TOSHIBA

TX00 Peripheral Driver Usage Example (TMPM066/067/068)

Rev 1.0

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M066UE-01xE

RESTRICTIONS ON PRODUCT USE

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

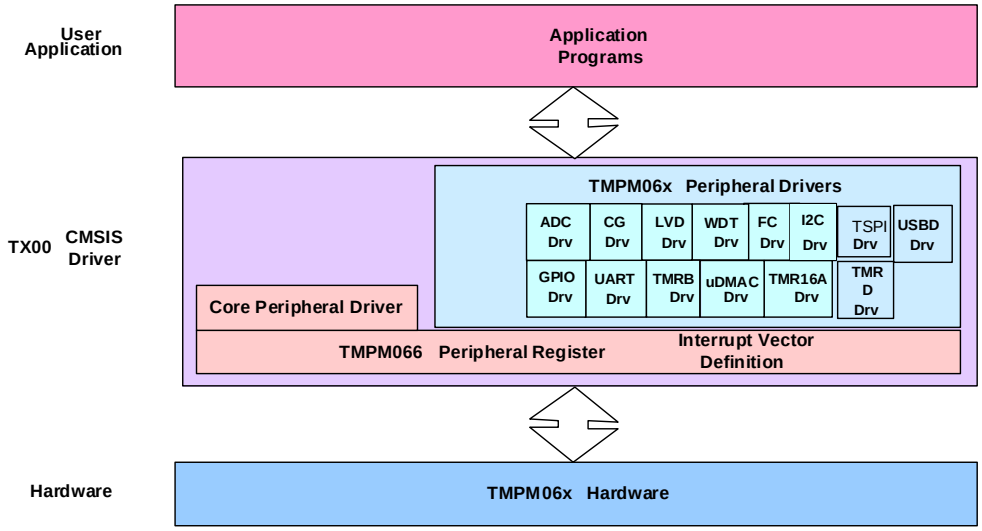
Index

1 General description

The example programs described hereafter are specially designed for TOSHIBA TMPM06x 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 TX00 peripheral driver as following.



3 Build-in hardware usage

Hardware	Channel	Use presence, use
CG	Clock Gear	Used for CG demo
	PLL	PLL on (2x)
Standby mode	-	Used for CG demo (STOP1 mode/ IDLE mode)
SysTick	-	Unused
Watch dog timer (WDT)	-	Used for WDT
Interrupt (INT)	INT0	Unused
	INT1	Used for CG demo to wake up system from idle mode
	INT2	Unused

	INT3	Unused
	INT4	Unused
	INT5	Used for CG demo to wake up system from stop mode
SIO	SIO0	Used for SIO demo/ UART FIFO demo
	SIO1	Used for SIO demo/ UART FIFO demo/ UART retarget demo
16-bit timerB	TMRB0	Used for TMRB: General Timer, PPG Output
	TMRB1	Unused
	TMRB2	Unused
	TMRB3	Unused
	TMRB4	Unused
	TMRB5	Unused
	TMRB6	Unused
	TMRB7	Unused
16-bit timerA	TMR16A0	Used for TMR16A: General Timer
	TMR16A1	Unused
LVD	-	Used for LVD demo
10-bit A/D converter	AIN0	Used for ADC demo
	AIN1	Unused
	AIN2	Unused
	AIN3	Unused
	AIN4	Unused
	AIN5	Unused
	AIN6	Unused
	AIN7	Unused
uDMAC	-	Used for DMAC demo
TSPI	TSPI0	Used for TSPI demo
TMRD	TMRD	Used for TMRD demo
I2C	I2C0	Used for I2C demo: receiver, transmitter
	I2C1	Used for I2C wake up demo
USB2.0 full-speed device	-	Unused
INTIFAO	-	Used for Idle demo/ Mode Switch demo/ Stop1 demo/ I2CS demo
AOREG	-	Used for Stop1 demo
I2CS	I2C1	Used for I2C wake up demo
INTIFSD	-	Used for Stop1 demo/ TMRD: General Timer/ WDT demo

4 Pin Usage

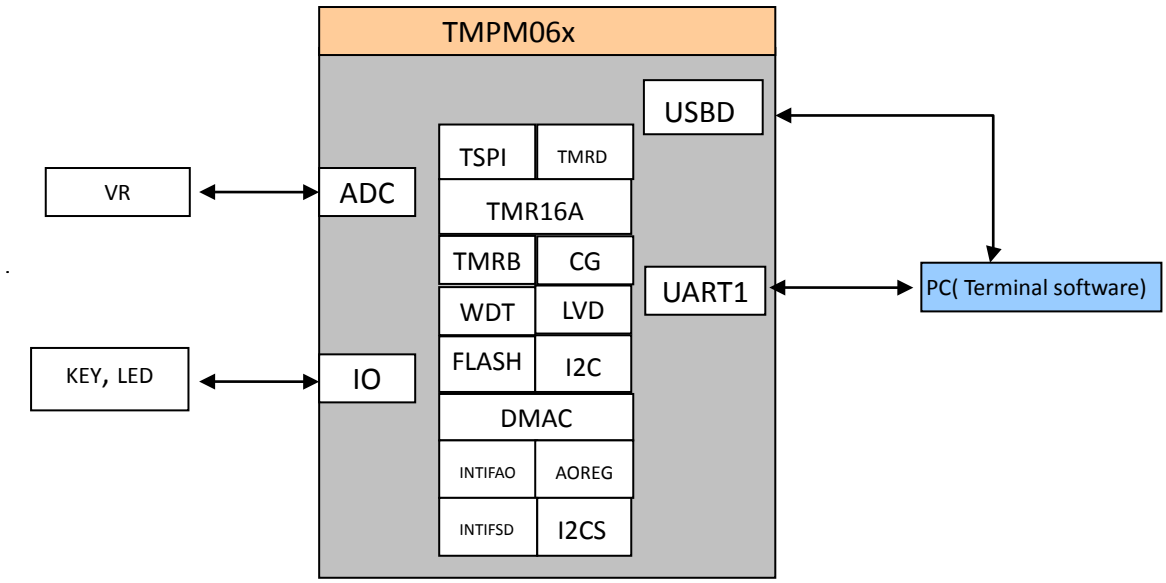
The example programs are tested on TMPM066FWUG evaluation board
Following is pin usage for example programs

Pin No.	Name	Usage
1	RESET	Reset input pin
2	MODE	MODE pin MODE pin must be connected to GND.
3	PB3	SW1(board S5)
4	PB2	SW0(board S4)
5	PB1	LED1(board D9)
6	PB0	LED0(board D10)
7	AVSS	GND pin for the analog circuit.
8	AVDD3	Power supply pin for the analog circuit.
9	PA0	AIN0(board RV1)
10	PA1	Unused
11	PA2	Unused
12	PA3	Unused
13	PA4	Unused
14	PA5	INT1
15	PA6	Unused
16	PA7	Unused
17	PC5	VBUS
18	PC4	SC0SCLK
19	PC3	SC0RXD
20	PC2	SC0TXD
21	PC1	I2C0SDA
22	PC0	I2C0SCL
23	PF7	Unused
24	PF6	Unused
25	PF5	Unused
26	PF4	Unused
27	PF3	Unused
28	PF2	TSPIRXD
29	PF1	TSPITXD
30	PF0	INT5
31	DVDD3	Power supply pin for the digital circuit.
32	DVSS	GND pin for the digital circuit.
33	PD0	Unused
34	PD1	TB0OUT/TDA0OUT1

35	PD2	Unused
36	PD3	TDA1OUT1
37	PD4	Unused
38	PD5	Unused
39	PH0	Unused
40	PH1	Unused
41	PH2	Unused
42	PH3	Unused
43	PE0	SC1SCLK
44	PE1	SC1RXD(board CN17)
45	PE2	SC1TXD(board CN17)
46	RVDD3	Power supply pin for the regulator.
47	PG0	I2C1SCL
48	PG1	I2C1SDA
49	PE3	BOOT
50	PE4	Unused
51	PE5	Unused
52	REGOUT	Pin connected with the capacitor for the regulator.
53	PJ0	Unused
54	PJ1	Unused
55	DVSS	GND pin for the digital circuit.
56	USBDM	USBDM
57	USBDP	USBDP
58	DVDD3	Power supply pin for the digital circuit.
59	PJ2	Unused
60	PJ3	Unused
61	DVDD3	Power supply pin for the digital circuit.
62	X1	Connected to a high-speed oscillator
63	DVSS	GND pin for the digital circuit.
64	X2	Connected to a high-speed oscillator

5 Development Environment

Following is development environment:



1. Hardware board:
   TMPM06x evaluation board

2. Development tool:
   - IAR:

   1) ICE: OnBoard JTAG mounted on the Evaluation Board
   2) IDE: IAR Embedded workbench 7.70.1 version

6 Functional description

6-1 Operation mode

There are three operation modes for TMPM06x: NORMAL, IDLE, STOP1 mode.

- **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.

IDLE and STOP1 modes are low power mode.
To shift to lower power mode, in stand-by register CGSTBYCR<STBY[2:0]>, select the IDLE or STOP1 mode, and run the WFI (Wait For Interrupt) command.

- **STOP1 mode:**

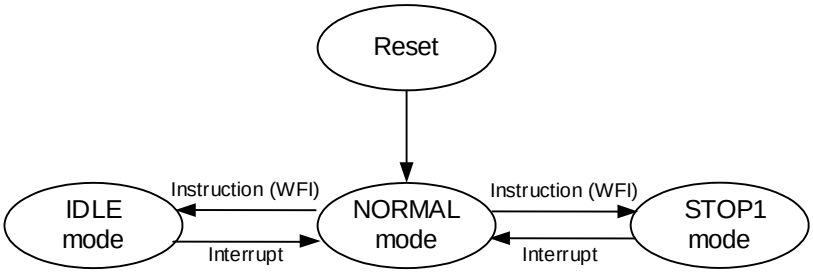
All the internal circuits including the internal oscillator are brought to a stop in STOP1 mode. And the internal oscillator activates the clock after releasing the STOP1 mode then transit to the Normal mode. The STOP1 mode enables to select the pin status by setting the port register.

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

- **IDLE mode:**

Only the CPU is stopped in this mode. Each peripheral function has one bit in its control register for enabling or disabling operation in the IDLE mode. When the IDLE mode is entered, peripheral functions for which operation in the IDLE mode is disabled stop operation and hold the state at that time.

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



6-2 ADC

There is a potentiometer that connected to PA0/AIN0. The voltage on it will be measured and print to stdout. As stdout is retargeted to UART, connect UART1 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. Toggle switch to switch the power mode.

Current mode	Action (Switch)	Operation	LED display
NORMAL	Turn on SW1	NORMAL→ STOP1	UART prints “NORMAL MODE” →”STOP MODE” LED 0,1 turns off
STOP1	Turn off SW1 at first, Turn on SW0	STOP1 →NORMAL	UART prints “STOP MODE” →” NORMAL MODE” LED 0,1 turns on

6-3-2 STOP1

Change the CPU operation mode between normal mode and stop1 mode.

Current mode	Action (Switch)	Operation	LED display
NORMAL	Turn off CG_SW	NORMAL→ STOP1	LCD displays “STOP MODE” LED 0,1 turns off
STOP1	Turn on CG_SW at first, then turn on External Interrupt SW*.	STOP1 →NORMAL	LCD displays “NORMAL MODE” LED 0,1 turns on

***Note**: CG_SW is SW1, External Interrupt SW is SW0.

6-3-3 IDLE

Change the CPU operation mode between normal mode and idle mode.

Current mode	Action (Switch)	Operation	LED display
NORMAL	Turn off CG_SW	NORMAL→ IDLE	LED_SW is on, others is off
IDLE	Turn on CG_SW* at first, then turn on External Interrupt SW*.	IDLE →NORMAL	LED 0 is on, others is off

***Note**: CG_SW is SW1, External Interrupt SW is SW0. LED_SW is LED1.

6-4 uDMAC

This demo will show how to reserve 1K RAM area for control data of each uDMAC unit, and use DMA with software trigger to transfer data from RAM area "src" to "dst".

The driver example step:

1. Initialize uDMAC unit A by setting transfer type as burst type, enable channel, enable channel in mask setting, use primary data, use normal priority.
2. Set primary base address in the head of the reserved 1K RAM area.
3. Configurate the setting data for software trigger and fill it to 'control data' area.
4. Enable DMA unit A after all configuration is finished.
5. If DMAC_BASIC mode is selected, need to trigger it again and again until transfer is finished. If DMAC_AUTOMATIC mode is selected, only need to trigger it once before transfer is finished.

Judge if transfer is finished, then compare the data in destination with source

6-5 FLASH

This example 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) is programmed in flash block in advance.

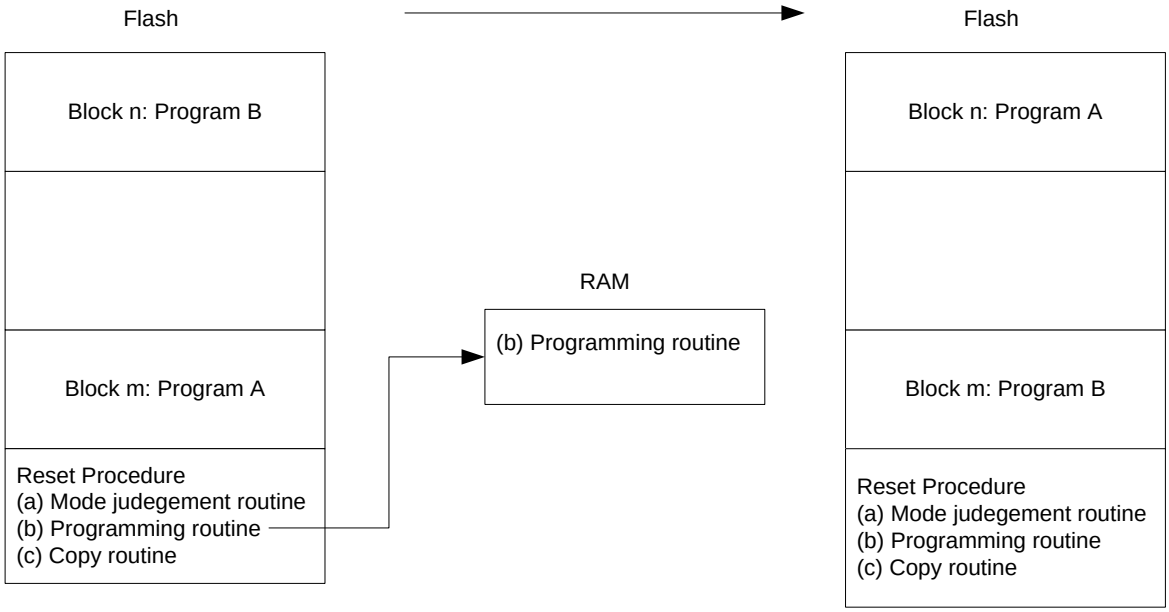
—Program A/B (A: LED0 blinks, B: LED1 blinks)

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.

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.

Swap when finished then LED0 lights and LED1 lights.

(3) The demo will reset by software automatically.

Then program B stored in flash runs.

LED1 blinks.

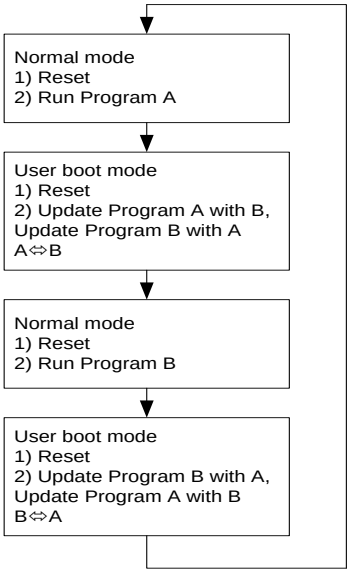
(4) Press RESET button again while SW0 is turned on, and release SW0.

Same as step (2).

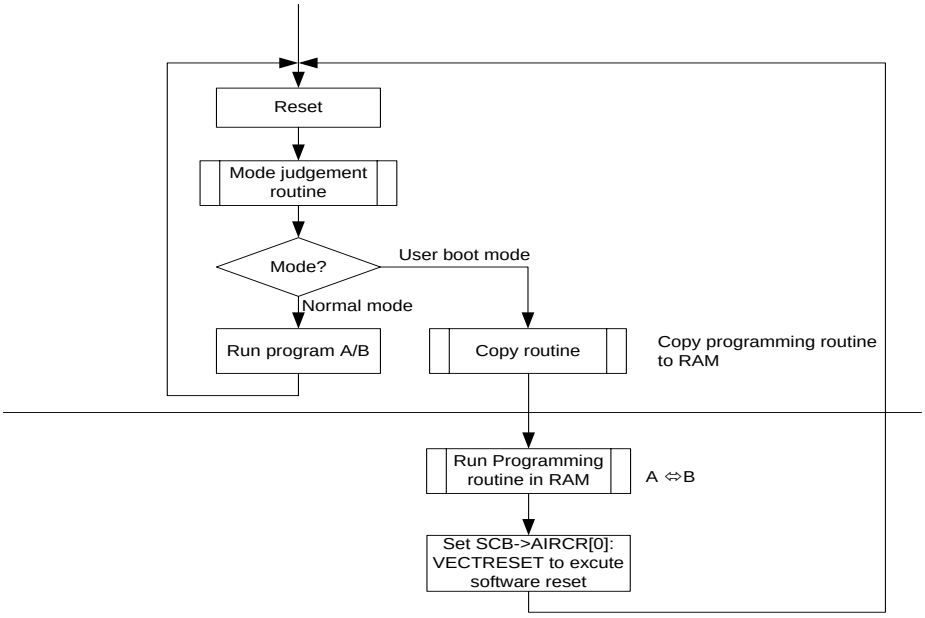
(5) The demo will reset by software automatically.

Then program A stored in flash runs.

LED0 blinks.



Demo process flow



6-6 GPIO

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

6-7 I2C

6-7-1 I2C Slave

This function intends to support the I2C bus slave mode.
Connect two I2C buses on evaluation board.
Work as I2C slave.

Use I2C interrupt to handle I2C bus read.
If the received address matches its slave address specified at I2CAR or is equal to the general-call address, the I2C pulls the SDA line to the "Low" level during the ninth clock and outputs an acknowledge signal.

I2C Slave receives “TOSHIBA” from Master and UART prints it.

I2C (slave) demo start,

```
K1: I2C SLAVE RECV
```

When SW1 is pressed, the string “TOSHIBA” that got from the I2C (slave) received buffer will be printed.

```
TOSHIBA
MASTER I2C to SLAVE I2C OK
```

6-7-2 I2C Master

This function intends to support the I2C bus master mode.
Connect two I2C buses on evaluation board.
Work as I2C master.

Use I2C interrupt to handle I2C bus write.

Address of I2C: Any.
I2C Master sends “TOSHIBA” to Slave (I2C).

I2C (master) demo start,
When SW1 is pressed, the string “TOSHIBA” will be sent from I2C (master) to I2C (slave).

6-7-3 I2CS wake up

This function is used in salve mode, if the frame address on the I2C bus matches a self-address (slave address) in low-power consumption mode (STOP1), an interrupt(INTI2CS) occurs to clear low-power consumption mode.

6-8 LVD

This application implements power supply voltage status detecting by LVD.
When the power supply voltage is lower than the detection voltage, UART prints “LOWER”.
When the power supply voltage is upper than the detection voltage, UART prints “UPPER”.

6-9 TMR16A

Set a value of T16A0RG<RG>, then start count-up, if the counter value matches with a value of T16A0RG<RG>, the counter will be cleared to "0x0000" and continued to count-up and the same time a match detection interrupt INTT16A0 will be output.

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

6-10 TMRD

6-10-1 Interval time

This function implements a general timer by using TMR0 16-bit interval timer.
Timer cycle is set to 500us.
Use this timer for all LEDs blinking and the period is 1s (500ms ON and 500ms OFF)

6-10-2 Interlock PPG output

This function implements interlock PPG output by using TMR0/1.
PPG cycle is set to 500us, and duty is 50%.(250us H level, 250us L level).
Phase 0 is set to fast than phase 1, and the phase shift (delay) (θ) is 120 degrees. The phase 01 and 11 can be observed by oscilloscope from PD1/TDA0OUT1 and PD3/TDA1OUT1.

6-11 TMRB

6-11-1 General Timer

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

6-11-2 PPG Output

Use key SW1 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 upon every SW1 pressed.
10% → 25% → 50% → 75% → 90% → 10%

***Note**: PD1 connect with oscilloscope to observe the PPG output.

6-12 TSPI

6-12-1 Basic transfer

This demo will show basic usage of TSPI module in M06x.

It use single transfer on TSPI0, (connect its Tx with Rx). The LEDs will blink to show the data is being received. User can define BITRATE_MIN to see the blink effect.

Note: Connect pin F1(PF1, TSPI_TXD) to pin F2(PF2, TSPI_RXD).

6-13 UART

6-13-1 UART

This example intends to retarget the stdin and stdout of the C lib.
Stdin and stdout are retargeted to UART. Then application code can use printf() to output to serial port.
***Note**: This UART channel is limited by hardware, and the follow example used UART1.

6-13-2 UART FIFO

In this sample program, UART0 sent the data "TMPM0661" to UART1 use FIFO, and at same time UART0 receive the data "TMPM0662" 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 = "TMPM0662",and RxBuffer1 = "TMPM0661".

6-13-3 SIO

This demo will show synchronously transfer and receive usage of SIO module in TMPM06x.
It uses the channel SIO0, SIO1 and transfer data synchronously between them. (Connect TXD0 with RXD1; connect TXD1 with RXD0; connect sclk0 with sclk1)

6-14 WDT

The watchdog timer cannot be used in the STOP mode while high-speed frequency clock is stopped.
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 and DEMO2 is defined by macro definition.
   DEMO1:
   When timer is overflow, NMI interrupt is generated (LED1 blink once) and then WDT will be cleared.
   DEMO2:
   WDT clear code will be written to the watchdog timer control register, and watch dog timer counter will be cleared. (LED0 blink all the time)

7 Software

This software project creates the sample applications based on TMPM06x evaluation board which will demonstrate the main feature of TMPM06x 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
        │   tmpm06x_wdt_int.c
        │   tmpm06x_wdt_int.h
        │
        ├─TX00_CMSIS
        │ │ system_TMPM06x.c
        │ │ system_TMPM06x.h
        │ │ TMPM06x.h
        │ │
        │ └─startup
              startup_TMPM06x.s
        │
        ├─TX00_Periph_Driver
        │   ├─inc
        │   │   tmpm06x_cg.h
        │   │   tmpm06x_gpio.h
        │   │   tmpm06x_wdt.h
        │   │   TX00_common.h
        │   │
        │   └─src
        │       tmpm06x_cg.c
        │       tmpm06x_gpio.c
        │       tmpm06x_wdt.c
        │
        └─TMPM06x-EVAL
                led.c
                led.h
                sw.c
                sw.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm06x_wdt_int.c
        │
        ├─TX00_CMSIS
        │   system_TMPM06x.c
        │   startup_TMPM06x.s
        │
```

```
├─TX00_Periph_Driver
│     tmpm06x_cg.c
│     tmpm06x_gpio.c
│     tmpm06x_wdt.c
│
└─TMPM06x-EVAL
       led.c
       sw.c
```

7-1 ADC

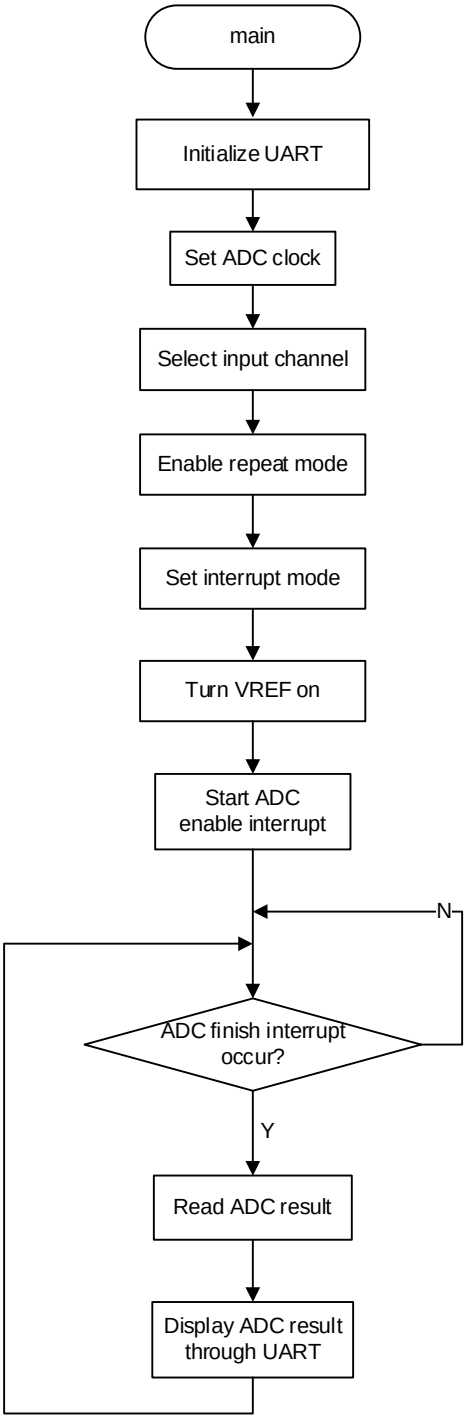
7-1-1 Example: ADC Data Read

This is a simple example based on the TX00 Peripheral Driver (ADC, GPIO and UART).

The example includes:

1. ADC configuration and initialization
2. Start ADC in fixed-channel repeat mode and read AD result of AIN0

- **Flowchart**



- **Code and Explanation for the Example**

At first, setting the clock supply and performs software reset.

```
 /* enable clock supply to ADC */
CG_SetFcPeriphA(CG_FC_PERIPH_ADC, ENABLE);
CG_SetADCClkSupply(ENABLE);

ADC_SWReset();
 /* set ADC clock */
ADC_SetClk(ADC_CONVERSION_42_CLOCK, ADC_FC_DIVIDE_LEVEL_16);
```

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

```
/* select ADC input channel */
ADC_SetInputChannel(ADC_AN_0);

/* 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:

```
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.

```
ADC_ResultTypeDef ADC_Result = ADC_GetConvertResult(ADC_REG_0);
```

7-2 CG

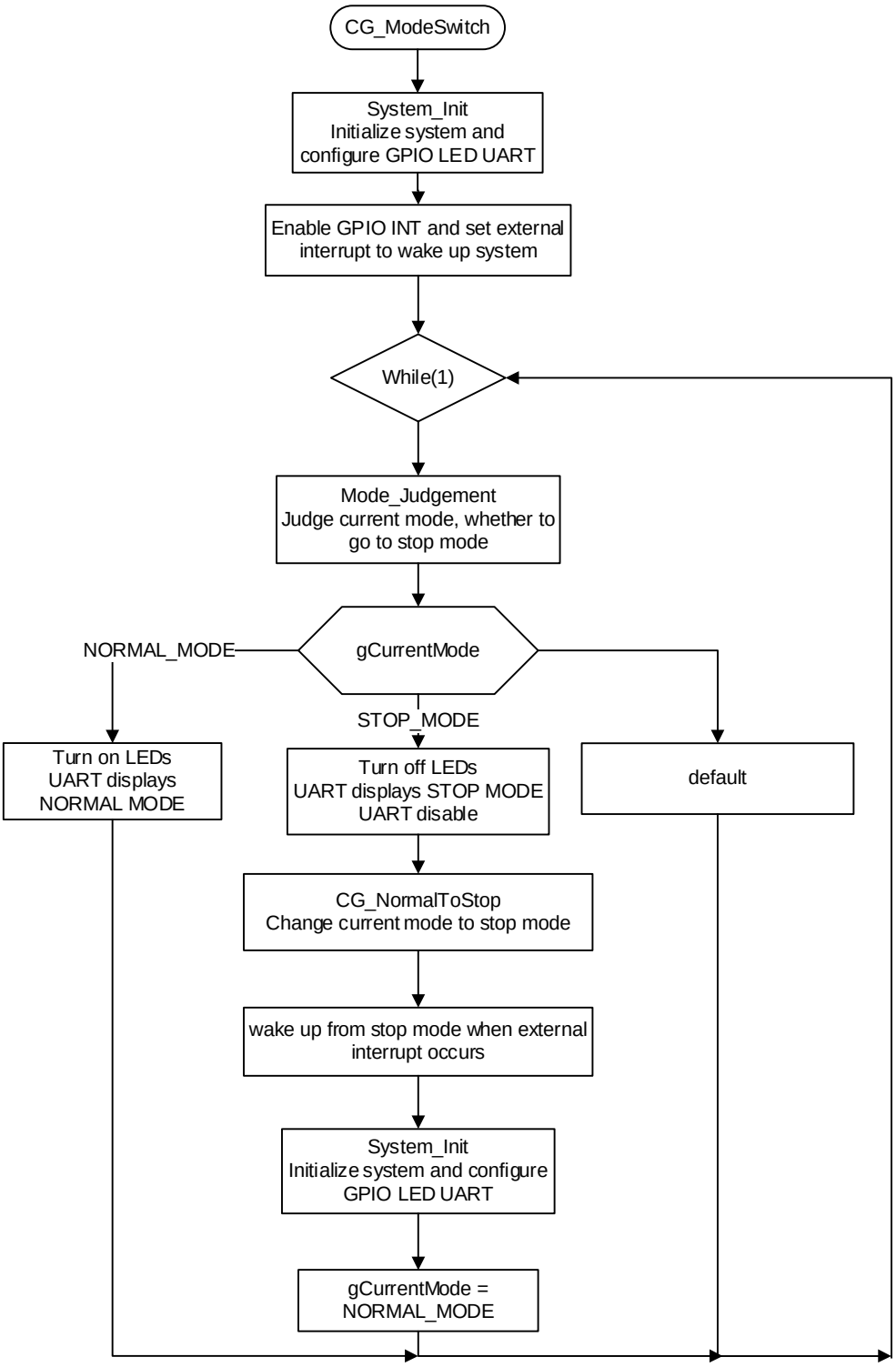
7-2-1 Example: Power mode change

This is a simple example based on the TX00 Peripheral Driver (CG, GPIO and UART).

The example includes:

1. Basic setup operation of CG
2. How to switch between normal mode and stop mode

- **Flowchart**



- **Code and Explanation for the Example**

The following simple example is based on TX00 Peripheral Driver (CG, GPIO and UART), which will switch between normal mode and stop mode.

Configure external interrupt to wake up system

Confiure external interrupt INT5 to wake up system. Clear interrupt pending request, then enable INT5.

```
    /* for M066, Int5 is PF0 = pin30, internal pull up */
    GPIO_SetInput(GPIO_PF, GPIO_BIT_0);
    GPIO_SetPullUp(GPIO_PF, GPIO_BIT_0, ENABLE);

    /* Configure GPIO to LED */
    LED_Init();
    SW_Init();

    /* Disable interrupts */
    __disable_irq();

    /* Set external interrupt to wake up system */
    INTIFAO_SetSTBYReleaseINTSrc(INTIFAO_INT_SRC_5,INTIFAO_INT_ACTI
VE_STATE_FALLING,ENABLE);
    NVIC_ClearPendingIRQ(INT5_IRQn);
    NVIC_EnableIRQ(INT5_IRQn);
    __enable_irq();

    /* Configure UART */
    UART_SWReset(UART_RETARGET);
    hardware_init(UART_RETARGET);
```

UART disable

Before shifting to STOP mode, disable UART.

```
      case STOP_MODE:
         LED_Off(LED_ALL);
         common_uart_disp("STOP MODE\r\n");
         do {
              regval = UART_RETARGET->MOD2;
              regval &= TX_MASK;
         } while(regval != TX_END);
         UART_Disable(UART_RETARGET);
```

Setup to enter STOP mode

Prepare for entering stop mode. Set warm up time, use __WFI_wait() instruction to enter stop mode.

```
  uint32_t regval;
  WorkState st = BUSY;

  TSB_WD->MOD = WDT_DISABLE;
  TSB_WD->CR = WDT_DISABLE_CODE;
  while((TSB_FC->SR & SR_BUSY_MASK) == SR_BUSY_FLAG){
  }
    do {
        st = CG_GetWarmUpState();
  } while (st != DONE);
  /* Set CG module: Normal ->Stop mode */
  CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_INT_HIGH,
                                                CG_WUODR_INT);
  /* Set low power consumption mode stop1 */
  CG_SetSTBYMode(CG_STBY_MODE_STOP1);
  CG_SetFcSrc(CG_FC_SRC_FOSC);
  while(CG_GetFcSrc() == CG_FC_SRC_FPLL){
  }
  CG_SetPLL(DISABLE);
  /* Set fosc source */
  CG_SetFoscSrc(CG_FOSC_OSC_INT);
  /* Wait for <OSCSEL> to become "0" */
  while (CG_GetFoscSrc() == CG_FOSC_OSC_EXT) {
```

```
    }
    regval = (TSB_CG->OSCCR & CG_OSCCR_EOSCEN_UNUSE);
    TSB_CG->OSCCR = regval;
    while(TSB_CG->OSCCR != regval){
    }
    /* Enter stop mode */
    __WFI_wait();
```

Normal setup for CG

The following code is just an example for setting CG in normal mode.

```
     Result retval = ERROR;
     WorkState st = BUSY;

    CG_SetFcSrc(CG_FC_SRC_FOSC);
    while(CG_GetFcSrc() == CG_FC_SRC_FPLL){
    }
     CG_SetPLL(DISABLE);
     CG_SetFPLLValue(CG_12M_MUL_8_FPLL);
     /* Set warm up time */
     CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
CG_WUODR_PLL);
     CG_StartWarmUp();
     do {
         st = CG_GetWarmUpState();
     } while (st != DONE);
     retval = CG_SetPLL(ENABLE);
     if (retval == SUCCESS) {
           /* Set warm up time */
           CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
CG_WUODR_PLL);
           CG_StartWarmUp();
           do {
                st = CG_GetWarmUpState();
           } while (st != DONE);
          CG_SetFcSrc(CG_FC_SRC_FPLL);
     } else {
           /*Do nothing */
     }
     return retval;
```

Enable multiple clock circuit

Set to 24MHz with PLL.

```
    Result retval = ERROR;
    WorkState st = BUSY;

    CG_SetFcSrc(CG_FC_SRC_FOSC);
    while(CG_GetFcSrc() == CG_FC_SRC_FPLL){
    }
     CG_SetPLL(DISABLE);
     CG_SetFPLLValue(CG_12M_MUL_8_FPLL);
     /* Set warm up time */
     CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
                              CG_WUODR_PLL);
     CG_StartWarmUp();
     do {
         st = CG_GetWarmUpState();
     } while (st != DONE);
     retval = CG_SetPLL(ENABLE);
```

```
    if (retval == SUCCESS) {
        /* Set warm up time */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_EXT_HIGH,
                                    CG_WUODR_PLL);
        CG_StartWarmUp();
        do {
            st = CG_GetWarmUpState();
        } while (st != DONE);
       CG_SetFcSrc(CG_FC_SRC_FPLL);
    } else {
        /*Do nothing */
    }
    return retval;
```

7-2-2 Example: NORMAL <-> STOP1 mode change

This is a simple example based on the TX00 Peripheral Driver (CG, GPIO).

The example includes:

1. Basic setup operation of CG
2. How to switch between normal mode and stop1 mode

- **Code and Explanation for the Example**

```
int main(void)
{
```

Configure the I/O pins for Key, LED:

```
    SW_Init();                  /* Initial the SW */
    LED_Init();                 /* Initial the LED, all LED is off *

    GPIO_ExtIntSrc();          /* configure the External interrupt sw's GPIO */
```

When a reset factor is STOP1 mode release, configure interrupt to release standby mode.

```
    /* after release standby mode */
    if (is_StanbyRelease()) {
      LED_On(LED_ALL);
      INTIFAO_SetSTBYReleaseINTSrc(CG_ExtINTSrc,
                                          INTIFAO_INT_ACTIVE_STATE_H,ENABLE);
      NVIC_EnableIRQ(ExtINTSrc_IRQn);
```

When a reset factor is not STOP1 mode release, configure KSCAN and configure interrupt to release standby mode.

```
    } else {
        INTIFAO_ClearINTReq(CG_ExtINTSrc);
        NVIC_ClearPendingIRQ(ExtINTSrc_IRQn);
        NVIC_EnableIRQ(ExtINTSrc_IRQn);
    }
```

Read a status of LED_EXT in while() loop after turn on LED_EXT.
When the status of CG_SW is OFF, turn on LED_SW. When the status of CG_SW is

ON, enter STO1 mode after turn off LED_EXT.

```
    LED_On(LED_ALL);
    while (1U) {
        if (SW_Get(CG_SW) == 0) {

            LED_Off(LED_ALL);     /* LED is off before enter stop1 */
            enter_STOP1();

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

Prepare to enter STOP1 mode.
Select STOP1 mode as a standby mode. Enable internal oscillation after PLL is OFF.
Then enable port keep function.

```
void config_STOP1(void)
{
    uint32_t regval;
    WorkState st = BUSY;

   TSB_WD->MOD = WDT_DISABLE;
   TSB_WD->CR = WDT_DISABLE_CODE;
   while((TSB_FC->SR & SR_BUSY_MASK) == SR_BUSY_FLAG){
   }
    do {
        st = CG_GetWarmUpState();
   } while (st != DONE);
    /* Set CG module: Normal ->Stop mode */
    CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC_INT_HIGH,
CG_WUODR_INT);
    /* Set low power consumption mode stop1 */
    CG_SetSTBYMode(CG_STBY_MODE_STOP1);
   CG_SetFcSrc(CG_FC_SRC_FOSC);
   while(CG_GetFcSrc() == CG_FC_SRC_FPLL){
   }
   CG_SetPLL(DISABLE);
    /* Set fosc source */
    CG_SetFoscSrc(CG_FOSC_OSC_INT);
    /* Wait for <OSCSEL> to become "0" */
    while (CG_GetFoscSrc() == CG_FOSC_OSC_EXT) {
    }
   regval = (TSB_CG->OSCCR & CG_OSCCR_EOSCEN_UNUSE);
   TSB_CG->OSCCR = regval;
   while(TSB_CG->OSCCR != regval){
   }
}
```

Finally, enter STOP1 mode by executing __WFI() instruction.

```
    __WFI();
```

7-2-2 Example: NORMAL <-> IDLE mode change

This is a simple example based on the TX00 Peripheral Driver (CG, GPIO).

The example includes:

1. Basic setup operation of CG
2. How to switch between normal mode and IDLE mode

- **Code and Explanation for the Example**

```
int main(void)
{
```

Configure the I/O pins for Key and LED.

```
    SW_Init();
    LED_Init();

    GPIO_ExtIntSrc();
    INTIFAO_ClearINTReq(INTIFAO_ExtINTSrc);
    NVIC_ClearPendingIRQ(ExtINTSrc_IRQn);

    NVIC_EnableIRQ(ExtINTSrc_IRQn);
```

Read a status of LED_EXT in while() loop after turn on LED_EXT.
When the status of CG_SW is OFF, turn off LED_SW. When the status of CG_SW is ON, enter IDLE mode after turn on LED_SW.

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

        if (SW_Get(CG_SW) == 0U) {          /*SW is OFF*/
            LED_On(LED_SW);

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

            enter_IDLE();

        } else {
            LED_Off(LED_SW);

            LED_Off(LED_EXT);
        }
    }
}
```

Prepare to enter IDLE mode.
Select IDLE mode as a standby mode.

```
INTIFAO_SetSTBYReleaseINTSrc(INTIFAO_ExtINTSrc,
                             INTIFAO_INT_ACTIVE_STATE_RISING, ENABLE);
    /*Set standby mode as IDLE */
    CG_SetSTBYMode(CG_STBY_MODE_IDLE);
    __DSB();
```

Finally, enter IDLE mode by using __WFI_wait() instruction.

```
    __WFI_wait();
```

Eight __NOP() instruction are added after __WFI() instruction because the status of interrupt might be disable.

```
    void __WFI_wait()
    {
        __WFI();
        __NOP();
        __NOP();
```

```
        __NOP();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
        __NOP();
    }
```

7-3 uDMAC

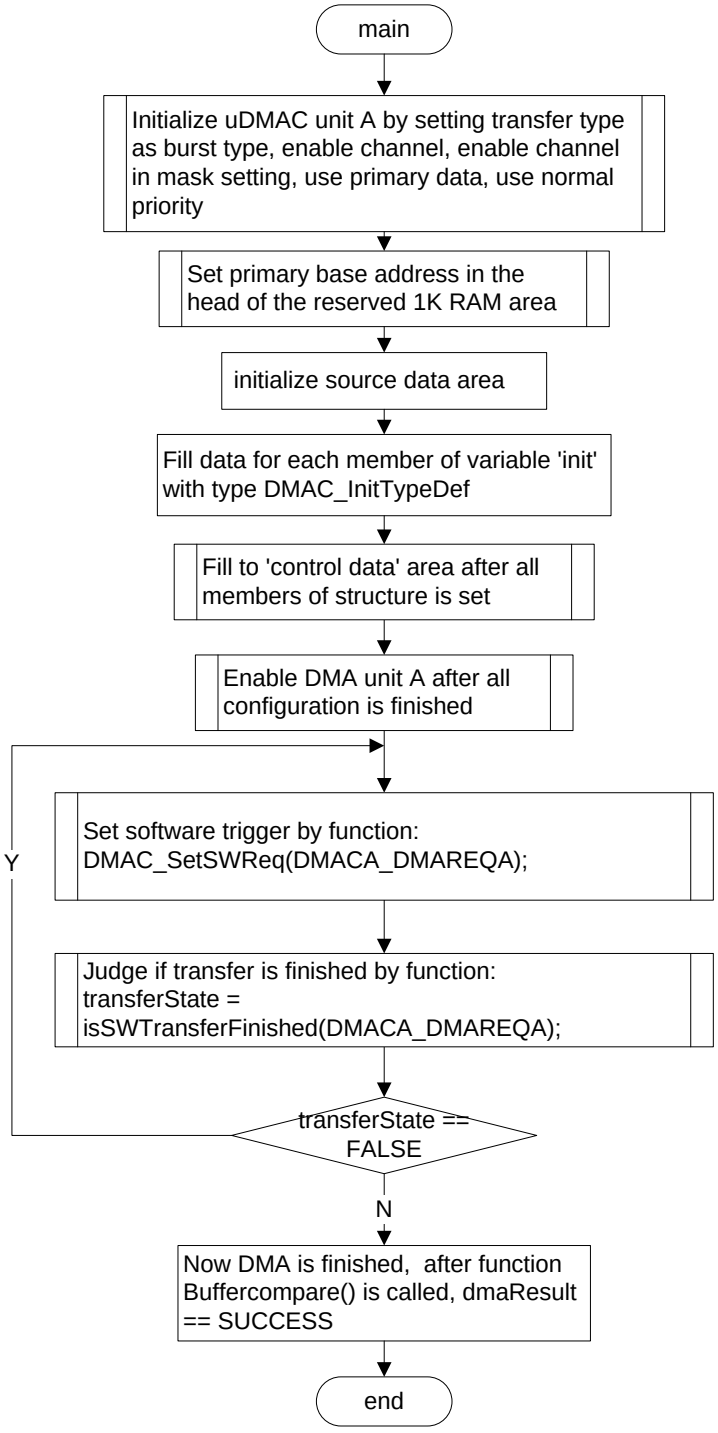
7-3-1 Example: DMA transfer from memory to memory

This is a simple example based on the TX00 Peripheral Driver (uDMAC).

The example includes:

1. Reserve 1K RAM area for uDMAC control data
2. uDMAC configuration and initialization
3. Start DMA transfer between memory by software trigger

- **Flowchart:**



- **Code and Explanation for the Example**

Since the uDMAC needs 1K bytes RAM area to save its configuration data, this area must be reserved in the application code, with bit0 to bit9 of its base address be 0, for example, 0x20000400 is OK but 0x20000300 cann't be used.

The example to reserve RAM area in IAR EWARM and Keil MDK(RealView) is showed as below:

```
#if defined (  __ICCARM__  )   /* IAR EWARM */
/* For TMPM066FWQG uDMAC_CFG are defined in file
TMPM066FWQG_Flash_For_uDMAC.icf*/
```

```
uint32_t uDMAC_Control_Data[256U] @ ".uDMAC_CFG " ;

#elif defined ( __CC_ARM   )       /* Keil MDK */
#define uDMAC_CFG         (0x20000400U)
uint32_t uDAMC_Control_Data[256U] __attribute__((at (uDMAC_CFG)));
#endif
```

For Keil MDK, the keyword '__at' is used and is enough but for IAR EWARM, the link configuration file (.lcf) file must be modified with content below. For more detail, please read file "TMPM066FWUG_Flash_For_uDMAC.icf".

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

    define region uDMAC_CFG_RAM    = mem:[from uDMAC_RAM_START to
uDMAC_RAM_END];

    place in    uDMAC_CFG_RAM     { readwrite section .uDMAC_CFG };
```

Set the clock supply.

```
        /* enable clock supply to DMAC */
        CG_SetFcPeriphA(CG_FC_PERIPH_DMAC, ENABLE);
```

After reserved the RAM, set transfer type as burst type, enable channel, enable channel in mask setting, use primary data, use normal priority.

```
        DMAC_SetTransferType(DMAC_UART0_TX, DMAC_BURST);
        DMAC_SetChannel(DMAC_UART0_TX, ENABLE);
        DMAC_SetMask(DMAC_UART0_TX, ENABLE);
        DMAC_SetPrimaryAlt(DMAC_UART0_TX, DMAC_PRIMARY);
        DMAC_SetChannelPriority(DMAC_UART0_TX,
                                DMAC_PRIOTIRY_NORMAL);
```

Then set primary base address in the head of the reserved 1K RAM area:

```
        DMAC_SetPrimaryBaseAddr((uint32_t)&uDMAC_Control_Data);
```

Initialize the source data area to be transfered

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

Now start setting the members of variable 'init' which is "DMAC_InitTypeDef" type.
Set the end address of source and destination

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

Select use BASIC or AUTOMATIC mode

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

Set other memebers

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

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

Fill to 'control data' area after all members of structure is set

```
DMAC_FillInitData(DMAC_UART0_TX, &init);
```

Enable DMA unit A after all configuration is finished

```
DMAC_Enable();
```

Set software trigger, and judge if transfer is finished.

```
do{
   /* because of "init.Mode = DMAC_BASIC" above, here need to trigger it
until transfer is finished, */
   /* if DMAC_AUTOMATIC is used, only need to trigger it once */
   DMAC_SetSWReq(DMAC_UART0_TX);

   transferState = isSWTransferFinished(DMAC_UART0_TX);

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

When transfer is finished, call function Buffercompare() to judge if dmaResult is SUCCESS.

```
dmaResult = ERROR;
dmaResult = Buffercompare( src, dst, TX_NUMBERS);
```

7-4 FLASH

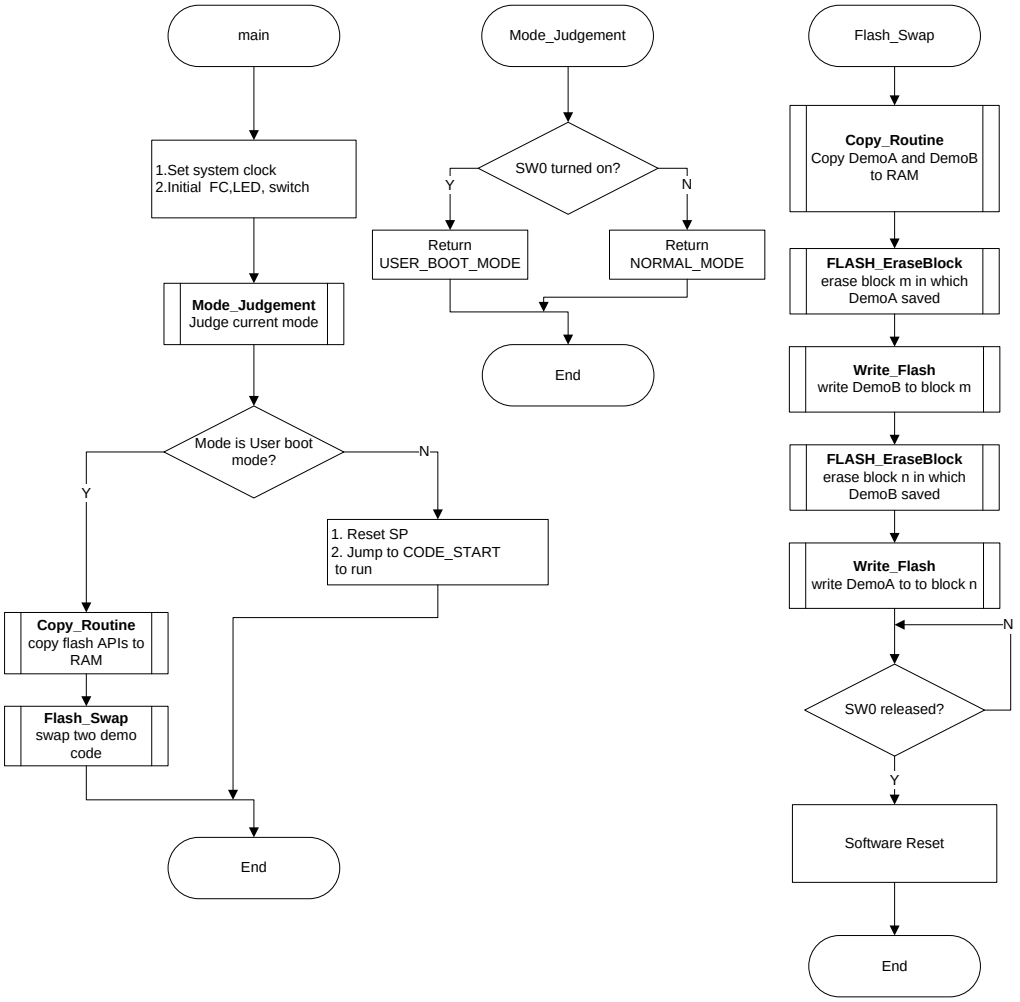
7-4-1 Example: Flash_UserBoot

This is a simple example based on the TX00 Peripheral Driver (FLASH, GPIO).

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 (GPIO) is used to judge current mode when reset .

```
    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 */
            c_stack = 0;
            __set_MSP(*c_stack);    /* reset SP */
#endif
            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 NVIC_SystemReset(). 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 (LED1 blinks, LED0 Always show).

```
    while (SW_Get(SW0) == 1U) {
    }
    /* software reset */
    NVIC_SystemReset();
```

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 16 bytes. 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 < PROGRAM_UNIT; i++) {
    *addr3 = *source;
    source++;
}
```

7-5 GPIO

This function configures GPIO to make LED and Switch work. Read Switch to control LED on and LED off.

7-5-1 Example: GPIO Data Read

This is a simple example based on the TX00 Peripheral Driver (GPIO).

The example includes:

1. GPIO initialization
2. Write data to GPIO
3. Read data from GPIO

- **Code and Explanation for the Example**

  At first, use GPIO_SetOutput(GPIO_Port GPIO_x, uint8_t Bit_x) to configure GPIO to LED, and GPIO_SetInput(GPIO_Port GPIO_x, uint8_t Bit_x) to configure GPIO to key. For example,

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

GPIO_SetInput(GPIO_PB,    GPIO_BIT_2 | GPIO_BIT_3);
```

In the for(; ;) process, run the LED demo: Read Switch to control LED on and LED off.

Read Switch by using GPIO_ReadDataBit(GPIO_Port GPIO_x, uint8_t Bit_x).

```
switch (sw) {
    case SW0:
        if (GPIO_ReadDataBit(GPIO_PB, GPIO_BIT_2) == 0U) {
            tmp = 1U;
        } else {
            /*Do nothing */
        }
        break;
    case SW1:
        if (GPIO_ReadDataBit(GPIO_PB, GPIO_BIT_3) == 0U) {
            tmp = 1U;
        } else {
            /*Do nothing */
        }
        break;
    }
```

Turn on LED by using GPIO_WriteData(GPIO_Port GPIO_x, uint8_t Data)

```
uint8_t tmp;
tmp = GPIO_ReadData(GPIO_PB);
tmp |= led & 0x0F;
GPIO_WriteData(GPIO_PB, tmp);
```

Turn off LED by using GPIO_WriteData(GPIO_Port GPIO_x, uint8_t Data)

```
uint8_t tmp;
tmp = GPIO_ReadData(GPIO_PB);
tmp &= (~led) & 0x0F;
GPIO_WriteData(GPIO_PB, tmp);
```

7-6 I2C

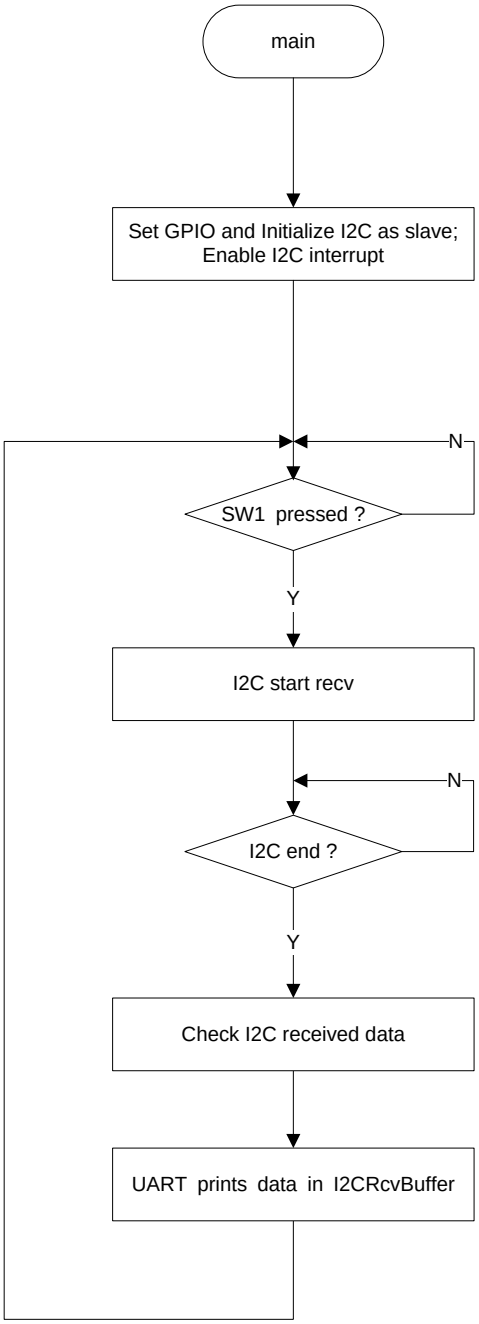
7-6-1 Example: I2C Slave

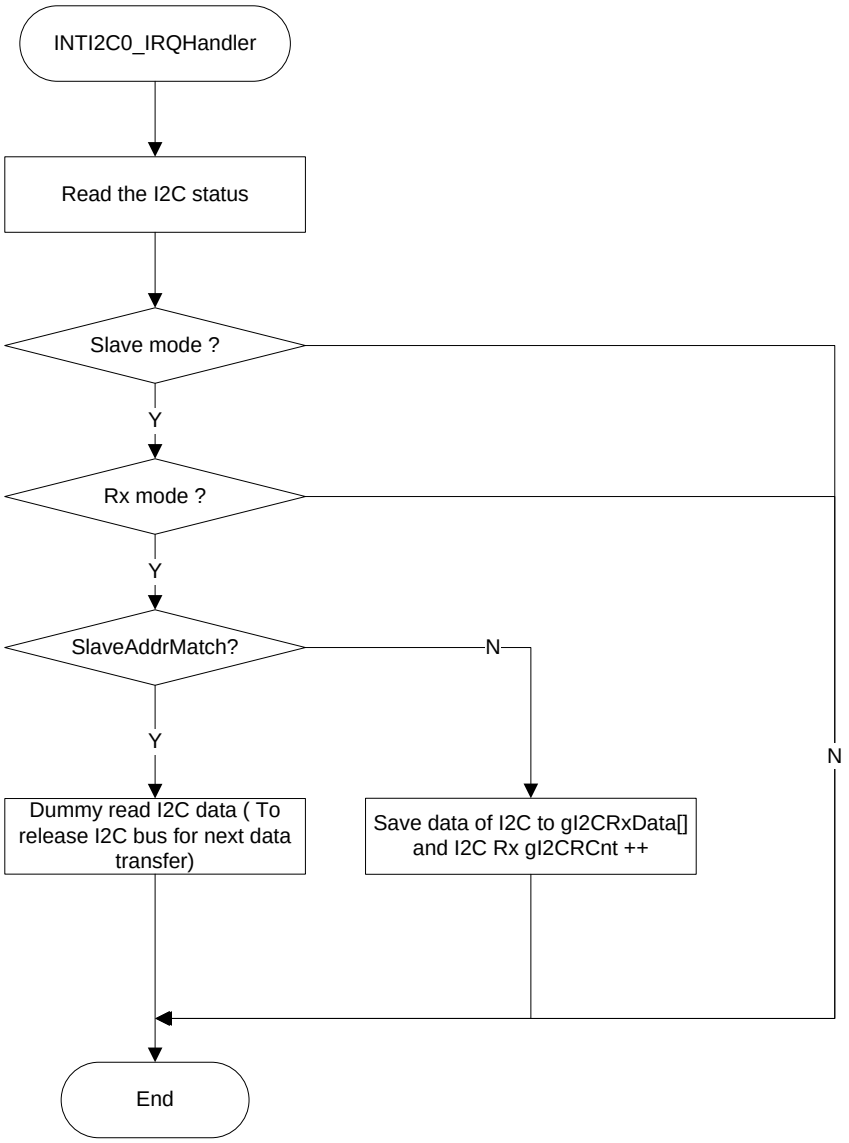
This is a simple example based on the TX00 Peripheral Driver (I2C, GPIO).

The example includes:

1. I2C configuration
2. I2C slave receive data process

- **Flowchart**


main
Set GPIO and Initialize I2C as slave;
Enable I2C interrupt
N
SW1 pressed ?
Y
I2C start recv
N
I2C end ?
Y
Check I2C received data
UART prints data in I2CRcvBuffer



- **Code and Explanation for the Example**

At first, set the clock supply.

```
/* enable clock supply to I2C0 */
CG_SetFcPeriphA(CG_FC_PERIPH_I2C0,ENABLE);
```

Next, configure GPIO for I2C mode.

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

Then enable, initialize and configure slave I2C channel and enable INTI2C0.

```
    myI2C.I2CSelfAddr = SLAVE_ADDR;
    myI2C.I2CDataLen = I2C_DATA_LEN_8;
    myI2C.I2CACKState = ENABLE;
    myI2C.I2CClkDiv = I2C_CLK_DIV_32;
    myI2C.PrescalerClkDiv = I2C_PRESCALER_DIV_12;
    I2C_SWReset(TSB_I2C0);
    I2C_Init(TSB_I2C0, &myI2C);
    NVIC_EnableIRQ(INTI2C0_IRQn);
    I2C_SetINTI2CReq(TSB_I2C0, ENABLE);
```

After the above setting, start I2C and wait for receive.
Set the I2C RX buffer.

```
    case MODE_I2C_INITIAL:
        for (gICnt = 0U; gICnt < 8U; gICnt++) {
            gI2CRxData[gICnt] = 0U;
        }
        gI2CMode = MODE_I2C_RCV;
        break;
```

The data receive is handled in INTI2C0.

In INTI2C0 function, I2C slave receive process is handled according to I2C bus state, I2C_GetReceiveData() is used to read received data in I2C buffer, I2C stop condition is controlled by master.

```
void INTI2C0_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_I2C_TypeDef *I2C;
    I2C_State I2C0_sr;

    I2Cy = TSB_I2C0;
    I2C0_sr = I2C_GetState(I2Cy);

    if (!I2C0_sr.Bit.MasterSlave) {        /* Slave mode */
        if (!I2C0_sr.Bit.TRx) { /* Rx Mode */
            if (I2C0_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = I2C_GetReceiveData();
                gI2CRCnt = 0U;
                I2C_SetSendData(I2Cy, tmp);
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = I2C_GetReceiveData();
                gI2CRxData[gI2CRCnt] = tmp;
                gI2CRCnt++;
```

```
                    I2C_SetSendData(I2Cy, tmp);
                }
            } else {                    /* Tx Mode */
                /* Do nothing */
            }
        } else {                        /* Master mode */
            /* Do nothing */
        }
}
```

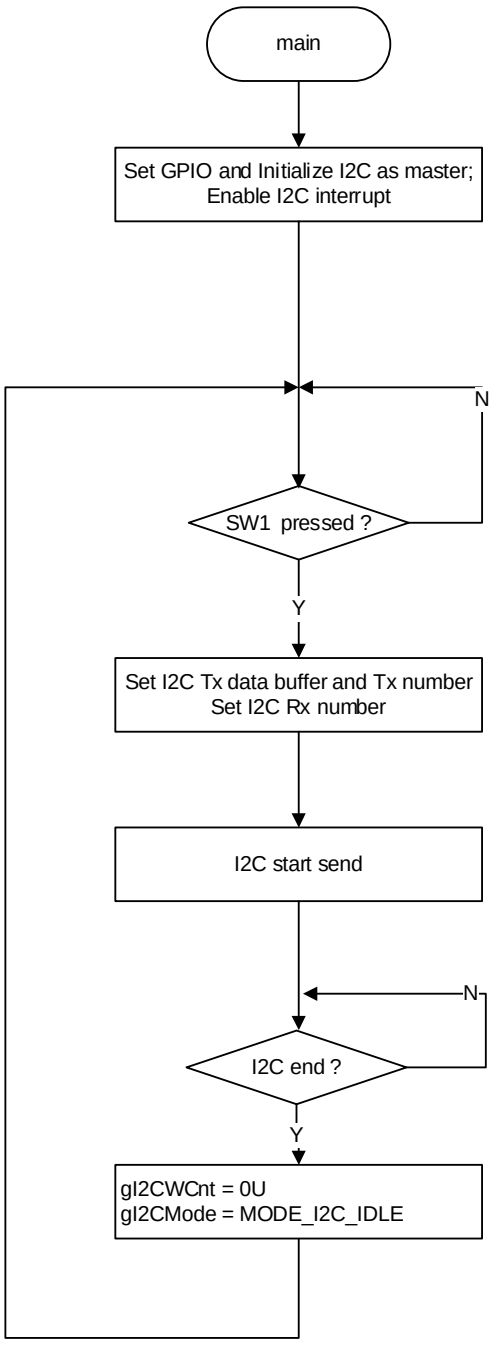
7-6-2 Example: I2C Master

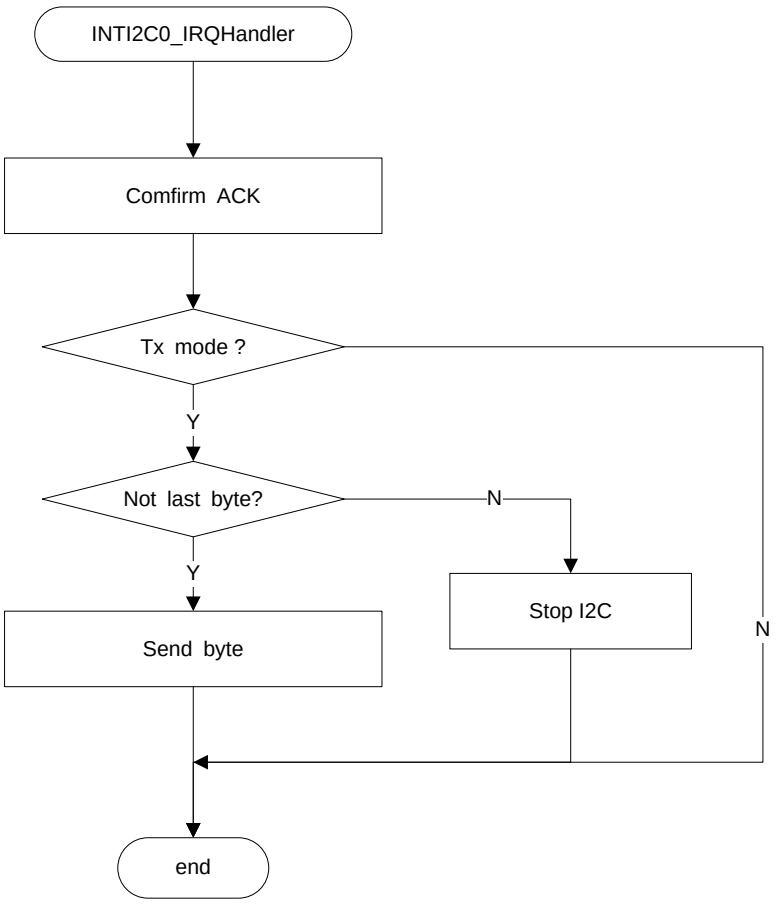
This is a simple example based on the TX00 Peripheral Driver (I2C, GPIO).

The example includes:

1 I2C configuration
2 I2C master send data process

- **Flowchart**


main
Set GPIO and Initialize I2C as master;
Enable I2C interrupt
N
SW1 pressed ?
Y
Set I2C Tx data buffer and Tx number
Set I2C Rx number
I2C start send
N
I2C end ?
Y
gI2CWCnt = 0U
gI2CMode = MODE_I2C_IDLE



- **Code and Explanation for the Example**

At first, set the clock supply.

```
/* enable clock supply to I2C0 */
CG_SetFcPeriphA(CG_FC_PERIPH_I2C0, ENABLE);
```

Next, configure GPIO for I2C mode.

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

Then enable, initialize and configure slave I2C channel and enable INTI2C0.

```
myI2C.I2CSelfAddr = SELF_ADDR;
```

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

After the above setting, start I2C send.
Set the I2C TX buffer and buffer length.

```
        /* Initialize Tx buffer and Tx length */
    case MODE_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;
        gI2CMode = MODE_I2C_START;
        break;
```

Check if the I2C bus is free or not, I2C_SetSendData() is used to set data “SLAVE_ADDR”.and direction is “I2C_SEND” to I2C data buffer; then I2C_GenerateStart() is used to to start I2C process.

```
    /* Check I2C bus state and start TRx */
    case MODE_I2C_START:
        i2c_state = I2C_GetState(TSB_I2C0);
        if (!i2c_state.Bit.BusState) {
            gI2CMode = MODE_I2C_TRX;
            I2C_SetSendData(TSB_I2C0, SLAVE_ADDR | I2C_SEND);
            I2C_GenerateStart(TSB_I2C0);
        } else {
            /* Do nothing */
        }
        break;
```

The data transfer is handled in INTI2C0.

In INTI2C0 function, I2C master send process is handled according to I2C bus state. During I2C master sending, I2C_SetSendData() is used to send next data, I2C_GenerateStop() is used to stop I2C when I2C process is finished.

```
void INTI2C_IRQHandler(void)
{
   TSB_I2C_TypeDef *I2Cx;
   I2C_I2CState I2C_sr;

   I2Cx = TSB_I2C0;
   I2C_sr = I2C_GetState(I2Cx);
   if (i2c_sr.Bit.MasterSlave) {            /* Master mode */
      if (i2c_sr.Bit.TRx) {      /* Tx mode */
         if (i2c_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further data. */
            I2C_GenerateStop(I2Cx);
         } else {                    /* LRB=0: the receiver requires further data. */
           if (gI2CWCnt <= gI2CTxDataLen) {
              I2C_SetSendData(I2CTx, gI2CTxData[gI2CWCnt]);    /* Send next data */
              gI2CWCnt++;
         } else {             /* I2C data send finished. */
              I2C_GenerateStop(I2Cx);    /* Stop I2C */
              gI2C_stop_flag = 1U;
           }
      }
     } else {                          /* Rx Mode */
         /* Do nothing */
     }
   } else {                          /* Slave mode */
            /* Do nothing */
      }
}
```

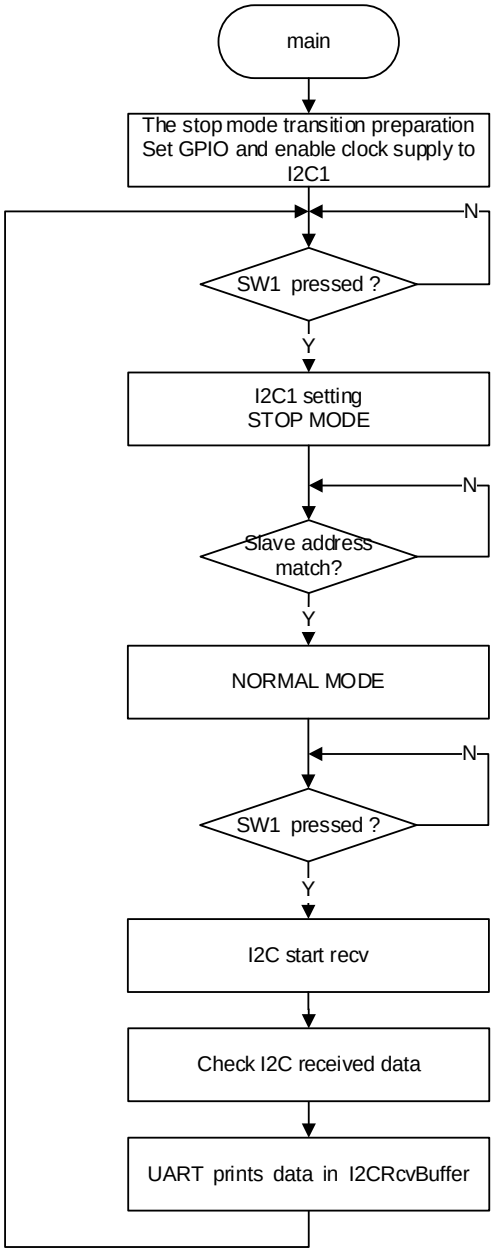
7-6-3 Example: I2CS wake up

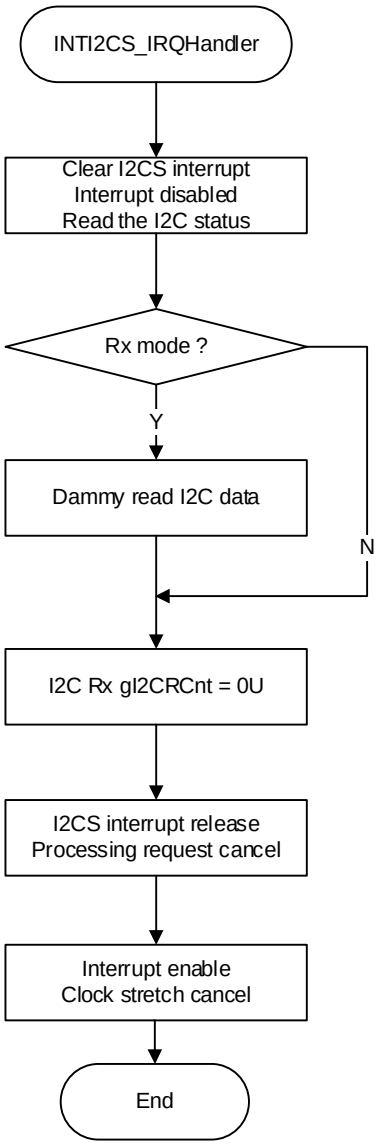
This is a simple example based on the TX00 Peripheral Driver (I2CS, I2C,GPIO).

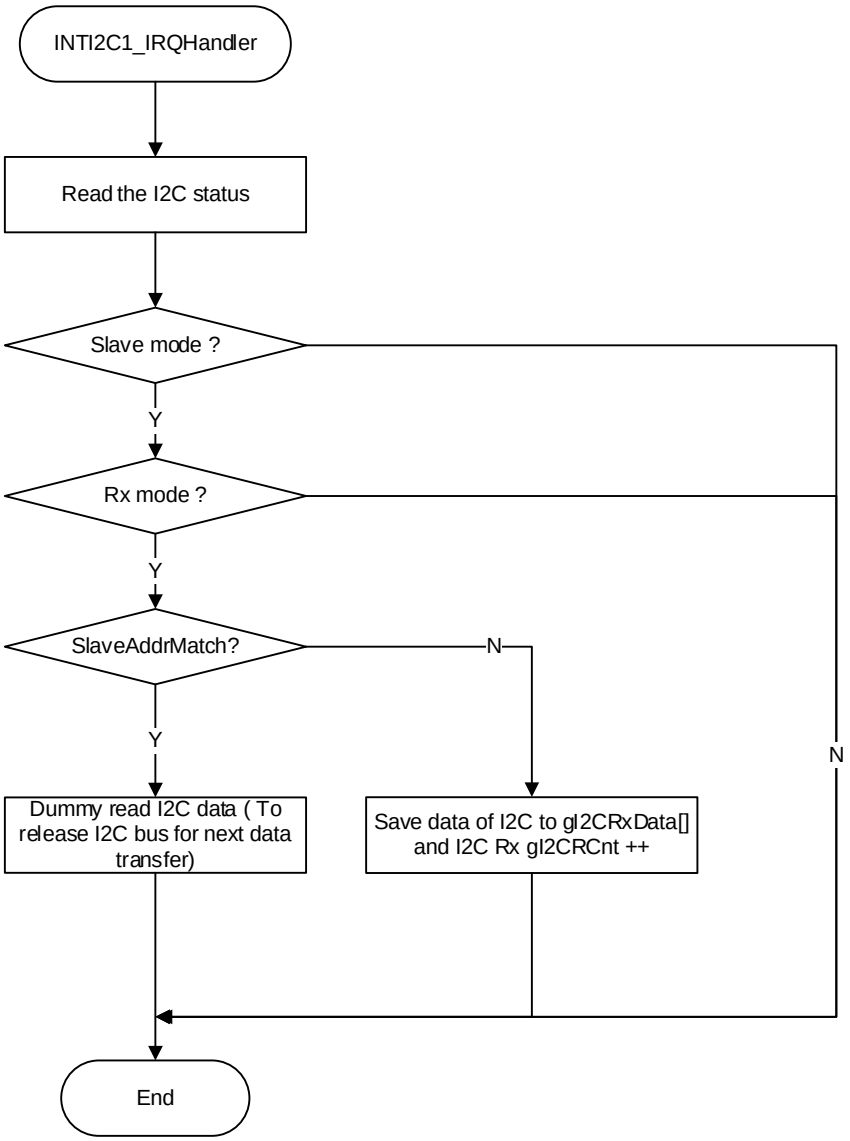
The example includes:

1. I2CS configuration
2. I2CS slave receive data process

- **Flowchart**


main
The stop mode transition preparation
Set GPIO and enable clock supply to I2C1
N
SW1 pressed ?
Y
I2C1 setting
STOP MODE
N
Slave address match?
Y
NORMAL MODE
N
SW1 pressed ?
Y
I2C start recv
Check I2C received data
UART prints data in I2CRcvBuffer


INTI2CS_IRQHandler
Clear I2CS interrupt
Interrupt disabled
Read the I2C status
Rx mode ?
Y
N
Dammy read I2C data
I2C Rx gI2CRCnt = 0U
I2CS interrupt release
Processing request cancel
Interrupt enable
Clock stretch cancel
End



- **Code and Explanation for the Example**

At first, set the SW, LED and configure UART.

```
SW_Init();
LED_Init();

UART_SWReset(UART_RETARGET);
hardware_init(UART_RETARGET);
```

Next, set the clock supply and configure GPIO for I2C mode.

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

```
    /* enable clock supply to I2C1 */
    CG_SetFcPeriphB(CG_FC_PERIPH_I2C1, ENABLE);
```

Set the I2C RX buffer.

```
  case MODE_I2C_INITIAL:
      for (gICnt = 0U; gICnt < 8U; gICnt++) {
          gI2CRxData[gICnt] = 0U;
      }
      gI2CMode = MODE_I2C_NORMAL;
      gI2CRCnt = 0U;
      common_uart_disp("S5: STOP MODE");
      common_uart_disp("¥r¥n¥r¥n");
      break;
```

Change the clock to be used in internal high-frequency, it will transition to the stop mode after the I2C configuration.

```
    case MODE_I2C_STOP:
        common_uart_disp("MASTER I2C START: NORMAL MODE");
        common_uart_disp("¥r¥n¥r¥n");
        gI2CMode = MODE_I2C_RCV_PRE;
        LED_Off(LED_ALL);
        do {
            regval = UART_RETARGET->MOD2;
            regval &= TX_MASK;
        } while(regval != TX_END);
        UART_Disable(UART_RETARGET);

        I2CS_StopStart();
        break;
```

In INTI2CS function, the INTI2CS interrupt request is cleared, and then release the clock stretch.

```
void INTI2CS_IRQHandler(void)
{
    TSB_I2C_TypeDef *I2Cy;
    I2C_State I2C1_sr;

    I2Cy = TSB_I2C1;

    I2CS_INTRelease(ENABLE);
    I2CS_INTRelease(DISABLE);
    __disable_irq();
    I2C1_sr = I2C_GetState(I2Cy);

    if (!I2C1_sr.Bit.TRx) { /* Rx Mode */
        I2C_GetReceiveData(I2Cy);
```

```
    }
    gI2CRCnt = 0U;

    I2CS_INTRelease(ENABLE);
    TSB_INTIFAO->STOP2INT_038 = I2CS_FLAG_ALLCLEAR;
    TSB_INTIFAO->STOP2INT_038 |= I2CS_MODEEN_SET;
    NVIC_ClearPendingIRQ(INTI2CS_IRQn);
    I2C_ClearINTReq(I2Cy);
    __enable_irq();
    I2CS_Reset(ENABLE);
}
```

In INTI2C1 function, I2C slave receive process is handled according to I2C bus state, I2C_GetReceiveData() is used to read received data in I2C buffer, I2C stop condition is controlled by master.

```
void INTI2C1_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_I2C_TypeDef *I2Cy;
    I2C_State I2C1_sr;

    I2Cy = TSB_I2C1;
    I2C1_sr = I2C_GetState(I2Cy);

    if (!I2C1_sr.Bit.MasterSlave) {        /* Slave mode */
        if (!I2C1_sr.Bit.TRx) { /* Rx Mode */
            if (I2C1_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = I2C_GetReceiveData(I2Cy);
                gI2CRCnt = 0U;
                I2C_SetSendData(I2Cy, tmp);
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = I2C_GetReceiveData(I2Cy);
                gI2CRxData[gI2CRCnt] = tmp;
                gI2CRCnt++;
                I2C_SetSendData(I2Cy, tmp);
            }
        } else {                    /* Tx Mode */
            /* Do nothing */
        }
    } else {                        /* Master mode */
        /* Do nothing */
    }
}
```

7-7 LVD

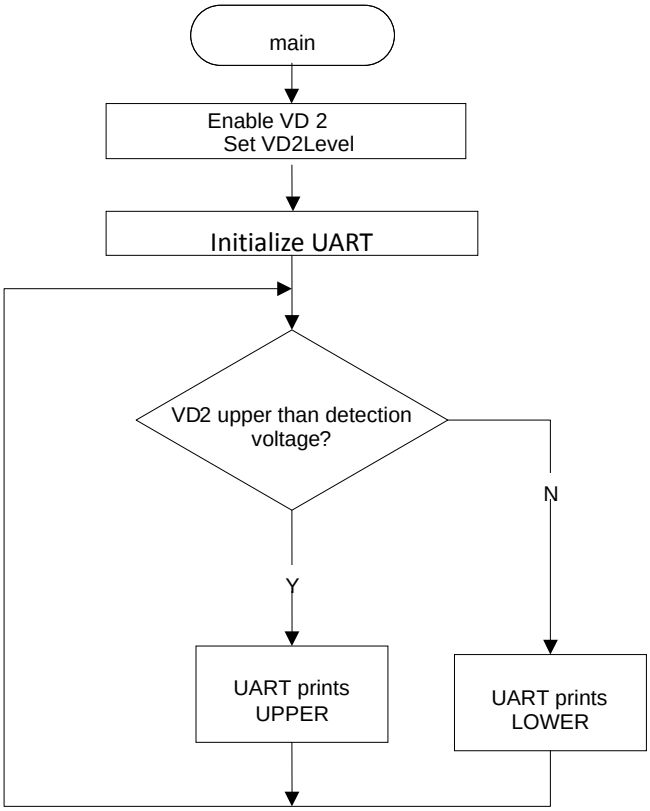
7-7-1 Example: LVD

This is a simple example based on the TX00 Peripheral Driver (LVD, GPIO).

The example includes:

1. LVD configuration
2. LVD status monitoring

- **Flowchart**



- **Code and Explanation for the Example**

Enable voltage detection 2; setting mode and detection level.

```
LVD_EnableVD2();
LVD_SetVD2Level(LVD_VDLVL2_290);
```

Initialize the system next to. Disabe WDT explicitly to prevent confusion.

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

Then a usual while(1) loop.

Check VD2 status, if VD2 is upper than the detection voltage, UART displays “UPPER”. When VD2 is lower than the detection voltage, UART displays “LOWER”.

```
        while (1) {
            if (LVD_GetVD2Status() == LVD_VD_UPPER) {
```

```
                common_uart_disp("UPPER¥n");
            } else {
                common_uart_disp("LOWER¥n");
            }
        }
```

7-8 TMR16A

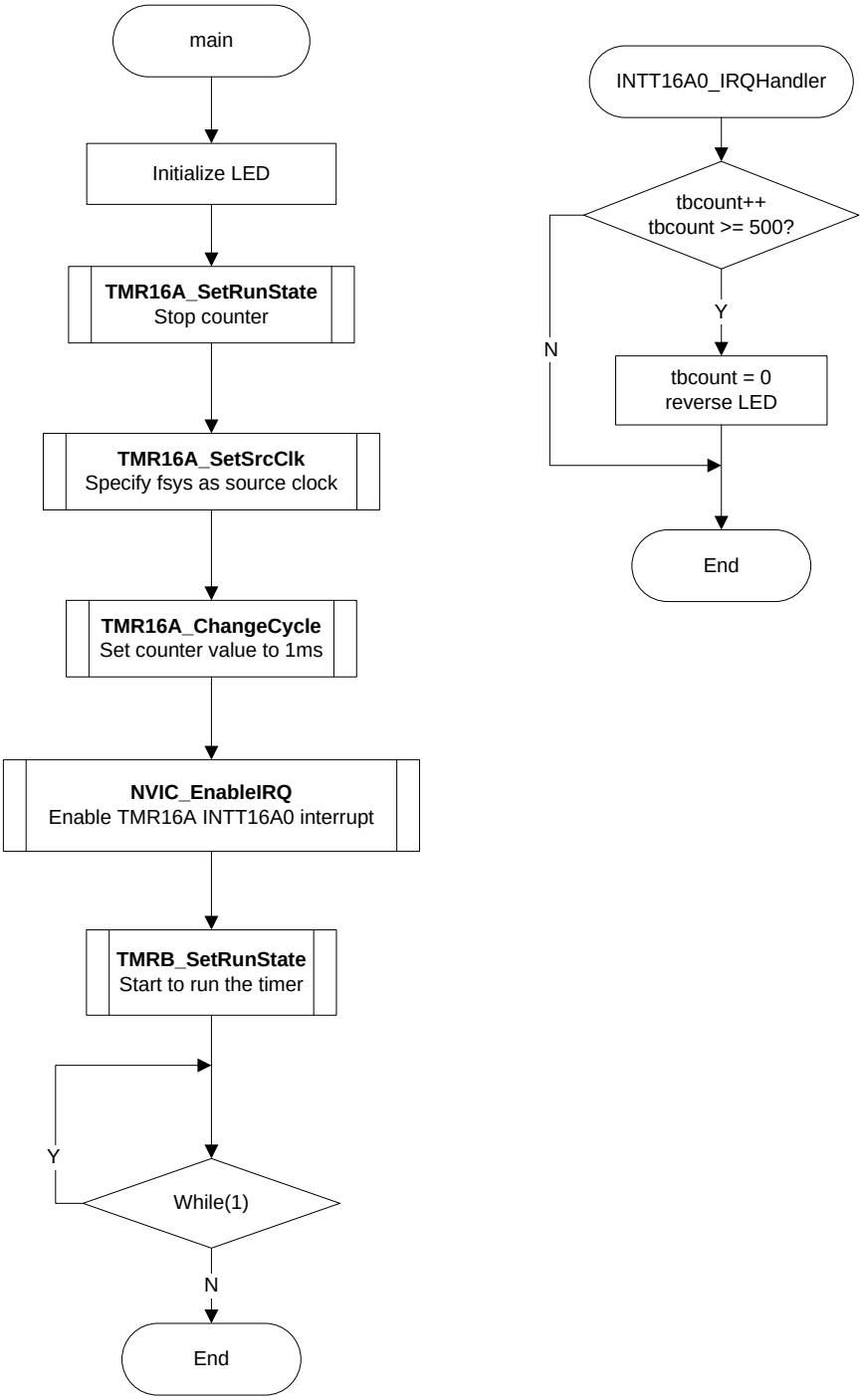
7-8-1 Example: General Timer

This is a simple example based on the TX00 Peripheral Driver (TMR16A, GPIO).

The example includes:

1. TMR16A setting
2. General Timer for 1ms

- **Flowchart**



- **Code and Explanation for the Example**

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

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

Set the clock supply.

```
    /* enable clock supply to TMR16A */
CG_SetFcPeriphA(CG_FC_PERIPH_TMR16A, ENABLE);
```

First stops the counter. This demo will set cycle to 1ms, macro TMR16A_1MS equals 0x5DC0U, because ftmrb = fphiT0 = fsys = fc = 24MHz, Ttmrb = 1/24us, 1ms*24 us = 24000 = 0x5DC0 (Please see CG part for more detail information

about the clock setting)

Then run the counter.

```
    TMR16A_SetRunState(TSB_T16A0, TMR16A_STOP);   /* Counter stops*/
    TMR16A_SetSrcClk(TSB_T16A0, TMR16A_SYSCK);    /* Set source clock to system clock */
    TMR16A_ChangeCycle(TSB_T16A0, TMR16A_1MS);    /* Set counter value to 1ms*/
    NVIC_EnableIRQ(INTT16A0_IRQn);
    TMR16A_SetRunState(TSB_T16A0, TMR16A_RUN);
```

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(LED_ALL);
        } else {
            LedOn(LED_ALL);
        }
    } else {
        /* do nothing */
    }
```

7-9 TMRD

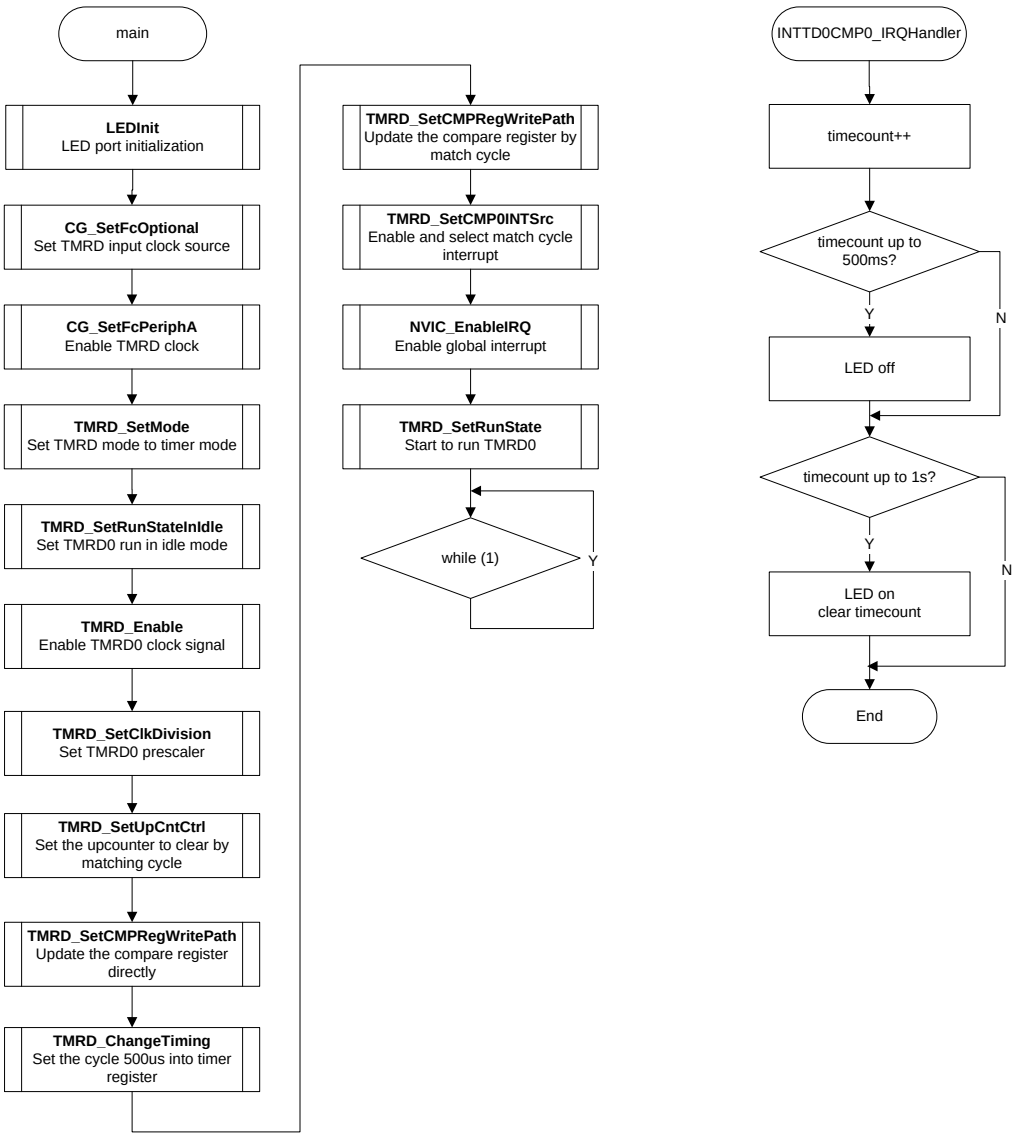
7-9-1 Example: Interval Timer

This is a simple example based on the TX00 Peripheral Driver (TMRD, CG).

The example includes:

1. TMRD interval timer mode
2. TMRD initialization and operation process
3. TMRD interrupt handling

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize the LED port on evaluation board, and turn on the LED.

```
    LED_Init();
```

Configure TMRD clock by using API CG_SetFtmrdSrc() and enable the clock source by using CG_SetTMRDClk().

```
    /* TMRD CG setting */
  CG_SetFcOptional(CG_FPLL_PERIPH_TMRD,ENABLE);     /* clock is set to Fpll/4 */
  CG_SetFcPeriphA(CG_FC_PERIPH_TMRD,ENABLE);          /* enable the TMRDACLK */
```

Set TMRD mode by using TMRD_SetMode(). In this demo, both TMR0 and TMR1 are set to timer mode.

```
    /* TMRD mode setting */
```

```
TMRD_SetMode(TMRD_MODE_BOTH_TMR);
```

Set TMRD0 to run in idle mode by using API TMRD_SetRunStateInIdle().

```
/* TMRD IDLE mode */
TMRD_SetRunStateInIdle(TMRD_UNIT_CH_0, TMRD_RUN);
```

Then enable TMRD0 clock signal and set the prescaler by using API TMRD_Enable() and TMRD_SetClkDivision().

```
/* TMRD enable clock signal */
TMRD_Enable(TMRD_UNIT_CH_0);

/* Set the TMRD prescaler */
TMRD_SetClkDivision(TMRD_UNIT_CH_0, TMRD_CLK_DIV_1);
```

Set the TMRD0 up-counter to clear when match the cycle (auto clear).

```
/* Set the upcounter clear mode */
TMRD_SetUpCntCtrl(TMRD_UNIT_CH_0, TMRD_AUTO_CLEAR);
```

After using API TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_DIRECT) to set to update compare register directly, the same data is written to the corresponding compare register when data is written to the timer register by using API TMRD_ChangeTiming().

The calculation method of macro TIME_500US is:
fosc = 12MHz(external high-speed oscillator)
fpll = 8 * fosc = 96MHz(in this demo, PLL multiplying value is set to 8 multiplying)
ftmrd = fpll / 4 = 50MHz(in this demo, ftmrd = fpll/ 4, please see CG_SetFtmrdSrc())
So the value of TIME_500US is 500us*ftmrd = 48000 = 0x_BB80

```
/* Update the compare register directly */
TMRD_SetCMPRegWritePath(
                        TMRD_UNIT_CH_0,
                        TMRD_CMP_WRITE_DIRECT);

/* Set the value to timer register */
TMRD_ChangeTiming(TMRD_TIMING_TD0_CYCLE,
                  TIME_500US);                    /* 500us */
```

Next, using API TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_INDIRECT) to enable writing data through the timer register to the compare register when up-counter matching the cycle time.

```
/* Indirct update the compare register */
TMRD_SetCMPRegWritePath(
                        TMRD_UNIT_CH_0,
                        TMRD_CMP_WRITE_INDIRECT);
```

Then choose the compare 0 interrupt source to match cycle by using API TMRD_SetCMP0INTSrc(), and enable the compare 0 interrupt by using NVIC_EnableIRQ().

```
    /* Enable TMRD intterrupt */
    TMRD_SetCMP0INTSrc(TMRD_UNIT_CH_0, TMRD_INT_MATCH_CYCLE);

    NVIC_EnableIRQ(INTTMRD_IRQn);
```

In the end, start the TMRD0 to run by using TMRD_SetRunState() to run the counter and enter into a dead loop.

```
    /* Start to run TMRD */
    TMRD_SetRunState(TMRD_UNIT_CH_0, TMRD_RUN);
```

When 500us is up, the compare 0 interrupt occurs. In IRQ INTTDA0CMP0_IRQHandler(), use a variable *timecount* to accumulate. If 500ms is up, turn off the LED. And if 1s is up, turn on the LED again and clear the *timecount* to accumulate from the start.

```
void INTTMRD_IRQHandler(void)
{
    static uint16_t timecount = 0U;

    INTIFSD_ClearINTReq(INTIFSD_INT_SRC_TMRD_00);
    INTIFSD_ClearINTReq(INTIFSD_INT_SRC_TMRD_01);
    INTIFSD_ClearINTReq(INTIFSD_INT_SRC_TMRD_02);
    INTIFSD_ClearINTReq(INTIFSD_INT_SRC_TMRD_03);
    INTIFSD_ClearINTReq(INTIFSD_INT_SRC_TMRD_04);

    timecount++;
    if (timecount == 1000U) {     /* 500ms */
        LED_Off(LED_ALL);
    } else if (timecount == 2000U) {       /* 1s */
        timecount = 0U;
        LED_On(LED_ALL);
    } else {
        /* Do nothing */
    }
}
```

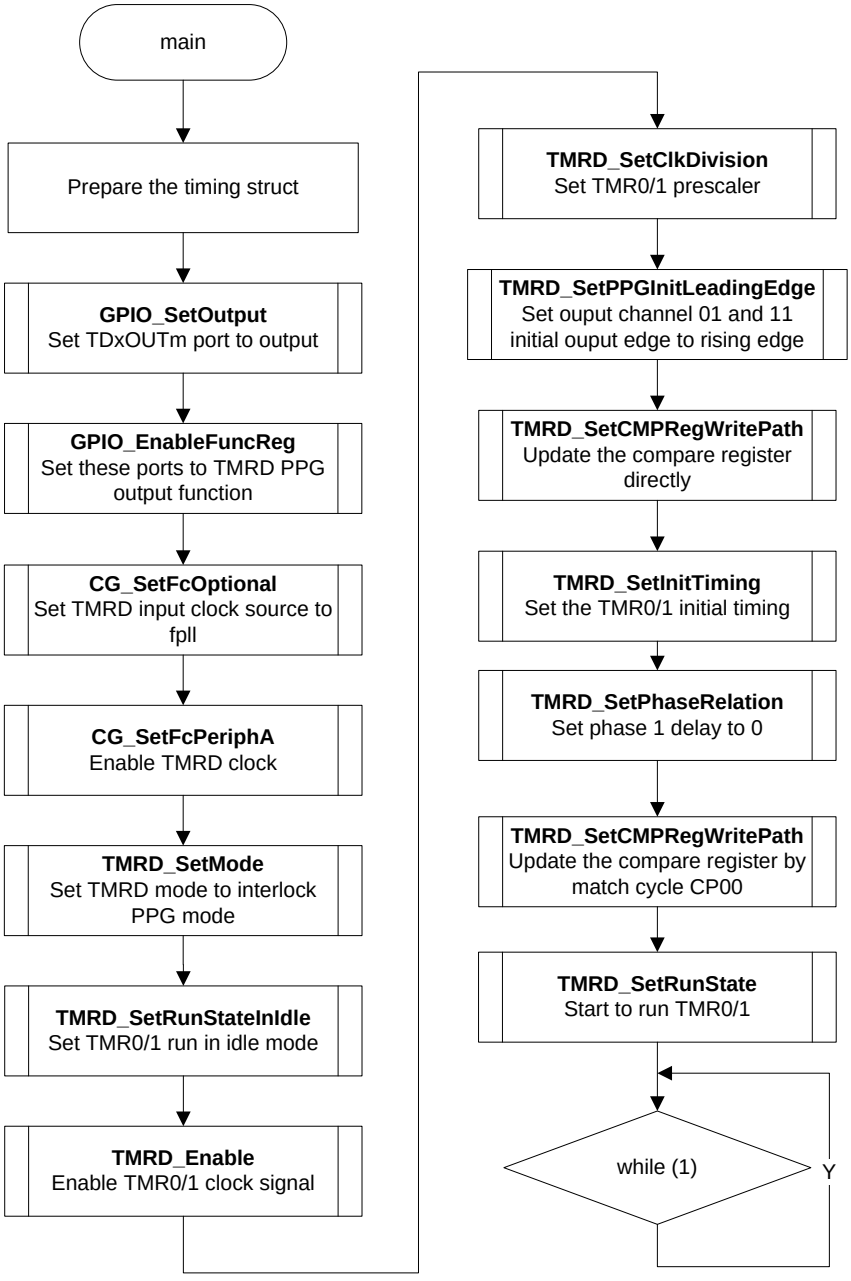
7-9-2 Example: PPG Output

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

The example includes:

1. TMRD interlock PPG mode
2. TMRD initialization and operation process
3. TMRD PPG mode parameters calculation method

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize the TMRD timing parameters structure. The calculation method is:

Cycle is 500us:

fosc = 12MHz(external high-speed oscillator)

fpll = 8 * fosc = 96MHz(in this demo, PLL multiplying value is set to 8 multiplying)
ftmrd = fpll / 4 = 50MHz(in this demo, ftmrd = fpll / 4, please see CG_SetFtmrdSrc())
So the value is 500us*ftmrd = 48000 = 0x_BB80

Duty is 50%:
Set LeadingTiming0 to 0x0000, then TrailingTiming0 is Cycle/2 = 0x5DC0 (250us)
LeadingTiming1 and TrailingTiming1 are not used, so keep their value to 0x0000.

Phase shift is 120 degree:
The phase shift can be calculated by using the formula:

θ = 360 degree*(PhaseShiftTiming/(Cycle+1))

so in this demo, PhaseShiftTiming should be set to Cycle/3 = 0x208DU (167us)

```
#define TIME_CYCLE                ((uint16_t)0xBB80U)      /* 500us */
#define TIME_DUTY                 ((uint16_t)0x5DC0U)      /* 250us */
#define TIME_PHASE_SHIFT          ((uint16_t)0x3E80U)      /* 500/3= 167us */

TMRD_TimingTypeDef timestruct = { 0U };
timestruct.Cycle = TIME_CYCLE;
timestruct.TrailingTiming1 = TIME_DUTY;
timestruct.PhaseShiftTiming = TIME_PHASE_SHIFT;
```

Configure the TDAxOUTm port to TMRD PPG output. In this demo, these ports are port A 0,1,2,3.

```
    /*TDA0OUT1, TDA1OUT1 port setting */
GPIO_SetOutput(GPIO_PD, GPIO_BIT_1 | GPIO_BIT_3);
GPIO_EnableFuncReg(GPIO_PD,GPIO_FUNC_REG_2,GPIO_BIT_1| GPIO_BIT_3);
```

Configure TMRD clock by using API CG_SetFcOptional () and enable the clock source by using CG_SetFcPeriphA ().

```
    /* TMRD CG setting */
CG_SetFcOptional(CG_FPLL_PERIPH_TMRD,ENABLE);     /* clock is set to fpll/4 */
CG_SetFcPeriphA(CG_FC_PERIPH_TMRD,ENABLE);       /* enable the TMRDACLK
*/
```

Set TMRD mode by using TMRD_SetMode(). In this demo, use interlock PPG mode.

```
    /* TMRD mode setting */
    TMRD_SetMode(TMRD_MODE_INTERLOCK_PPG_3CH);
```

Set TMRD0A/1A to run in idle mode by using API TMRD_SetRunStateInIdle().

```
    /* TMRD IDLE mode */
TMRD_SetRunStateInIdle(TMRD_UNIT_CH_0, TMRD_RUN);
TMRD_SetRunStateInIdle(TMRD_UNIT_CH_1, TMRD_RUN);
```

Then enable TMRD clock signal and set the prescaler by using API TMRD_Enable() and TMRD_SetClkDivision(). Note that, in interlock PPG mode, the prescaler of TMRD1 is the same as the value setting of TMRD0. So only setting TMRD0 prescaler is enough.

```
    /* TMRD enable clock signal */
    TMRD_Enable(TMRD_UNIT_CH_0);
    TMRD_Enable(TMRD_UNIT_CH_1);
    /* Set the TMRD prescaler */
    TMRD_SetClkDivision(TMRD_UNIT_CH_0, TMRD_CLK_DIV_1);
```

Set the PPG output initial wave edge, both channel a0 and b0 are set to rising edge by using API TMRD_SetPPGInitLeadingEdge().

```
/* Set the PPG output edge */
TMRD_SetPPGInitLeadingEdge( TMRD_PPG_CHANNEL_01,
                                                TMRD_WAVE_EDGE_RISING);
TMRD_SetPPGInitLeadingEdge( TMRD_PPG_CHANNEL_11,
                                                TMRD_WAVE_EDGE_RISING);
```

After using API TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_DIRECT) to set to update compare register directly, the same data is written to the corresponding compare register when data is written to the timer register by using API TMRD_SetInitTiming ().The timing structure has been prepared at the beginning of this chapter.

```
/* Direct update the compare register */
TMRD_SetCMPRegWritePath(TMRD_UNIT_CH_0,
                                                TMRD_CMP_WRITE_DIRECT);
TMRD_SetCMPRegWritePath(TMRD_UNIT_CH_1,
                                                TMRD_CMP_WRITE_DIRECT);

/* Set the value to timer register */
TMRD_SetInitTiming(TMRD_UNIT_CH_0, &timestruct);
TMRD_SetInitTiming(TMRD_UNIT_CH_1, &timestruct);
```

Then setting the phase relation between phase A and B by using API TMRD_SetPhaseRelation(), and in this demo, phase B is delay to phase A.

```
/* Set the relation of phase A and B */
TMRD_SetPhaseRelation(TMRD_PHASE_DELAY_OR_SAME);
```

Next, using API TMRD_SetCMPRegWritePath(…, TMRD_CMP_WRITE_INDIRECT) to enable writing data through the timer register to the compare register when up-counter matching the cycle time.

```
/* Indirct update the compare register */
TMRD_SetCMPRegWritePath(TMRD_UNIT_CH_0,
                                                TMRD_CMP_WRITE_INDIRECT);
TMRD_SetCMPRegWritePath(TMRD_UNIT_CH_1,
                                                TMRD_CMP_WRITE_INDIRECT);
```

In the end, start the TMRD0/1 to run by using TMRD_SetRunState() to run the TMRD0 and enter to a dead loop. In interlock PPG mode, TMRD1 will start to operation in tandem with COUNTER0 of TMRD0A, setting TMRD1 to run by TMRD_SetRunState() becomes invalid, so there is no need to set TMRD1 to run.

```
    /* Start to run TMRD */
    TMRD_SetRunState(TMRD_UNIT_CH_0, TMRD_RUN);
```

7-10 TMRB

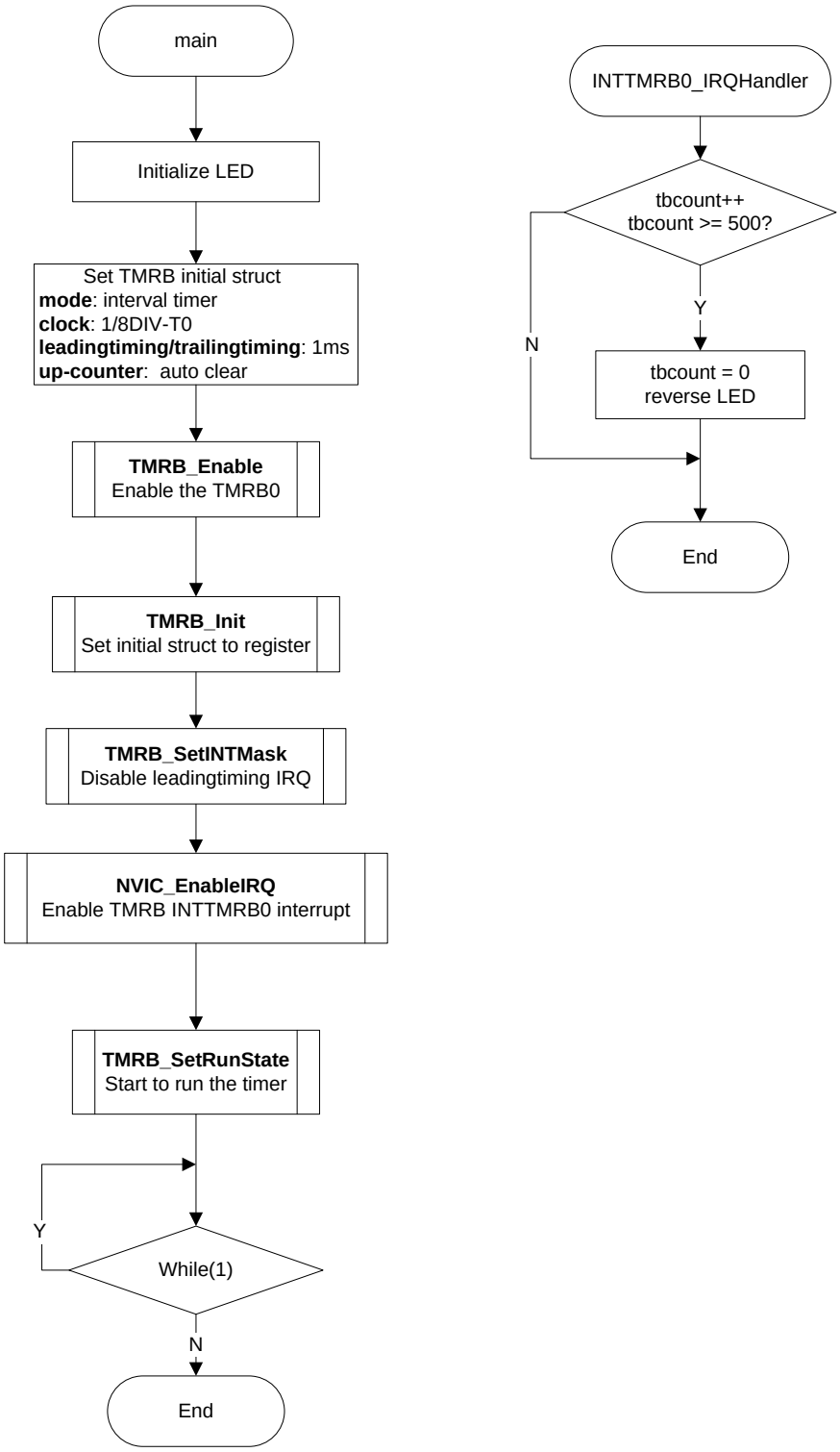
7-10-1 Example: General Timer

This is a simple example based on the TX00 Peripheral Driver (TMRB, GPIO).

The example includes:

1. TMRB0 initialization
2. General Timer for 1ms

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize LED channel on Eval board and turn off LED.

```
TMRB_InitTypeDef m_tmrb;

LED_Init();                                    /* LED initialize */
LED_Off(LED_ALL);                              /* Turn off LED_ALL */
```

Set the clock supply.

```
        /* enable clock supply to TB0 */
    CG_SetFcPeriphA(CG_FC_PERIPH_TMRB0_3, ENABLE);
```

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 0x0BB8, because fphiT0 = fsys = fc = fosc = 24MHz, ftmrb = 1/8 fphiT0 = 3MHz, Ttmrb = 0.333…us, 1ms/0.333…us = 3000 = 0x0BB8 (Please see CG part for more detail information about the clock setting)

```
    CG_SetFcPeriphA(CG_FC_PERIPH_TMRB0_3, ENABLE);

    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 INTTMRB0 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 */
    TMRB_SetINTMask(TSB_TB0,
                    TMRB_MASK_MATCH_LEADINGTIMING_INT);
    NVIC_EnableIRQ(INTTMRB0_IRQn);          /* enable INTTMRB0 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(LED_ALL);
        } else {
            LED_On(LED_ALL);
        }
    } else {
        /* Do nothing */
    }
```

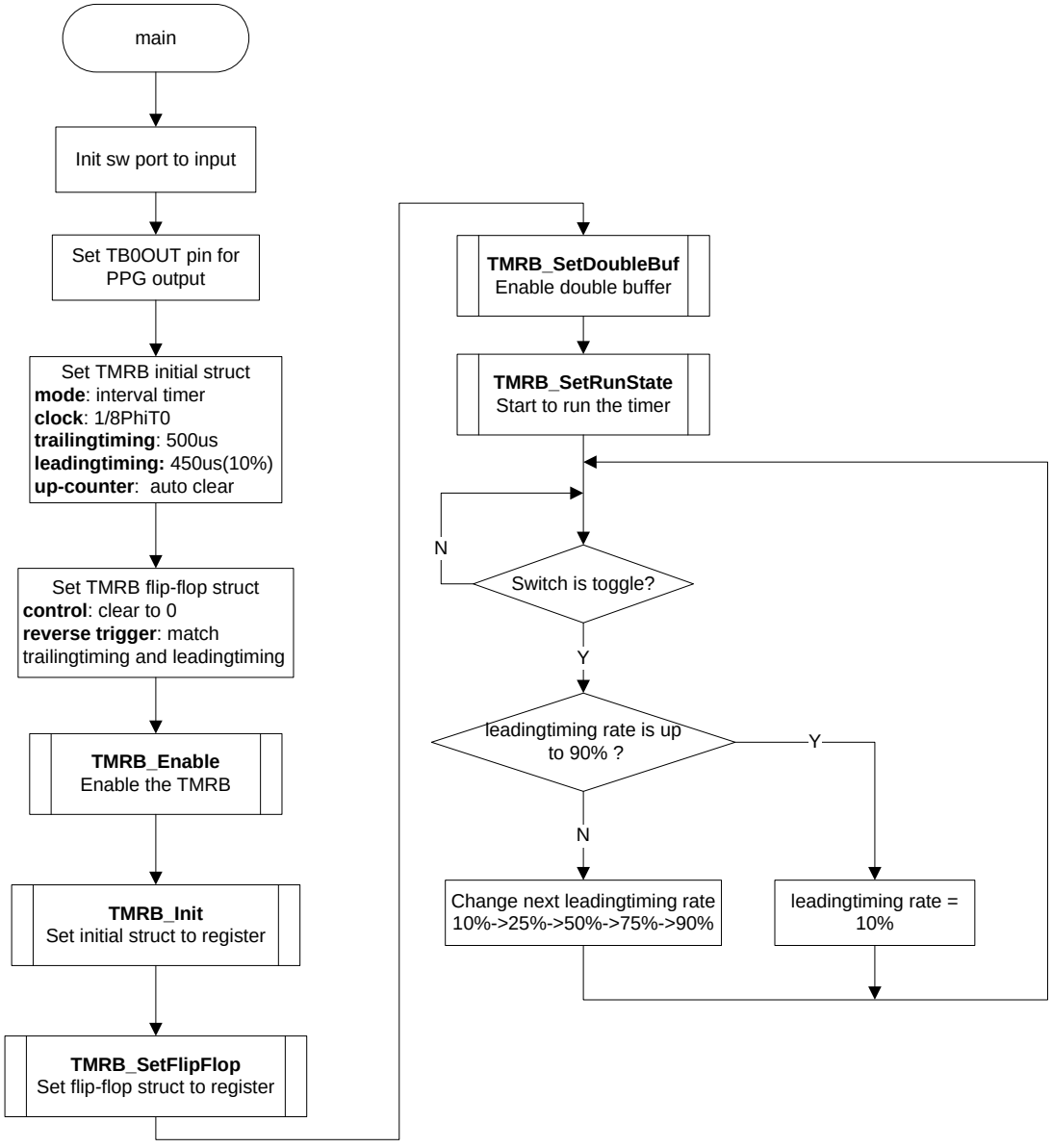
7-10-2 Example: PPG Output

This is a simple example based on the TX00 Peripheral Driver (TMRB, GPIO).

The example includes:

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

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize sw port; set PD1 as TB0OUT for PPG output.

```
SW_Init();                          /* Init sw port to input */
/* enable clock supply to TB0 */
```

```
    CG_SetFcPeriphA(G_FC_PERIPH_TMRB0_3, ENABLE);
    /* Set PD1 as TB0OUT for PPG output */
    GPIO_SetOutput (GPIO_PD, GPIO_BIT_1);
    GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_1, GPIO_BIT_1);
```

Prepare TMRB initialization structure, and then fill TMRB mode, clock, up-counter clear method, trailingtiming and leadingtiming into it. This demo will set trailingtiming to 500us, macro TMRB0TIME equals 0x05DCU, because fphiT0 = fsys = fc = fosc = 24MHz, ftmrb = 1/8 fphiT0 = 3MHz, Ttmrb = 0.333…us, 500us/0.333…us = 1500 = 0x5DCU (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.UpCntCtrl = TMRB_AUTO_CLEAR;         /* up-counter auto clear */
    m_tmrb.TrailingTiming = TMRB0TIME;          /* T = 500us */
    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_TB0);
    TMRB_Init(TSB_TB0, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB0, &PPGFFInital);
    TMRB_SetDoubleBuf(TSB_TB0, ENABLE);          /* enable double buffer */
    TMRB_SetRunState(TSB_TB0, TMRB_RUN);
```

Wait switch change status, and at the same time.

```
    keyvalue = SW_Get(SW1);
    if (keyvalue == 1U) {
        delay(0xFFFFU);         /* noise cancel */
        keyvalue = SW_Get(SW1);
        if (keyvalue == 0U) {
            changestatus = 1U;
        } else {
                /* Do nothing */
        }
```

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

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

```
if (changestatus == 1U) {
        Rate++;                 /* change leadingtiming rate */
        if (Rate >= LEADINGTIMINGMAX) {
            Rate = LEADINGTIMINGINIT;
        } else {
            /* Do nothing */
        }
        TMRB_ChangeLeadingTiming(TSB_TB0, LeadingTiming[Rate]);
        changestatus = 0U;
    } else {
        /* Do nothing */
    }
```

The calculation method of leadingtiming value:
TrailingTiming = 500us, ftmrb = 1/8 fphiT0 = 3MHz, Ttmrb = 0.333…us (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 = 0x546U

LeadingTiming = 25%: high-level time is 500*25% = 125us, low-level time is 500-125 = 375us, counter value = 375us/Ttmrb = 0x465U

LeadingTiming = 50%: high-level time is 500*50% = 250us, low-level time is 500-250 = 250us, counter value = 250us/Ttmrb = 0x2EEU

LeadingTiming = 75%: high-level time is 500*75% = 375us, low-level time is 500-375 = 125us, counter value = 125us/Ttmrb = 0x177U

LeadingTiming = 90%: high-level time is 500*90% = 450us, low-level time is 500-450 = 50us, counter value = 50us/Ttmrb = 0x96U

That is the calculation method of leadingtiming array

```
uint32_t LeadingTiming[5] = { 0x546U, 0x465U, 0x2EEU, 0x177U, 0x96U };
/* leadingtiming: 10%, 25%, 50%, 75%, 90% */
```

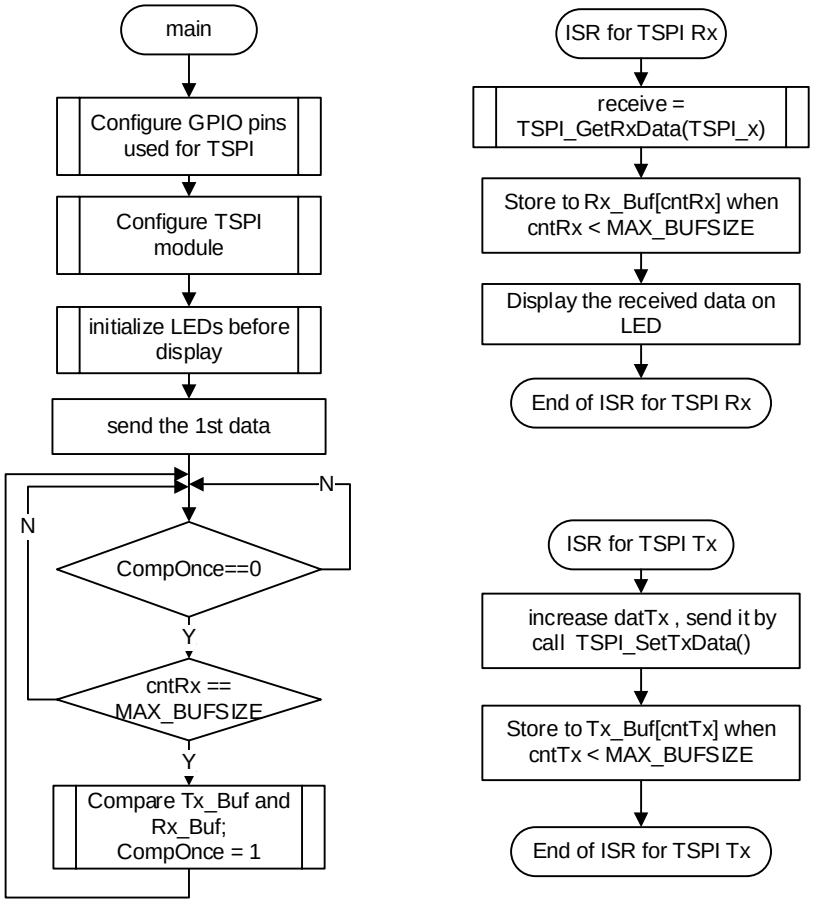
7-11 TSPI

7-11-1 Example: Basic transfer

This is a simple example based on the TX00 Peripheral Driver (TSPI).

The example includes:
1. Basic setup operation of TSPI
2. TSPI interrupt of Tx and Rx

- **Flowchart**



- **Code and Explanation for the Example**

At first, set the clock supply.

```
        /* enable clock supply to TSPI */
    CG_SetFcPeriphA(CG_FC_PERIPH_TSPI, ENABLE);
```

Next, configure the GPIO pins for TSPI by setting the CR, FR1, IE register of proper port. And initialize LEDs port for display.

Then, configure the TSPI module by calling driver functions.

```
    void ConfigTSPI(void)
```

```
{
    /* Single transfer */
    uint8_t transfer_num = 1U;
    /* maximum clock: clk = phiT0/2, divider=1 */
    TSPI_BaudClock clk = TSPI_PHIT0_DIVIDE_2;
    uint8_t divider = 1U;

    TSPI_InitTypeDef init = { 0U };

    TSPI_Enable(TSPI_x);
    TSPI_SWReset(TSPI_x);

    TSPI_SetTxRxCtrl(TSPI_x, ENABLE);
    TSPI_SelectMode(TSPI_x, TSPI_MODE_SIO);
    TSPI_SelectTransferMode(TSPI_x, TSPI_TRMODE_TXRX);

    TSPI_SetTransferNum(TSPI_x, transfer_num);

    TSPI_SetFIFOLevelINT(TSPI_x, TSPI_TX , 0U);
    TSPI_SetFIFOLevelINT(TSPI_x, TSPI_RX , 0U);

    /* enable TSPI interrupt */
    TSPI_SetINT(TSPI_x, TSPI_INT_ALL, ENABLE);

#ifdef BITRATE_MIN
    clk = TSPI_PHIT0_DIVIDE_1024;
    divider = 16U;
#endif
    TSPI_SetBaudRate(TSPI_x, clk, divider );

    init.DataDirection = TSPI_LSB_FIRST;
    init.FrameLength = sizeof(datTx) * 8U;          /* size of datTx in main() */
    init.CycleBetweenFrames = 1U;
    init.ClkPolority = TSPI_CLKPOL_LOW;
    init.ShortestIdleCycle = 1U;
    init.CS0Polarity = TSPI_CS_POLARITY_NEGATIVE;
    init.CS1Polarity = TSPI_CS_POLARITY_NEGATIVE;
    init.DelayCycleNum_CS_SCLK = 1U;
    init.DelayCycleNum_Negate_CS = 1U;
    init.ParityCheck = DISABLE;
    init.Parity = TSPI_PARITY_ODD;
    init.LastBit_Hold_Time = 0U;

    TSPI_Init(TSPI_x, &init);
```

```
    NVIC_ClearPendingIRQ(INTSPIRX_IRQn);
    NVIC_EnableIRQ(INTSPIRX_IRQn);

    NVIC_ClearPendingIRQ(INTSPITX_IRQn);
    NVIC_EnableIRQ(INTSPITX_IRQn);

    NVIC_ClearPendingIRQ(INTSPIERR_IRQn);
    NVIC_EnableIRQ(INTSPIERR_IRQn);

    __enable_irq();


}
```

After that, send the 1st data to trigger the Tx interrupt.

```
  Tx_Buf[0] = datTx;
  TSPI_SetTxData(TSPI_x, datTx);
```

In ISR of Tx, continue sending the rest data.

```
void INTSPITX_IRQHandler(void)
{
    datTx++;
    TSPI_SetTxData(TSPI_x, datTx);
    if (cntTx < MAX_BUFSIZE) {
        Tx_Buf[cntTx] = datTx;
        cntTx++;
    }
}
```

Use ISR of Rx to receive data

```
void INTSPIRX_IRQHandler(void)
{
    uint8_t tmp = 0U;

    receive = TSPI_GetRxData(TSPI_x);
    if (cntRx < MAX_BUFSIZE) {
        Rx_Buf[cntRx] = (uint8_t)receive;
        cntRx++;
    }
    /* adjust the loop blinking speed */
```

```
        tmp = (uint8_t) (receive >> 4U);
        LED_Off(~tmp);
        LED_On(tmp);
    }
```

7-12 UART

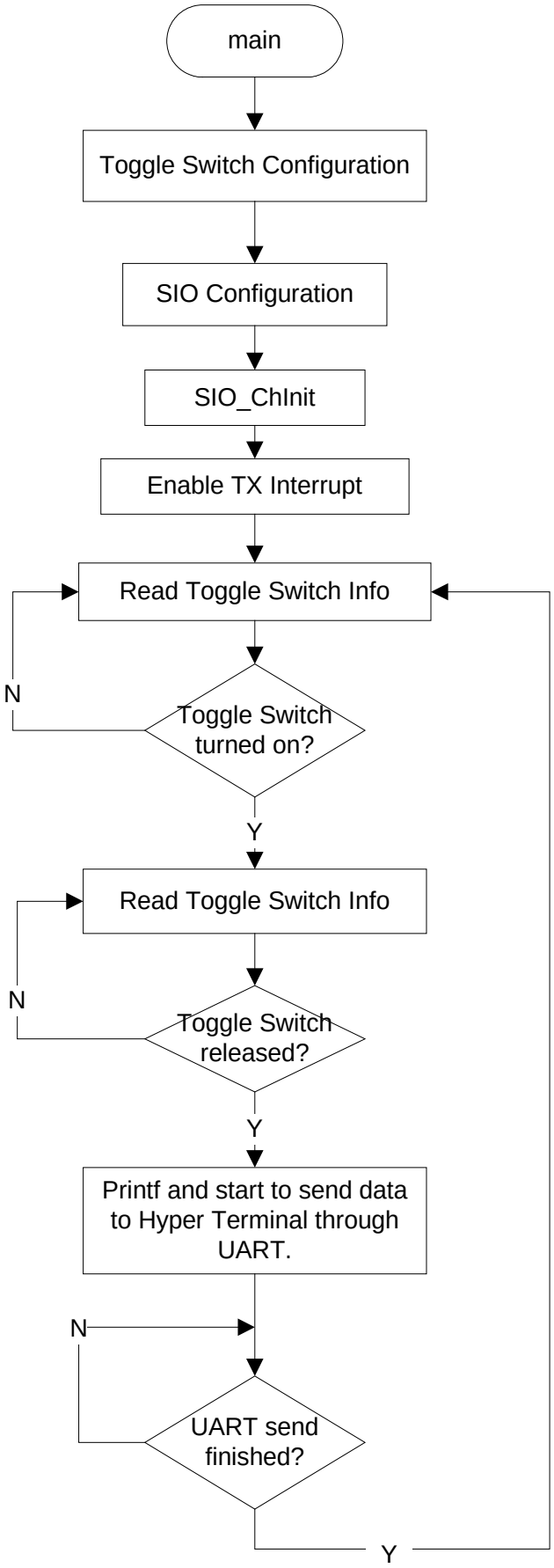
7-12-1 Example: Retarget

This is a simple example based on the TX00 Peripheral Driver (UART, GPIO).

The example includes:

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

- **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_PE, GPIO_BIT_2, ENABLE);
GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_2, DISABLE);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_2);
```

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 */
};
```

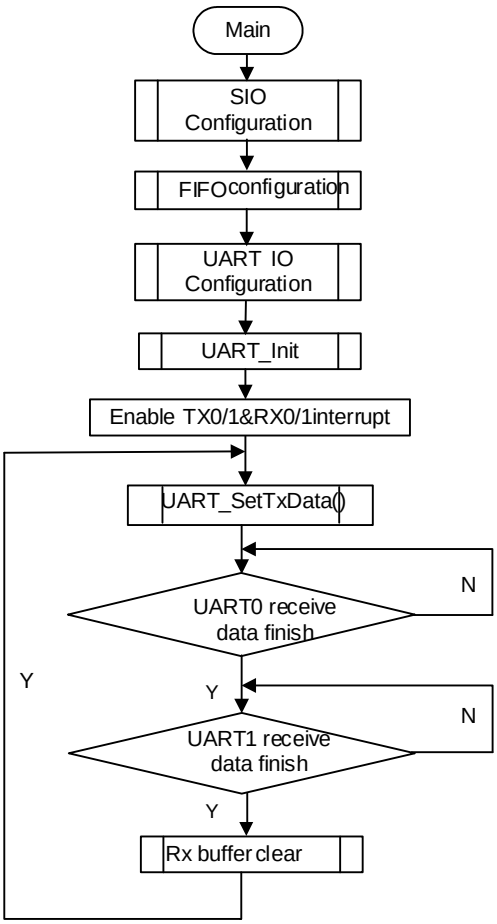
```
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE * f)
#elif defined ( __ICCARM__ )       /*IAR Compiler */
int putchar(int ch)
#endif
{
      return (send_char(ch));
}

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;
}
```

7-12-2 Example: UART FIFO

This is a simple example based on the TX00 Peripheral Driver (UART, GPIO).

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

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART.

Use GPIO peripheral drivers configure GPIO for UART0 and UART1.

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
    if (SCx == TSB_SC0) {
        T_TSB_PC->CR |= GPIO_BIT_2;
        TSB_PC->FR1 |= GPIO_BIT_2;
        TSB_PC->FR1 |= GPIO_BIT_3;
        TSB_PC->IE |= GPIO_BIT_3;
    } else if (SCx == TSB_SC1) {
        TSB_PE->CR |= GPIO_BIT_2
        TSB_PE->FR1 |= GPIO_BIT_2;
        TSB_PE->FR1 |= GPIO_BIT_1;
        TSB_PE->IE |= GPIO_BIT_1;
    }
}
```

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

```
    UART_InitTypeDef myUART;

    /* enable clock supply to SIO0 */
    CG_SetFcPeriphA(CG_FC_PERIPH_SIO0, ENABLE);
```

```
    /* enable clock supply to SIO1 */
    CG_SetFcPeriphB(CG_FC_PERIPH_SIO1, ENABLE);
    /* configure SIO0 for reception */
    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;
```

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_TxBufferClear(UART0);
    UART_TxBufferClear(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:

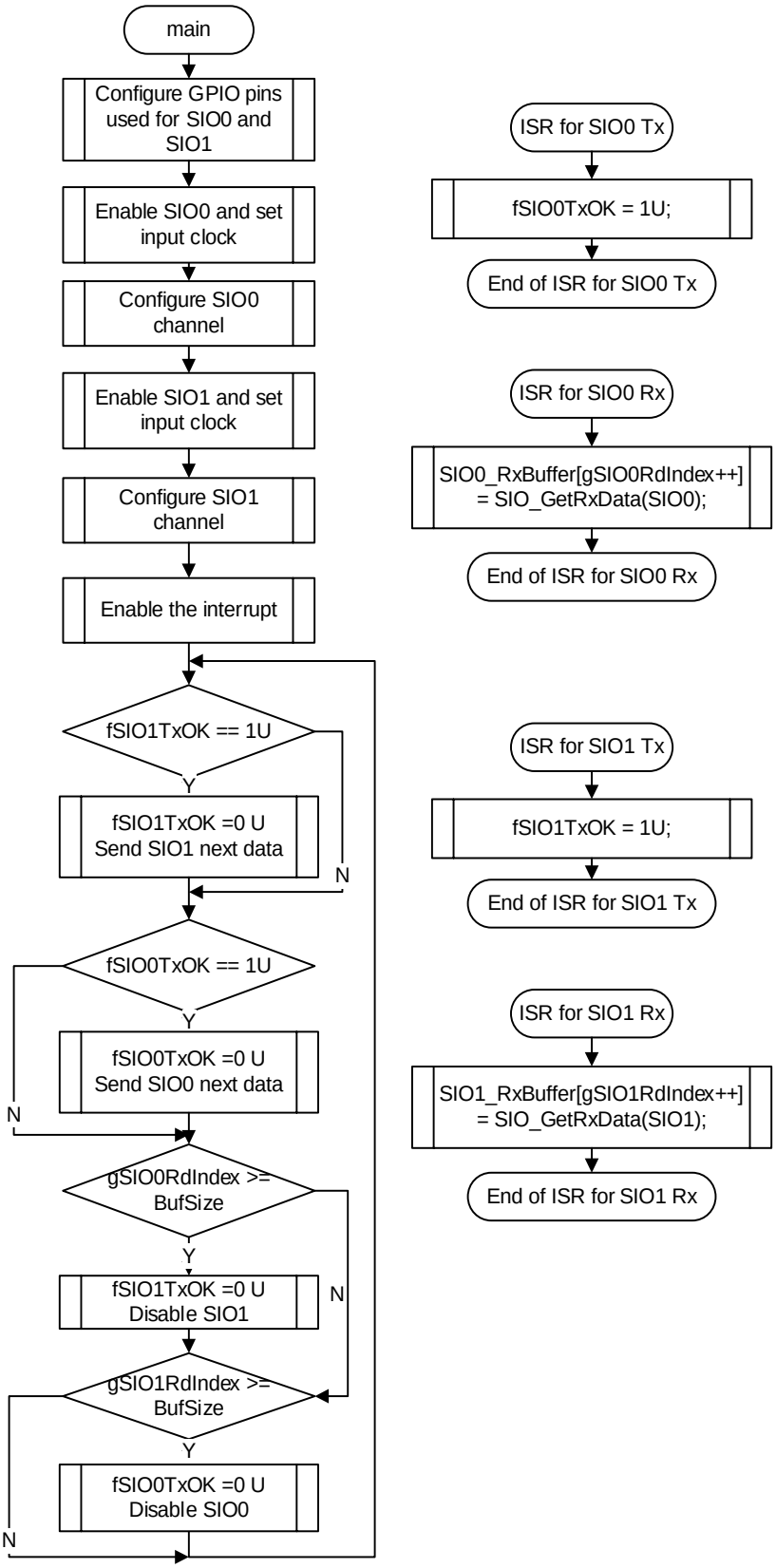
```
void INTRX1_IRQHandler(void)
{
    volatile UART_Err err;

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

7-12-3 Example: SIO

This is a simple example based on the TX00 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, set the clock supply and configure the IO port of SIO0.

```
/* enable clock supply to SIO0 */
CG_SetFcPeriphA(CG_FC_PERIPH_SIO0, ENABLE);
```

```
/* enable clock supply to SIO1 */
CG_SetFcPeriphB(CG_FC_PERIPH_SIO1, ENABLE);

/*configure the IO port of SIO */
SIO_Configure();
```

Next, 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.

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

/*initialize the SIO0 struct */
SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO0_Init.TIDLE = SIO_TIDLE_HIGH;
SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_8;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_TS2;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;

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

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.TIDLE = SIO_TIDLE_HIGH;
SIO1_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO1_Init.TransferDir = SIO_LSB_FRIST;
SIO1_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO1_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO1_Init.TXDEMP = SIO_TXDEMP_HIGH;
SIO1_Init.EHOLDTime = SIO_EHOLD_FC_64;

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

Enable the interrupt of SIO TX, RX.

```
/* Enable SIO0 Channel TX interrupt */
NVIC_EnableIRQ(INTTX0_IRQn);
/* Enable SIO0 Channel RX interrupt */
NVIC_EnableIRQ(INTRX0_IRQn);

/* Enable SIO1 Channel TX interrupt */
NVIC_EnableIRQ(INTTX1_IRQn);
/* Enable SIO1 Channel RX interrupt */
NVIC_EnableIRQ(INTRX1_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)
{
      SIO1_RxBuffer[gSIO1RdIndex++] = SIO_GetRxData(SIO1);
}
```

7-13 WDT

7-13-1 Example:WDT

This is a simple example based on the TX00 Peripheral Driver (WDT, GPIO).
The example includes:

1. WDT initialization
2. In DEMO1 WDT isn't cleared before timer overflow, NMI interrupt is generated _LED1 will blink once
3. In DEMO2 WDT is cleared before timer overflow, the LED0 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;
```

Set the clock supply.

```
/* enable clock supply to WDT */
CG_SetFcPeriphB(CG_FC_PERIPH_WDT, ENABLE);
```

Initialize WDT and enable it.

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

In DEMO1 Waiting for the NMI interrupt flag.

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

In DEMO1 When NMI interrupt is occurred the WDT will be disabled and the LED1 will blink once.

```
WDT_Disable();
```

In DEMO2 WDT will be cleared and the LED0 will blink all the time.

```
WDT_WriteClearCode();
```