TOSHIBA

TX03 Peripheral Driver Usage Example (TMPM365)

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M365UE-01E

RESTRICTIONS ON PRODUCT USE

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

© 2017 Toshiba Electronic Devices & Storage Corporation

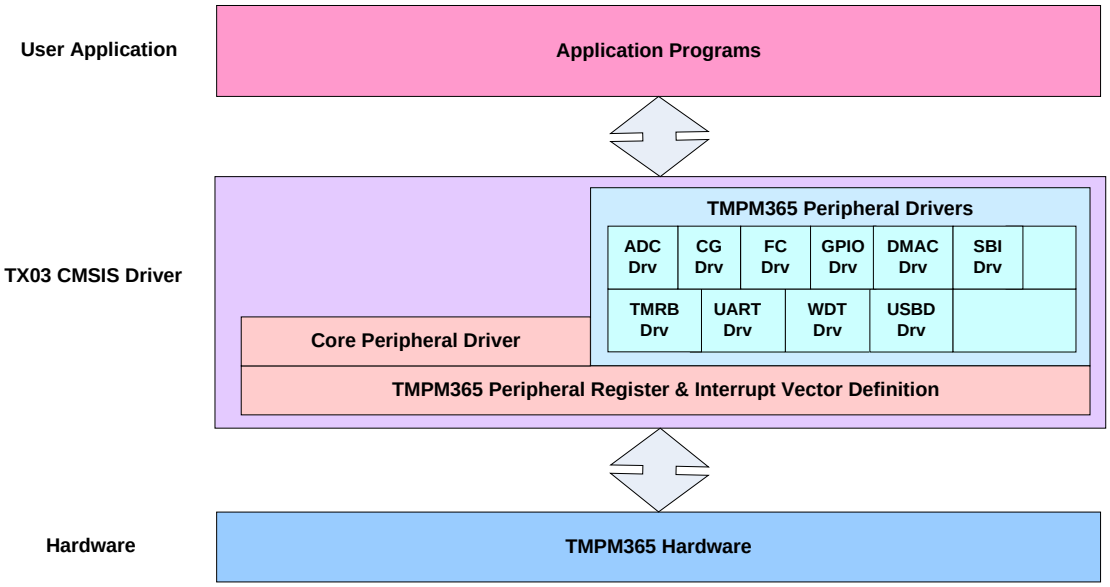
Index

1 General description

The example programs described hereafter are specially designed for TOSHIBA TMPM365 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
ADC	AIN0	Used for ADC demo
	AIN1 to AIN11	Not used
CG	-	Used for CG demo
Standby mode	-	Use SLEEP mode
External interrupt	INT0	Used for call CG wake up.
	INT1 to INT9	Not used
DMAC	-	Used for DMAC demo
SIO	SIO0	Used for UART retarget demo
	SIO1	Used for UART FIFO and SIO demo
16-bit timerB	TMRB0	Used for TMRB: General Timer, PPG Output
	TMRB1 to TMRB9	Not used
SBI	SBI0	Used for I2C Slave Rx demo
	SBI1	Used for I2C Slave Tx demo
WDT	WDT0	Used for WDT demo
USBD	-	Used for an HID mouse demo

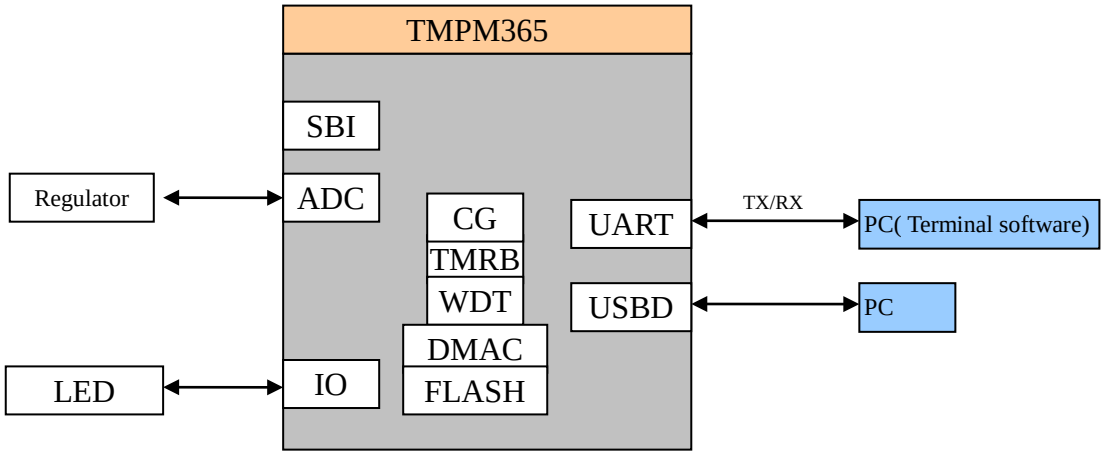
4 Pin Usage

The example programs are tested on TOSHIBA TMPM365 Evaluation Board.
Following is pin usage for the example programs.

Pin No.	Name	Usage
R10	PA0	LED0, or Pull_up control in USBD demo
R11	PA1	LED1
P11	PA2	LED2
R12	PA3	LED3
N15	PB4	SW0
M14	PB5	SW1
R2	PC0/Txd1	UART1_Tx
R3	PC1/Rxd1	UART1_Rx
D15	PD1	For Key input in TMRB demo
A15	PE0/Txd0	UART0_Tx
C14	PE1/Rxd0	UART0_Rx
B10	PE4/SDA1	SBI demo, SDA1
B9	PE5/SCL1	SBI demo, SCL1
P5	PG0/SDA0	SBI demo, SDA0
P4	PG1/SCL0	SBI demo, SCL0
R1	PG3/INT0	For CG demo, INT0
L2	PI3	SWCLK for SWD debug
K2	PI4	SWDIO for SWD debug
G15	D+	D+ pin for USBD demo
H15	D-	D- pin for USBD demo
F1 R8 J14 B13 H14 A7 E1 E2	Gnd	Gnd for demo

5 Development Environment

Following is development environment:



1. Hardware board:
   TOSHIBA M365 Evaluation Board.

2. Development tool:
   - IAR:
     1) J-Link: IAR J-Link-7.0/8.0
     2) IDE: IAR Embed Workbench for ARM version 6.40
   - KEIL:
     1) u-Link: Realview ULINK2
     2) IDE: KEIL uVision MDK version 4.21

6 Functions

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

6-1 Operation mode

There are three operation modes for TMPM365: NORMAL, IDLE and STOP1 mode.

IDLE and STOP1 mode are low power mode.

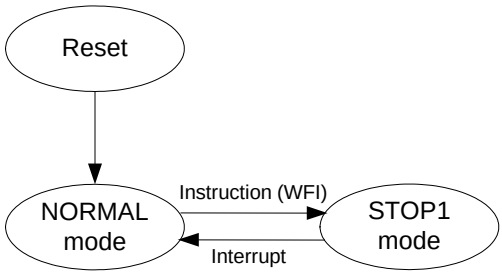
To shift to lower power mode, in system control register CGSTBYCR<STBY2:0>, select the IDLE or STOP1 mode, and run the WFI (Wait For Interrupt) command.

- **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 demostrated into CG example.



6-2 ADC

Change the value of VR3, which is connected to AIN0, the ADC result for the voltage on it will be measured and printed to stdout. As stdout is retargeted to UART, connect UART with a PC and the AD result can be displayed on terminal software

The UART configuration is: baud rate = 115200, no handshake, 8-bit data, 1-bit stop.

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. LED0 is turned on in the NORMAL mode and turned off in the STOP1 mode.

The following operation is for TMPM365-EVAL 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

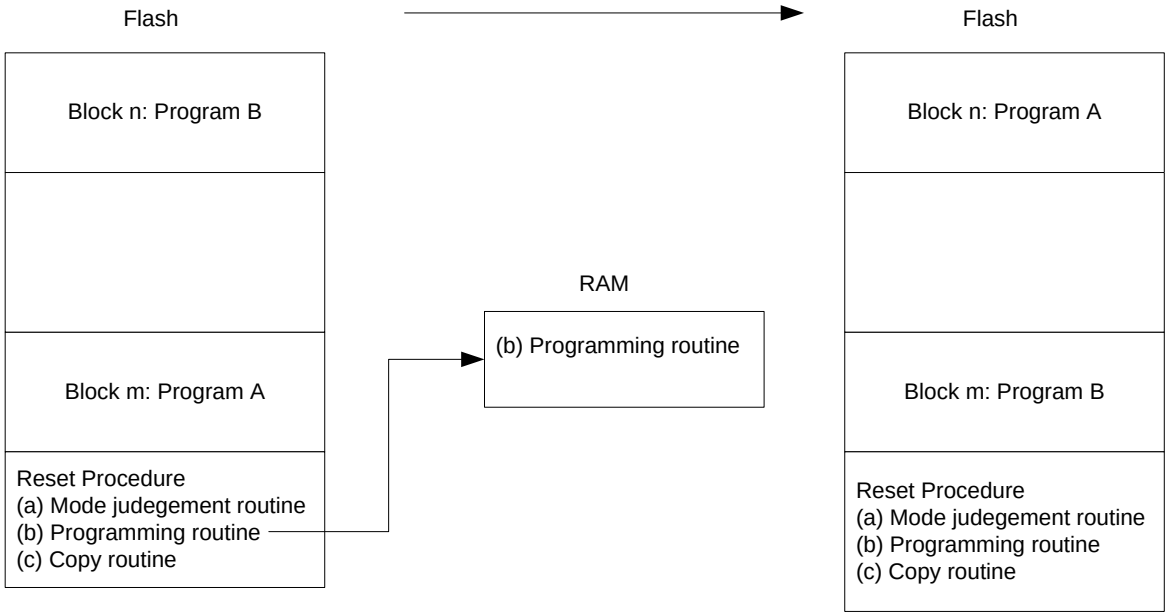
6-4 Flash

This example demonstrates the feature of flash APIs such as erase and write 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: LED0 blinks
  B: LED1 blinks)

By default, the program A will run firstly

－Toggle switch SW0 is used for mode judgment in Reset procedure
  SW0 turned on → User boot mode
  SW0 turned off → Normal mode



Demo sequence

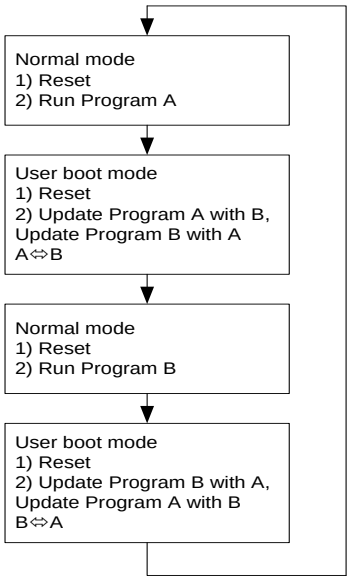
(1) Power on
The demo board runs Reset Procedure.
Then initial program A stored in flash runs.
LED0 blinks。

(2) Press RESET button while SW0 is turned on, and release SW0 until LED3 is ON.
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.

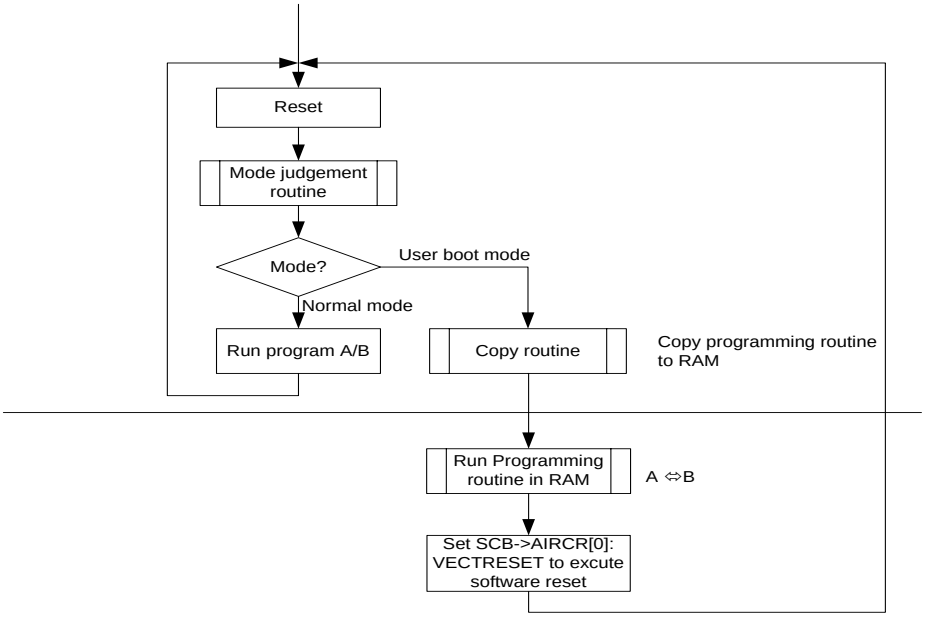
(3) After finished restart, the demo will reset by software automatically.
Then program B stored in flash runs.
LED1 blinks.

(4) Press RESET button again while SW0 is turned on, and release SW0 until LED3 is ON.
Same as step (2).

(5) After finished restart, the demo will reset by software automatically.
Then program A stored in flash runs.
LED0 blinks.



・**Demo process flow**



6-5 DMAC

Memory to peripheral

This function 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 TMPM365 evaluation board must be modified:
Connect the TX of UART0 and RX of UART1 via wire

6-6 GPIO

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

6-7 SBI

This function intends to support the I2C bus slave mode.
Connect two I2C buses (SBI0 & SBI1) on TOSHIBA TMPM365 evaluation 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, use TOSHIBA TMPM365 evaluation board which must be modified.
Connect the pin PE5(SCL1) and pin PG1(SCL0).
Connect the pin PE4(SDA1) and pin PG0(SDA0).

6-8 TMRB

6-8-1 General Timer

This function implements a general timer utilizing MCU timer.
Timer trailing timing is set to 1ms.
Use this timer for LED blinking and the period is 1s (500ms ON and 500ms OFF).

6-8-2 PPG Output

Use Toggle Switch 1 to change PPG waveform. Leading timing can be changed as following:

10%, 25%, 50%, 75%, 90%

Power on to enter PPG mode and starts the PPG output.

Change the leading timing every switch changing:

10% ->25%--> 50%-->75% --> 90%-->10%

6-9 SIO/UART

6-9-1 Retarget

This function intends to retarget the stdin and stdout of the C lib.
Stdin and stdout are retarget to UART1. Then application code can use printf() and getchar() to output to or input from serial port.

6-9-2 UART FIFO

In this sample program, UART0 sent the data "TMPM3651" to UART1 use FIFO, and at same time UART0 receive the data "TMPM3652" 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 = "TMPM3652",and RxBuffer1 = "TMPM3651".

6-9-3 SIO

This demo will show synchronously transfer and receive usage of SIO module in TMPM365
It use the channel SIO0, SIO1 and transfer data synchronously between them.(connect TXD0 with RXD1, connect TXD1 with RXD0, connect sclk0 with sclk1)

6-10 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 TOSHIBA TMPM365 Evaluation Board which will demonstrate the main feature of TMPM365MCU.

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:

```
\---TMPM365
    +---Libraries
    |   +---TX03_CMSIS
    |   |   |   system_TMPM365.c
    |   |   |   system_TMPM365.h
    |   |   |   TMPM365.h
    |   |   \---startup
    |   |       +---arm
    |   |       |       startup_TMPM365.s
    |   |       \---iar
    |   |               startup_TMPM365.s
    |   \---TX03_Periph_Driver
    |       +---inc
    |       |       tmpm365_adc.h
    |       |       tmpm365_cg.h
    |       |       tmpm365_dmac.h
    |       |       tmpm365_fc.h
    |       |       tmpm365_gpio.h
    |       |       tmpm365_sbi.h
    |       |       tmpm365_tmrb.h
    |       |       tmpm365_uart.h
    |       |       tmpm365_wdt.h
    |       |       tx03_common.h
    |       \---src
    |       |       tmpm365_adc.c
    |       |       tmpm365_cg.c
    |       |       tmpm365_dmac.c
    |       |       tmpm365_fc.c
    |       |       tmpm365_gpio.c
    |       |       tmpm365_sbi.c
    |       |       tmpm365_tmrb.c
    |       |       tmpm365_uart.c
```

```
|       |       tmpm365_wdt.c
\---Project
    +---Examples                //only 1 examples listed here
    |   +---ADC
    |   |   \---ADC_Data_Read
    |   |       +---App
    |   |       |       main.c
    |   |       +---IAR
    |   |       |       ADC_Data_Read.ewd
    |   |       |       ADC_Data_Read.ewp
    |   |       |       ADC_Data_Read.eww
    |   |       \---KEIL
    |   |                   ADC_Data_Read.uvopt
    |   |                   ADC_Data_Read.uvproj
    |   \---Workspace
    |       +---IAR
    |       |       Examples_for_M365_Driver.eww
    |       \---KEIL
    |                   Examples_for_M365_Driver.uvmpw
    \---Template
        +---IAR
        |       TMPM365_flash.icf
        \---KEIL
                tmpm365.sct
```

7-1 ADC

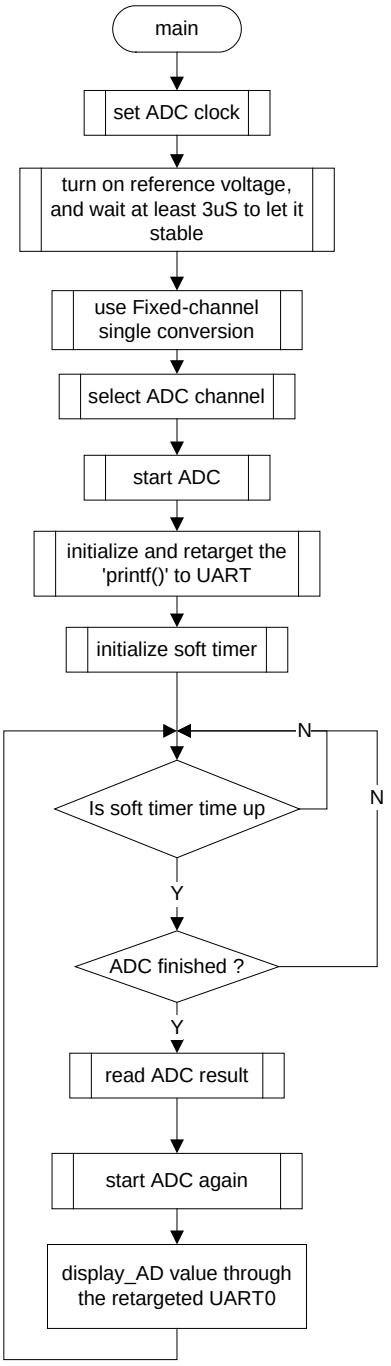
7-1-1 Example: ADC Data Read

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

The example includes:

1. ADC configuration and initialization
2. Start AD conversion by software and read AD result

- **Flowchart:**



- **Code and Explanation for the Example**

At first, set ADC clock, turn on reference voltage,

```
/* 1. set ADC clock */
ADC_SetClk(ADC_CONVERSION_81_CLOCK,
                ADC_FC_DIVIDE_LEVEL_16);

/* 2. turn on reference voltage, and wait at least 3uS to let it stable */
ADC_SetVref(ENABLE);
```

```
timeUp = 300U;
while( timeUp ) {
    timeUp -- ;
}
```

Then set to use fixed channel single conversion, and select AD channel:

```
/* 3. use Fixed-channel single conversion */
ADC_SetScanMode(DISABLE);
ADC_SetRepeatMode(DISABLE);

/* 4. select ADC channel */
ADC_SetInputChannel(ADC_CHANNEL);
```

After that, start ADC:

```
/* 5. now start ADC */
ADC_Start();
```

To use the standard “printf()” and soft timer, we need to do some initialize:

```
/* initialize and retarget the 'printf()' to UART */
Retarget_Init();

/* initialize soft timer */
softTimer_Init();
```

After started AD conversion, check ADC module state. When AD conversion is finished, call ADC_Display() and start another AD conversion.

```
while (1U) {
    softTimer_Run();
    if(softT.flag_TimeUp) {
        softT.flag_TimeUp = 0U;

        /* check ADC module state */
        adcState = ADC_GetConvertState();
        if (adcState.Bit.NormalComplete ) {
            /* read ADC result when it is finished */
            adResult = ADC_GetConvertResult(ADC_REG_RESULT);
            myResult = (uint32_t)adResult.Bit.ADResult ;
            ADC_Start();
            display_AD(myResult);
        }
    } else {
        /* Do nothing */
    }

}
```

In display_AD() function, use the standard “printf()” to send ADC results to PC:

```
printf("ADC Value is %4d\n", myResult);
```

7-2 CG

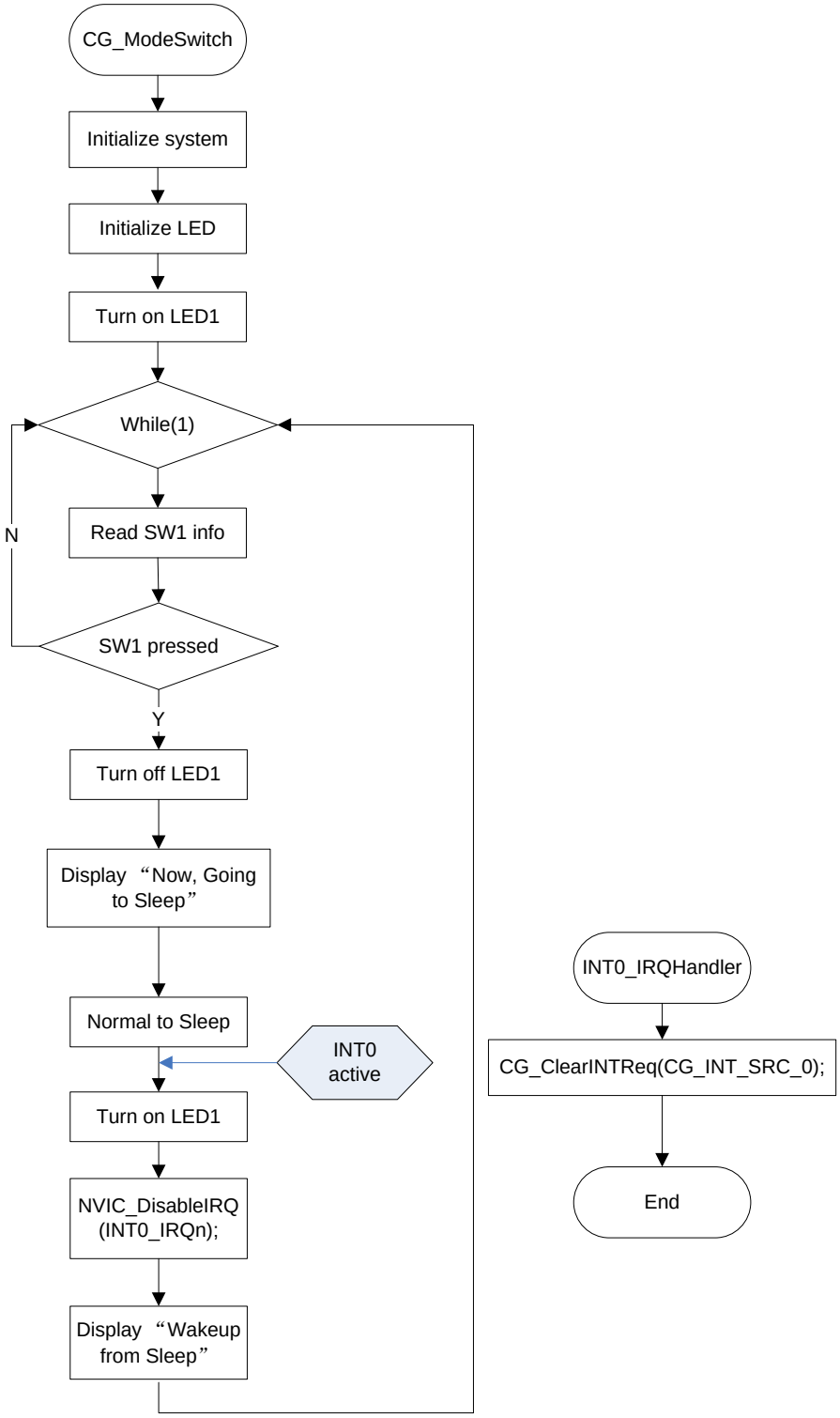
7-2-1 Example: Power mode change

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.

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

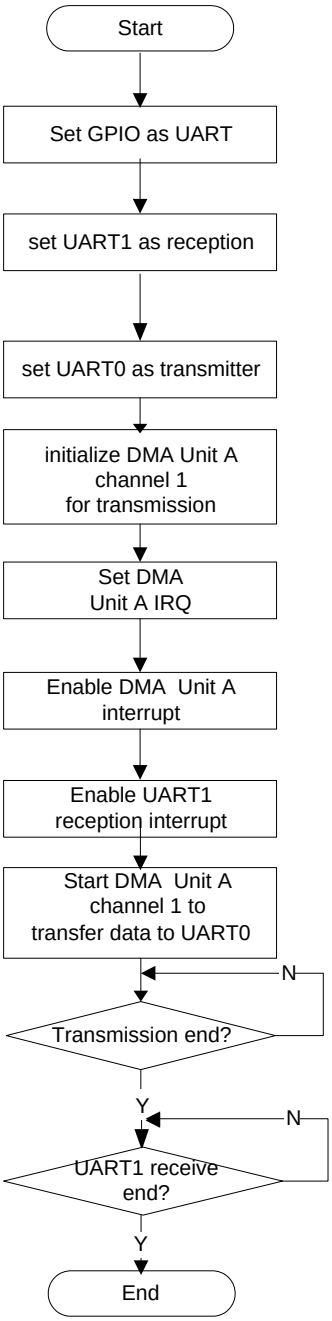
7-3-1 Example: Memory to peripheral

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 */
}
```

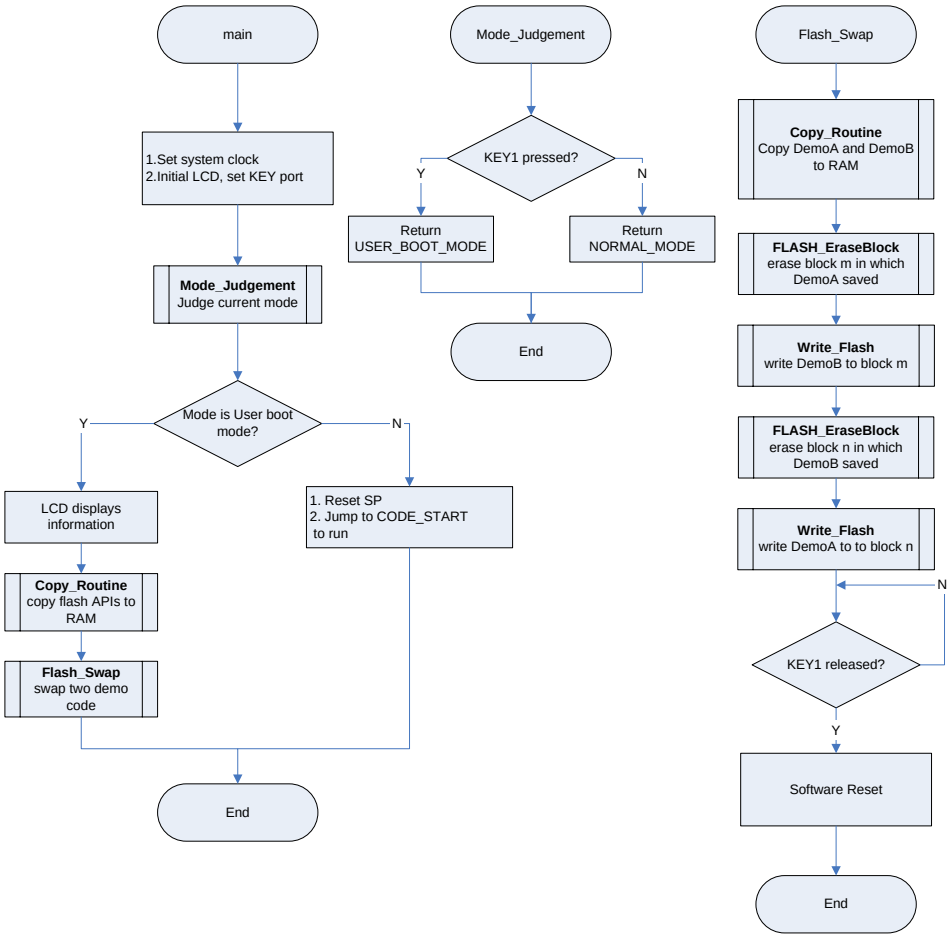
7-4 FLASH

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

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 initialize LED and switch. SW0 is used to judge current mode when reset and Flash operation information.

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

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

```
uint8_t Mode_Judgement(void)
{
    return (SW_Get(SW0) == 1U) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

If the SW0 is not turned on 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 (LED0 blinks).

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

If the SW0 is turned on when reset, the routine will enter user boot 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.

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

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

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

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

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

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

After SW0 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，then Demo B will run (LED1 blinks).

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

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

Flash memory operation function FLASH_EraseBlock() will erase a specified block automatically. The block is specified by parameter “block_addr”. Firstly, 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 (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;
                break;
            } 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) 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 4~67 */
      *addr3 = *source;
      source++;
}
```

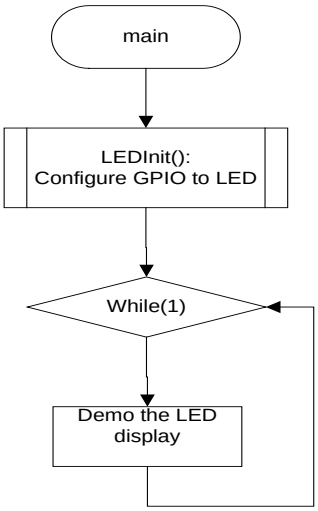
7-5 GPIO

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

The example includes:

1. GPIO initialization
2. Write data to GPIO

- **Flow chart:**



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

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

```
GPIO_SetOutput(LED_DATA_PORT,LED_ALL);
GPIO_WriteData(LED_DATA_PORT,led_off);
```

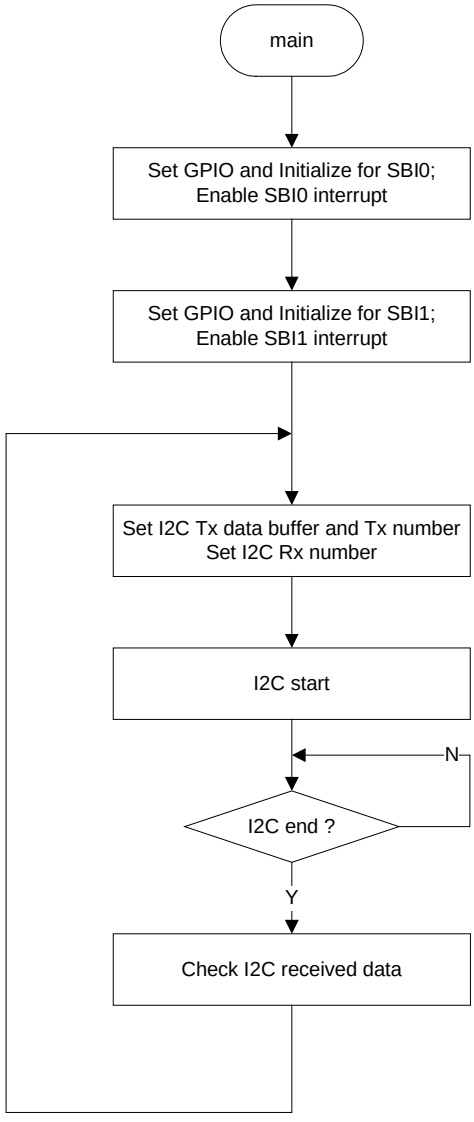
In the While process, do the LED demo: LED on one by one.

7-6 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 */
    }
}
```

7-7 TMRB

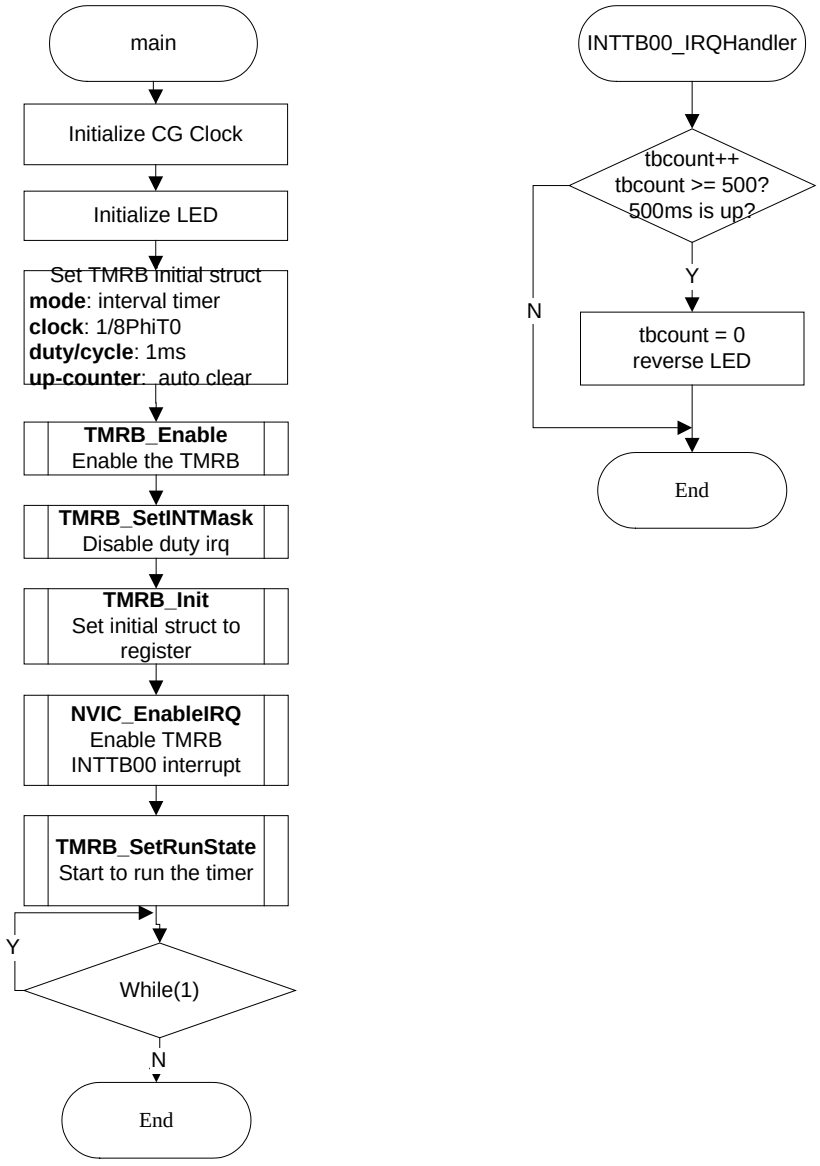
7-7-1 Example: General Timer

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.

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

Prepare TMRB initialization structure, fill TMRB mode, clock, up-counter clear method, trailing timing and leading timing. This demo will set trailing timing and leading timing 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_SetINTMask(TSB_TB0,TMRB_MASK_MATCH_LEADINGTIMING_INT);/*le
ading timing is not used in TMRB_INTERVAL_TIMER mode */
TMRB_Enable(TSB_TB0);                        /* enable the TMRB0 */
TMRB_Init(TSB_TB0, &m_tmrb);                 /* initial the TMRB0 */
NVIC_EnableIRQ(INTTB0_IRQn);                /* enable INTTB0 interrupt */
TMRB_SetRunState(TSB_TB0, TMRB_RUN);     /* run TMRB0*/
```

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) {
        Led_Off(LED0);
    } else {
        Led_On(LED0);
    }
} else {
    /* Do nothing */
}
```

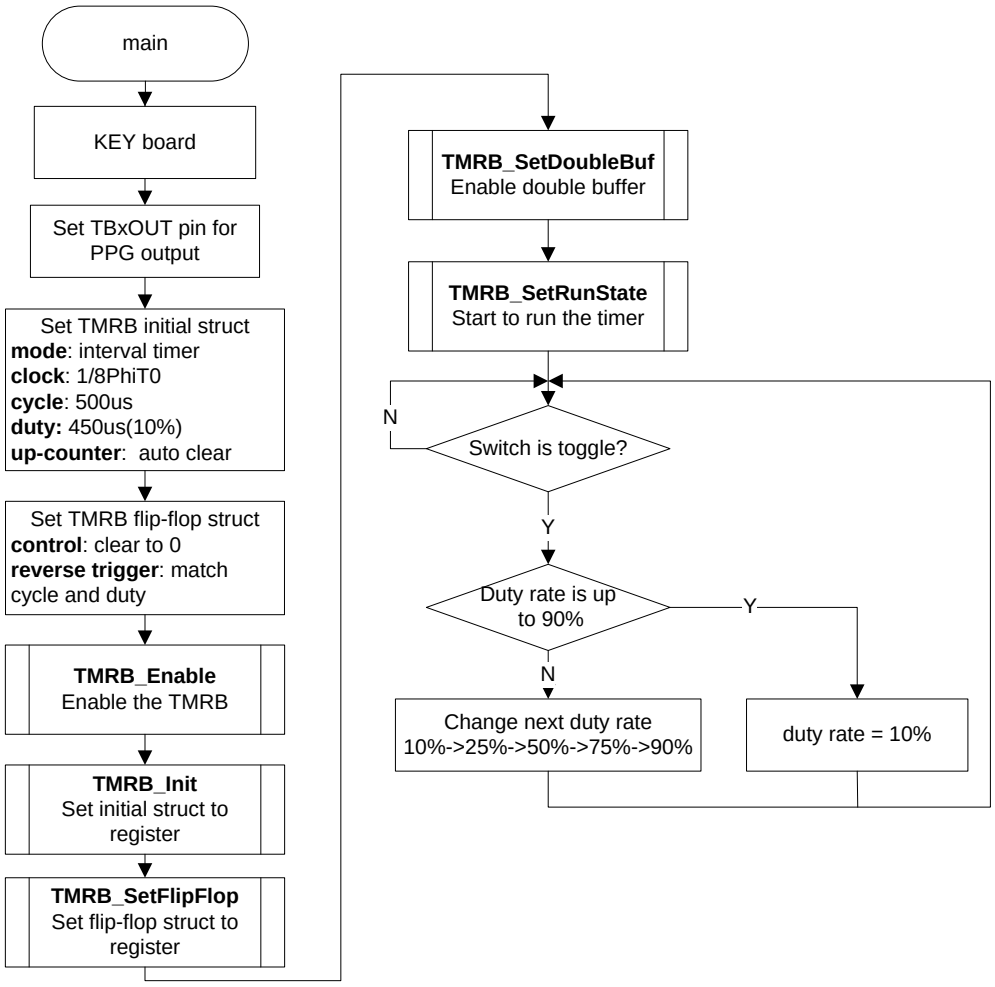
7-7-2 Example: PPG Output

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 leading timing 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, trailing timing and leading timing into it. This demo will set trailing timing 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;                    /* trailing timing is 500us
*/
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;       /* up-counter auto clear */
    m_tmrb.LeadingTiming  =  LeadingTiming[Rate];                    /*  leading
timing, initial value 10% */
```

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

```
    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 leading timing 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 leading timing 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 leading timing 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 leading timing array

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

7-8 SIO/UART

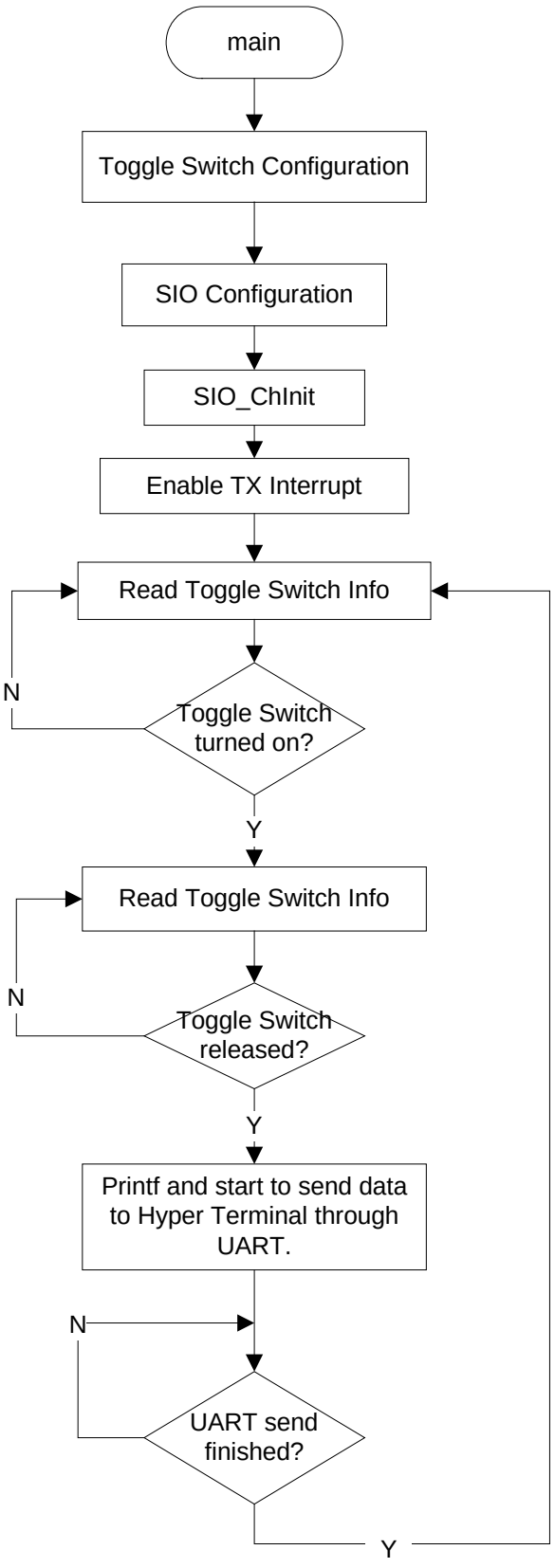
7-8-1 Example: Retarget

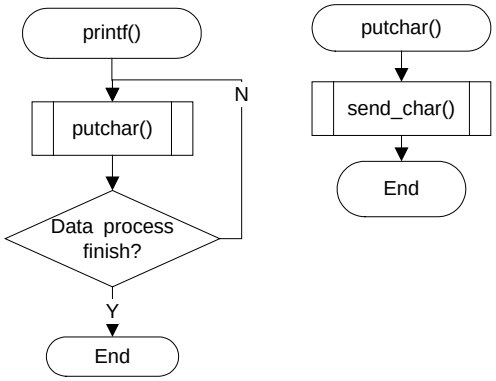
This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:

1. UART configuration and initialization.
2. UART TX process.
3. Use UART1 TX interrupt to send data.
4. Retarget printf() to UART1.

- **Flowchart**


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



- **Code and Explanation for the Example**

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

```
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;

/* configure SIO1 for reception */
UART_Enable(UART_RETARGET);
myUART.BaudRate = 115200U;    /* baud rate = 115200 */
myUART.DataBits = UART_DATA_BITS_8; /* no handshake, 8-bit data, clock by
baud rate generator */
myUART.StopBits = UART_STOP_BITS_1; /* 1-bit stop, LSB,    W-buff enable */
myUART.Parity = UART_NO_PARITY;
myUART.Mode = UART_ENABLE_TX;
myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
UART_Init(UART_RETARGET, &myUART);
```

After above setting, enable UART TX interrupt.

```
NVIC_EnableIRQ(RETARGET_INT)
```

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

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

The rest process of data flow is finished in ISR of UART1 TX interrupt routine
UART1 TX interrupt routine:

```
void INTTX1_IRQHandler(void)
{
    if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
        UART_SetTxData(UART_RETARGET,          gSIOTxBuffer[gSIORdIndex++]);
/* send data */
        fSIO_INT = SET;              /* SIO1 INT is enable */
    } else {
        /* disable SIO1 INT */
        fSIO_INT = CLEAR;
        NVIC_DisableIRQ(RETARGET_INT);
        fSIOTxOK = YES;
    }
```

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

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

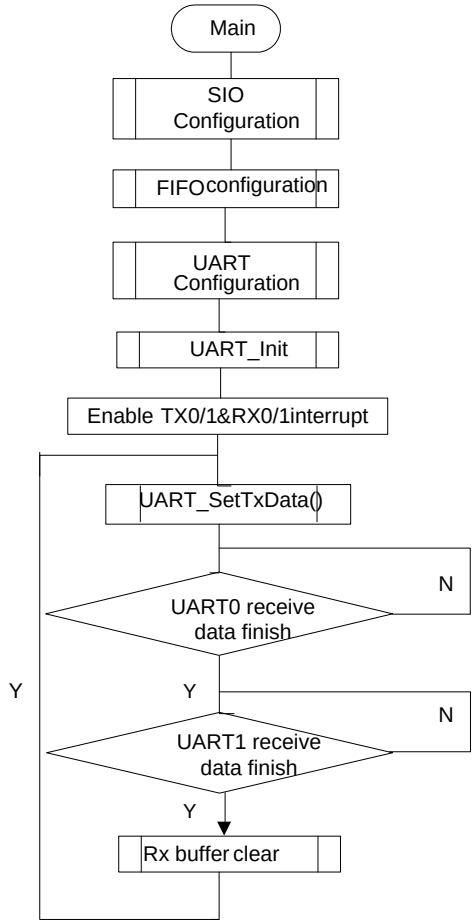
```
#if defined ( __CC_ARM      )          /* RealView Compiler */
struct __FILE {
    int handle;                        /* Add whatever you need here */
};
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE * f)
#elif defined ( __ICCARM__ )       /*IAR Compiler */
int putchar(int ch)
#endif
{
   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

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

    return ch;
}
```

7-8-2 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);
	}
}
```

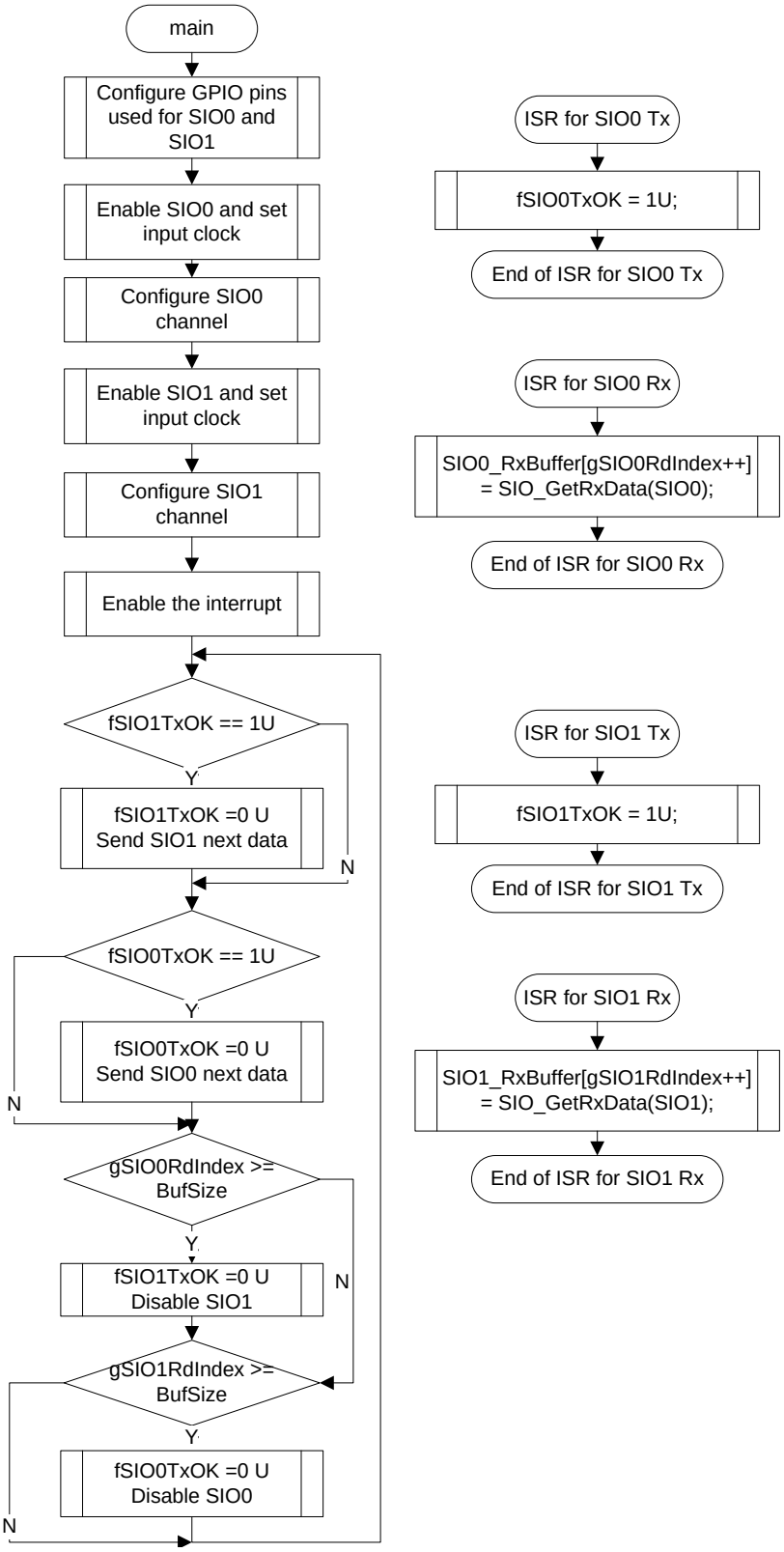
7-8-3 Example: SIO

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);
}
```

7-9 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();
```