO_ChInit
Enable TX0 Interrupt
Read SW1 Info
N
SW1 pressed
Y
Read SW1 Info
N
SW1 released?
Y
Printf(TMPM366) and start to send data to Hyper Terminal through UART0
N
UART0 send finished?
Y


printf()
N
putchar()
Data process finish?
Y
End
putchar()
send_char()
End

- **Code and Explanation for the Example**

At first, Configure Toggle Switch and SIO1, then Initialize UART0.
Use GPIO peripheral drivers configure GPIO for Toggle Switch.

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

Use GPIO peripheral drivers configure GPIO for UART0.

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

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

```
UART_InitTypeDef myUART;

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

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);
```

Then Read Toggle Switch info:

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

Then Judge whether SW1 has been pressed, if SW1 is pressed, then wait for SW1 released:

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

After SW1 has been released, start to send data by printf() through UART0.

```
printf("%s\r\n", TxBuffer);        /* SIO2 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 (fSIO_INT == CLEAR) {       /* if SIO INT disable, enable it */
            fSIO_INT = SET;               /* set SIO INT flag */
            UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);
            NVIC_EnableIRQ(INTTX0_IRQn);
      }

      return ch;
}
```

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

UART0 TX interrupt routine:

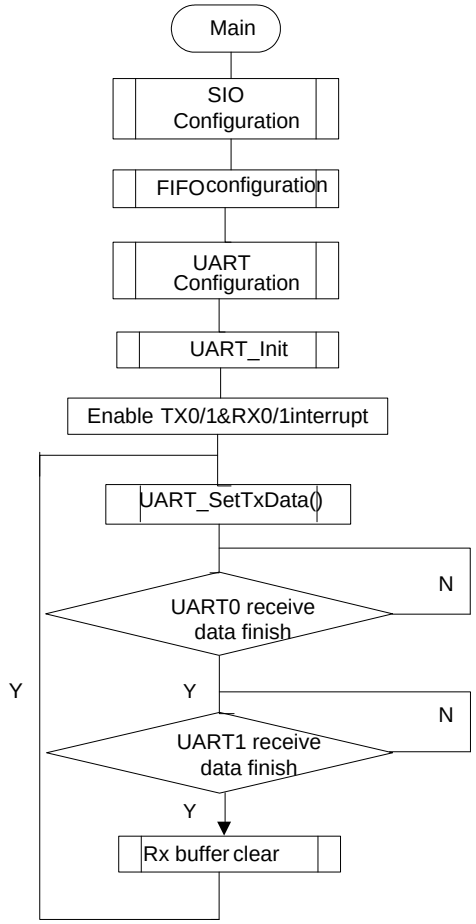
```
void INTTX0_IRQHandler(void)
{
      if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
            UART_SetTxData(UART0, gSIOTxBuffer[gSIORdIndex++]);/* send data */
            fSIO_INT = SET;               /* SIO0 INT is enabled */
      } else {
            /* disable SIO0 INT */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

7-9-2 Example: UART FIFO

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

The example includes:
UART and FIFO configuration and initialization.
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_0;
            TSB_PE->FR1 |= GPIO_BIT_0;
            TSB_PE->FR1 |= GPIO_BIT_1;
            TSB_PE->IE |= GPIO_BIT_1;
      } else if (SCx == TSB_SC1) {
            TSB_PC->CR |= GPIO_BIT_0;
            TSB_PC->FR1 |= GPIO_BIT_0;
            TSB_PC->FR1 |= GPIO_BIT_1;
            TSB_PC->IE |= GPIO_BIT_1;
      }
}
```

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);
	}
}
```

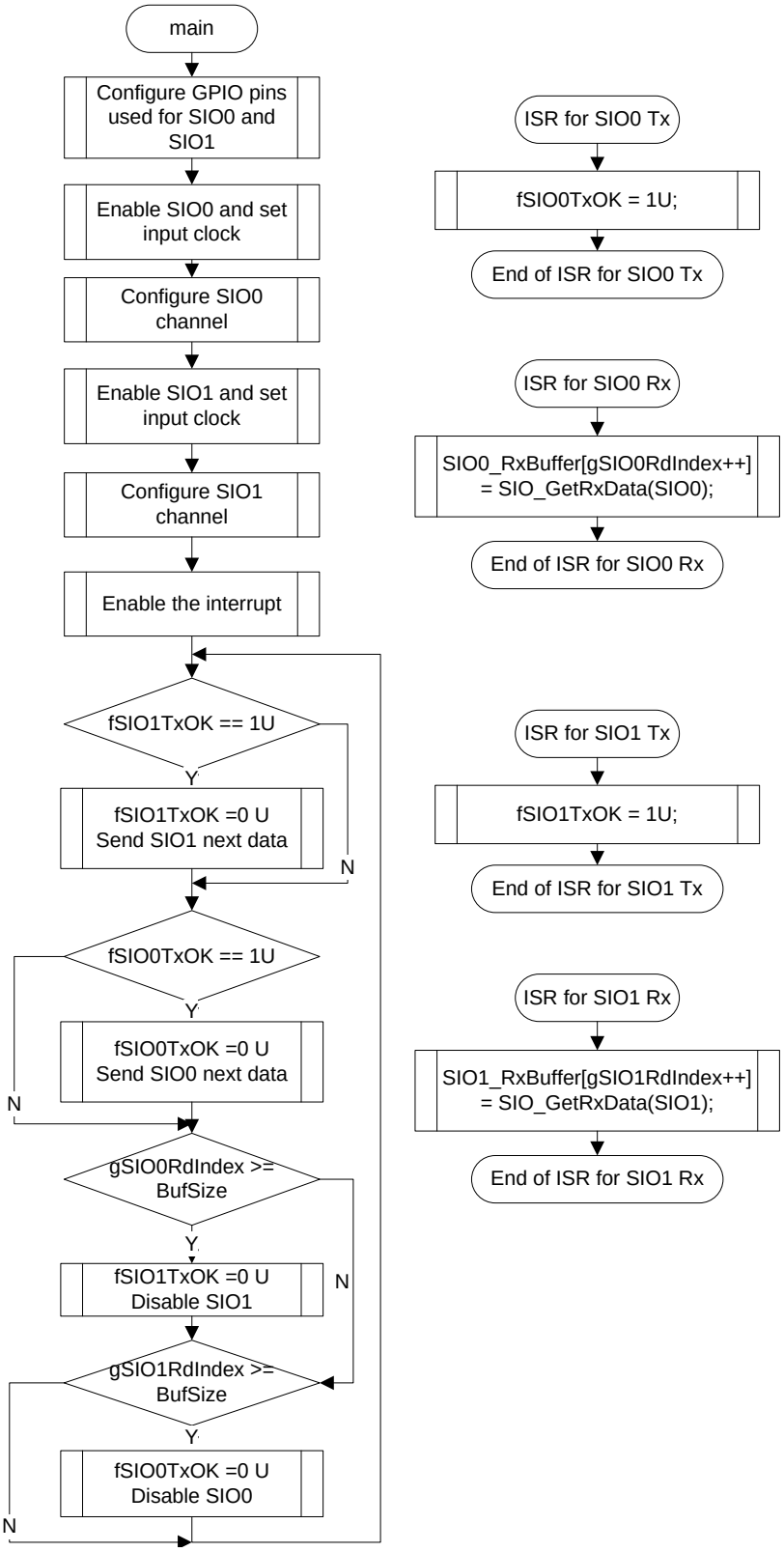
7-9-3 Example: SIO

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_8;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_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.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);
}
```

7-10 SSP

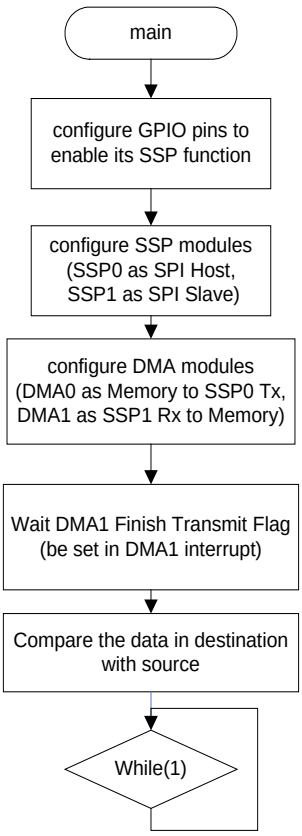
7-10-1 Example: SSP0 to SSP1 via DMAC

This is a simple example based on the TX03 Peripheral Driver (SSP, DMAC, GPIO).
It will send some data from SSP0 to SSP1 via DMA.

The example includes:

1. Configuration of SSP0 as SPI Host, and SSP1 as SPI Slave
2. Configuration of DMA0 as Memory to SSP0 Tx, and DMA1 as SSP1 Rx to Memory
3. Check the data transmit flow: Memory → SSP0 Tx → SSP1 Rx → Memory

- **Flowchart**



- **Code and Explanation for the Example**

Below is the main.c for the flowchart above.

```
int main(void)
{
	GPIO_SetSSP();
	initSSP();
	InitDMA();

	/* Wait the end of transmission */
	while (TxEndFlag != DONE) {
		/* Do nothing */
	}

	/* now DMA is finished, Set a Break Point here, */
	/* after function Buffercompare() is called, result == SAME */
	result = Buffercompare( SRC_Buffer, DST_Buffer, BUFFER_SIZE);

	while(1) {
		/* do nothing */
	}
}
```

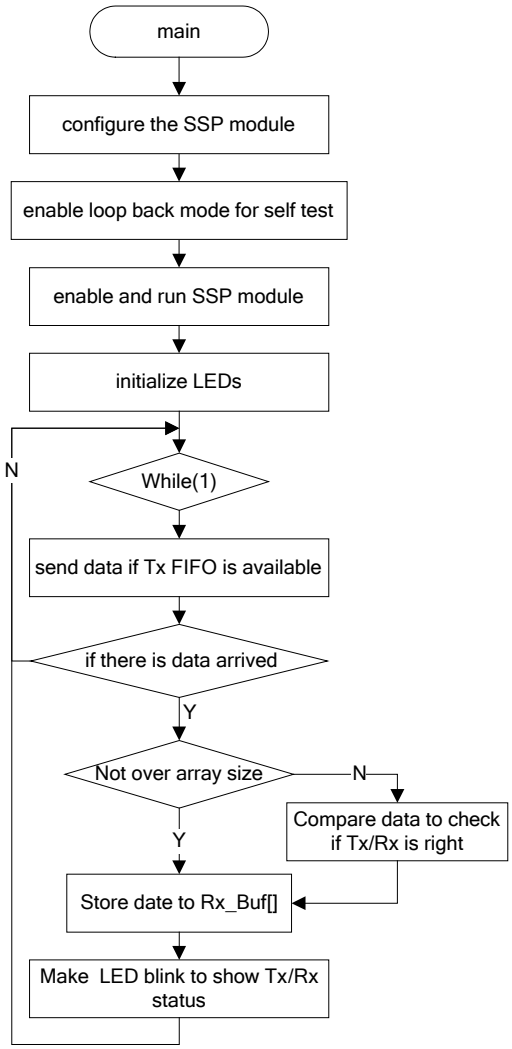
For detail of functions: GPIO_SetSSP(), initSSP(), InitDMA() above, please refer to the comments in the code file.

7-10-2 Example: SSP0 Self Loop Back

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

The example includes:
1. Configuration and initialization SSP module channel 0
2. Enable its loop back mode to do self Tx/Rx

- **Flowchart**



- **Code and Explanation for the Example**

```
int main(void)
{
    SSP_InitTypeDef initSSP;
    SSP_FIFOState fifoState;

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

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

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

    /* default is to run at maximum bit rate */
    initSSP.PreScale = 2U;
    initSSP.ClkRate = 1U;
    /* define BITRATE_MIN to run at minimum bit rate */
    /*     BitRate = fSYS / (PreScale x (1 + ClkRate)) */
#ifdef BITRATE_MIN
    initSSP.PreScale = 254U;
    initSSP.ClkRate = 255U;
#endif
    initSSP.ClkPolarity = SSP_POLARITY_LOW;
    initSSP.ClkPhase = SSP_PHASE_FIRST_EDGE;
    initSSP.DataSize = 16U;
    initSSP.Mode = SSP_MASTER;
    SSP_Init(TSB_SSP0, &initSSP);

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

    /* enable and run SSP module */
    SSP_Enable(TSB_SSP0);

    /* initialize LEDs on M366-SK board before display something */
    LEDInit();

    while (1) {

        datTx++;
        /* send data if Tx FIFO is available */
        fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_TX);
        if ((fifoState == SSP_FIFO_EMPTY) || (fifoState == SSP_FIFO_NORMAL)) {
            SSP_SetTxData(TSB_SSP0, datTx);
            if (cntTx < MAX_BUFSIZE) {
                Tx_Buf[cntTx] = datTx;
                cntTx++;
            } else {
                /* do nothing */
            }
        } else {
            /* do nothing */
        }

        /* check if there is data arrived */
        fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_RX);
        if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
            receive = SSP_GetRxData(TSB_SSP0);
            if (cntRx < MAX_BUFSIZE) {
                Rx_Buf[cntRx] = receive;
                cntRx++;
            } else {
                /* Place a break point here to check if receive data is right.    */
                /* Success Criteria:                                              */
                /* Every data transmited from Tx_Buf is received in Rx_Buf. */
```

```
                    /* When the line "#define BITRATE_MIN" is commented, the SSP is
run in maxium */
                    /* bit rate, so we can find there is enough time to transmit date from
1 to */
                    /* MAX_BUFSIZE one by one. but if we uncomment that line, SSP
is run in */
                    /* minimum bit rate, we will find that receive data can't catch
"datTx++", */
                    /* in this so slow bit rate, when the Tx FIFO is available, the cntTx */
                    /* has been increased so much.                                    */
                    __NOP();
                    result = Buffercompare( Tx_Buf, Rx_Buf, MAX_BUFSIZE);
                }

            } else {
                /* do nothing */
            }

            DisplayLED(receive);
        }
}
```

7-11 TMRB

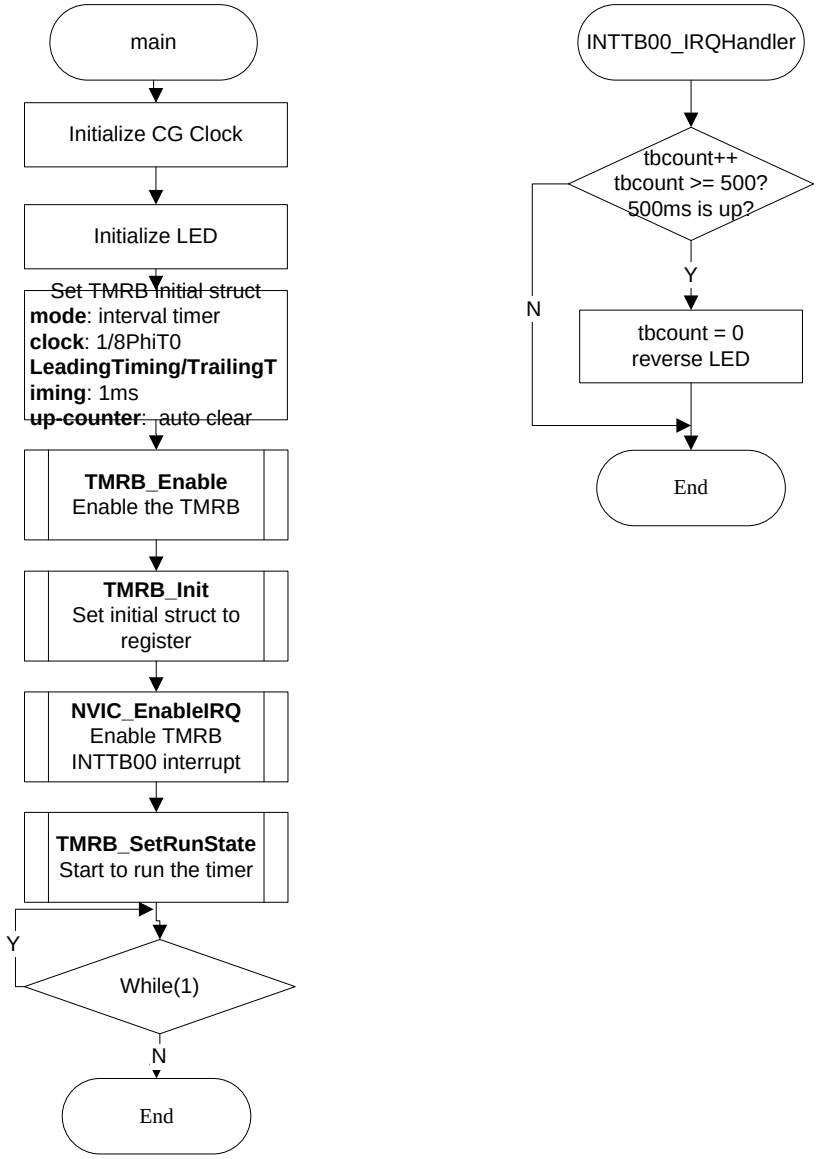
7-11-1 Example: General Timer

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

The example includes:

1. TMRB0 initialization
2. General Timer for 1ms

- **Flow Chart**



- **Code and Explanation for the Example**

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

```
CG_InitSystem();                    /* CG_SetSystem */
LEDInit();                          /* LED initialize */
LedDisable(LED1 | LED2 | LED3);
LedOn(LED4);                                    /* Turn on LED1 */
```

Prepare TMRB initialization structure, fill TMRB mode, clock, up-counter clear method, trailingtiming and leadingtiming. This demo will set trailingtiming and leadingtiming to 1ms, macro TMRB_1MS equals 0x1770, because fphiT0 = fsys = fc = 12MHz * PLL* 1/2= 48MHz, ftmrb = 1/8 fphiT0 = 6MHz, Ttmrb = 0.167us, 1ms/0.167us = 6000 = 0x1770 (Please see CG part for more detail information about the clock setting)

```
TMRB_InitTypeDef m_tmrb;

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

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

Enable the TMRB module and set the initialization structure to specified registers. Enable INTTB00 interrupt, this interrupt will be triggered every 1ms, in the end, start the TMRB to run.

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

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.

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

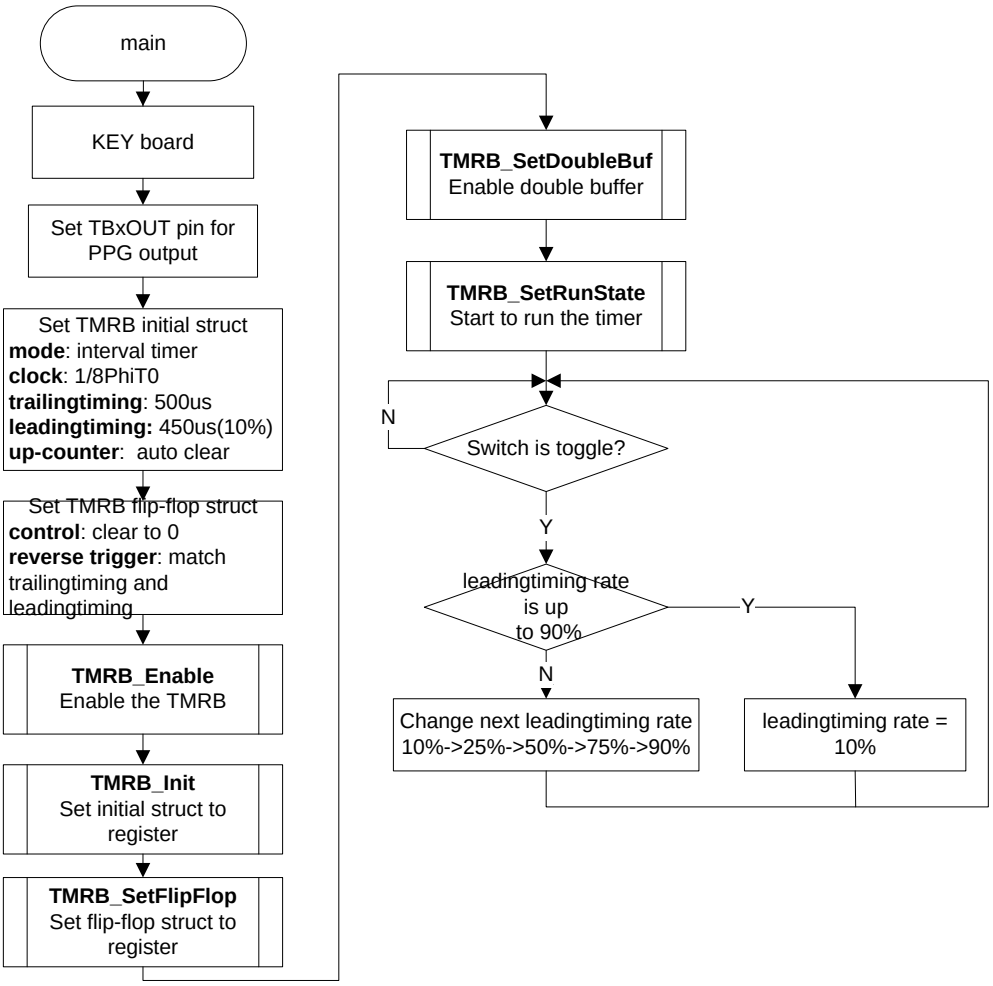
7-11-2 Example: PPG Output

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

The example includes:

1. TMRB4 initialization
2. PPG process setting and start
3. PPG leadingtiming adjustment

- **Flow Chart**



- **Code and Explanation for the Example**

At first, initialize KEY board, set PH2 as TB4OUT for PPG output.

```
    GPIO_SetInput(KEYPORT, GPIO_BIT_1);                    /* set KEY port to input */

    /* Set PH2 as TB4OUT for PPG output */
    GPIO_SetOutput(GPIO_PH, GPIO_BIT_2);
    GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_3, GPIO_BIT_2);
```

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 TMRB4TIME equals 0x0BB8, because fphiT0 = fsys = fc = 12MHz * PLL * 1/2 = 48MHz, ftmrb = 1/8 fphiT0 = 6MHz, Ttmrb = 0.167us, 500us/0.167us = 3000 = 0x0BB8 (Please see CG part for more detail information about the clock setting)

```
    TMRB_InitTypeDef m_tmrb;

    m_tmrb.Mode = TMRB_INTERVAL_TIMER;      /* internal timer */
    m_tmrb.ClkDiv = TMRB_CLK_DIV_8;         /* 1/8PhiT0 */
    m_tmrb.TrailingTiming = TMRB4TIME;              /* trailingtiming is 500us
*/
```

```
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;        /* up-counter auto clear */
    m_tmrb.LeadingTiming  =  LeadingTiming[Rate];                              /*
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_TB4);
    TMRB_Init(TSB_TB4, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB4, &PPGFFInital);
    TMRB_SetDoubleBuf(TSB_TB4, ENABLE);     /* enable double buffer */
    TMRB_SetRunState(TSB_TB4, TMRB_RUN);
```

Wait switch from low to high, and at the same time, display current leadingtiming on oscillometer.

```
    Do {                            /* wait if switch is Low */
          keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_1);
        } while (GPIO_BIT_VALUE_0 == keyvalue);
    delay(0xFFFU);                  /* noise cancel */
```

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

```
    Do {
          keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_1);
        } while (GPIO_BIT_VALUE_1 == keyvalue);
    delay(0xFFFU);                  /* noise cancel */

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

    TMRB_ChangeLeadingTiming(TSB_TB4,  LeadingTiming[Rate]);      /* switch is
High again */
```

The calculation method of leadingtiming value:

TrailingTiming = 500us, ftmrb = 1/8 fphiT0 = 6MHz, Ttmrb = 0.167us (these time parameters will be different if use different CG setting)

LeadingTiming = 10%: high-level time is 500*10% = 50us, low-level time is 500-50 = 450us, counter value = 450us/Ttmrb = 0XA8C

LeadingTiming = 25%: high-level time is 500*25% = 125us, low-level time is 500-125 =

375us, counter value = 375us/Ttmrb = 0x8CA

LeadingTiming = 50%: high-level time is 500*50% = 250us, low-level time is 500-250 = 250us, counter value = 250us/Ttmrb = 0x5DC

LeadingTiming = 75%: high-level time is 500*75% = 375us, low-level time is 500-375 = 125us, counter value = 125us/Ttmrb = 0x2EE

LeadingTiming = 90%: high-level time is 500*90% = 450us, low-level time is 500-450 = 50us, counter value = 50us/Ttmrb = 0x12C

That is the calculation method of leadingtiming array

```
uint32_t LeadingTiming[5] = { 0XA8CU, 0x8CAU, 0x5DCU, 0x2EEU, 0x12CU };
/* leadingtiming: 10%, 25%, 50%, 75%, 90% */
```

7-12 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 flag.

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

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();
```