TOSHIBA

TX03 Peripheral Driver Usage Example (TMPM341)

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M341UE-01E

RESTRICTIONS ON PRODUCT USE

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

Index

1 General description 1
2 Overview 1
3 Build-in hardware usage 1
4 Pin Usage 4
5 Development Environment 8
6 Functions 9
6-1 Operation mode 9
6-2 OFD 10
6-3 ADC 10
6-3-1 ADC Data Read 10
6-4 CG 10
6-4-1 Power mode change 10
6-5 DAC 11
6-5-1 Sawtooth waveform output 11
6-6 DMAC 11
6-6-1 Memory to peripheral 11
6-7 EXB 12
6-7-1 SRAM Read/Write 12
6-8 FLASH 12
6-9 GPIO 14
6-10 PHC 14
6-11 SBI 15
6-12 SIO 16
6-12-1 Retarget 16
6-12-2 UART FIFO 16
6-12-3 SIO 16
6-13 SSP 16
6-13-1 SSP Self Loop Back 16
6-14 TMRB 17
6-14-1 General Timer 17
6-14-2 PPG Output 17
6-15 TMRD 17
6-15-1 Interval timer 17
6-15-2 Interlock PPG output 17
6-16 WDT 17
7 Software 18
7-1 ADC 19
7-1-1 Example: ADC Data Read 19
7-2 CG 21
7-2-1 Example: Power mode change 21
7-3 DAC 24

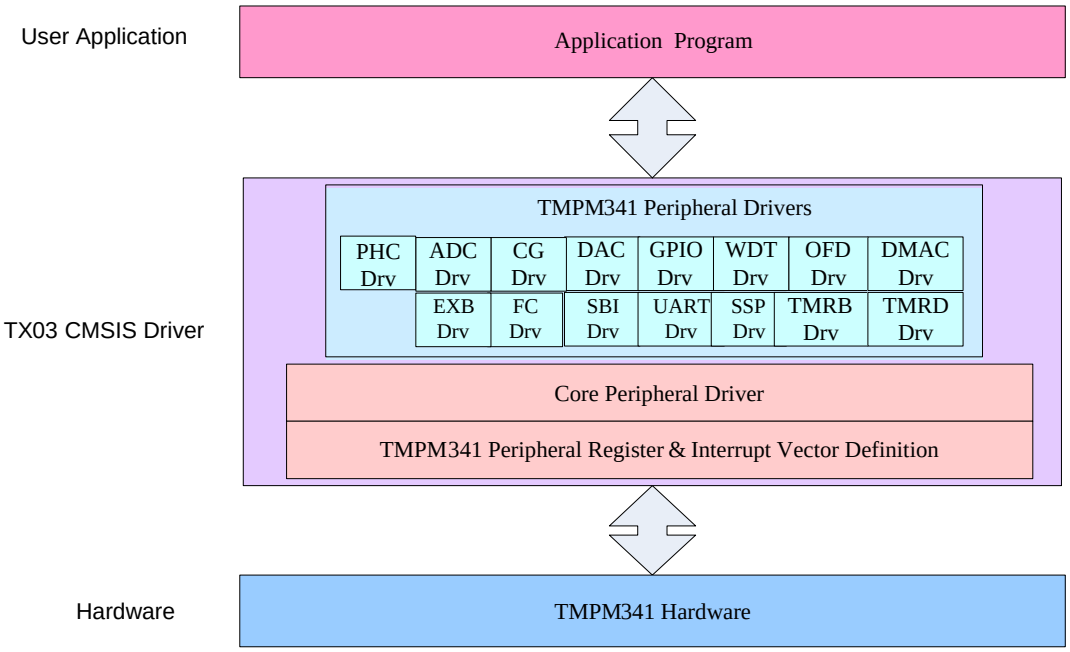
7-3-1 Example: Sawtooth waveform output 24
7-4 DMAC 26
7-4-1 Example: Memory to peripheral 26
7-5 EXB 28
7-5-1 Example: Read/Write SRAM 28
7-6 FLASH 32
7-7 GPIO 36
7-1 OFD 37
7-2 PHC 40
7-3 SBI 42
7-4 SIO 45
7-4-1 Example: Retarget 45
7-4-2 Example: UART FIFO 49
7-4-3 Example: SIO 52
7-5 SSP 55
7-5-1 Example: SSP Self Loop Back 55
7-6 TMRB 58
7-6-1 Example: General Timer 58
7-6-2 Example: PPG Output 60
7-7 TMRD 64
7-7-1 Interval timer 64
7-7-2 Interlock PPG output 66
7-8 WDT 70

1 General description

The example programs described hereafter are specially designed for TOSHIBA TMPM341 MCU. The programs can be executed individually each to show the main MCU functions. Users can utilize some portions of the programs and integrate them into users' own programs to perform customized functions.

2 Overview

User application utilizes TX03 peripheral driver as following.



Note: The CMSIS core included in the IDE tool (IAR EWARM and KEIL MDK) is used for CMSIS core peripheral access layer.

3 Build-in hardware usage

Hardware	Channel	Use presence, use
CG	Clock Gear	Used for CG demo
	PLL	PLL on (8x)
Standby mode	-	Use STOP1 mode

Hardware	Channel	Use presence, use
SysTick	-	Unused
Watch dog timer (WDT)	-	Used for WDT and OFD demo
External interrupt (INT)	INT0	Unused
	INT1	Used
	INT2	Unused
	INT3	Unused
	INT4	Unused
	INT5	Unused
	INT6	Unused
	INT7	Unused
	INT8	Unused
	INT9	Unused
	INTA	Unused
	INTB	Unused
SIO	SIO0	Used for UART & DMAC demo Tx Used for UART retarget demo Used for SIO_Demo Used for UART_FIFO_Demo
	SIO1	Used for SIO_Demo Used for UART_FIFO_Demo
	SIO2	Used for UART & DMAC demo Rx
	SIO3	Unused
	SIO4	Unused
Synchronous serial port (SSP)	SSP	Used for SSP Self Loop Back demo
Serial bus (SBI)	SBI0	Used for Master Tx
	SBI1	Used for Slave Rx
16-bit timer	TMRB0	Used for TMRB: General Timer
	TMRB1	Unused
	TMRB2	Unused
	TMRB3	Unused
	TMRB4	Unused
	TMRB5	Unused
	TMRB6	Unused
	TMRB7	Used for TMRB: PPG Output
	TMRB8	Unused
	TMRB9	Unused
OFD	-	Used for Startup demo
DAC	DA0	Used for DAC demo (DAC0 output)
	DA1	Unused

Hardware	Channel	Use presence, use
TMRD	TMRD0	Used for TMRD: interval timer demo and interlock PPG demo
	TMRD1	Used for TMRD: interlock PPG demo
10-bit A/D converter	AIN0	Unused
	AIN1	Unused
	AIN2	Unused
	AIN3	Unused
	AIN4	Unused
	AIN5	Unused
	AIN6	Unused
	AIN7	Unused
	AIN8	Unused
	AIN9	Unused
	AIN10	Unused
	AIN11	Unused
	AIN12	Unused
	AIN13	Unused
	AIN14	Used for ADC data read
External bus interface (EXB)	CS0	Unused
	CS1	Used for SRAM write/read
PHCNT	PHCNT0	Used for PHC counter

4 Pin Usage

The example programs are tested on the IAR TMPM341-SK Evaluation Board except SBI & SSP that are tested on TOSHIBA TMPM341 Evaluation Board.
Following is pin usage for example programs on IAR TMPM341-SK Evaluation Board.

No	Name	Usage
A1	PK7,AIN15	Unused
A2	PK4,AIN12	Unused
A3	PJ7,AIN07	Unused
A4	PJ3,AIN03,PHC1IN1	Unused
A5	DVDD3A	+3V
A6	NMI	Unused
A7	X1/ECLKIN	Unused
A8	DVSSC	GND
A9	X2	Unused
A10	P15,TCK/SWCLK	Unused
A11	PH5,TRACEDATA3	Unused
B1	PK6,AIN14	AIN14
B2	PK5,AIN13	Unused
B3	PK0,AIN08,TB1IN0	Unused
B4	PJ4,AIN04,PHC2IN0	Unused
B5	PJ0,AIN00,PHC0IN0	PHCNT0 input
B6	RESET	RESET
B7	PH3,PHC3IN0	Unused
B8	RVSS	GND
B9	RVDD3	+3V
B10	PI4,TDI	Unused
B11	PH6,TACEDATA2	Unused
C1	AVDD3	+3V
C2	AVREFH	Unused
C3	PK1,AIN09,INTA,TB1IN1	Unused
C4	PJ5,AIN05,PHC2IN1	Unused
C5	PJ1,AIN05,PHC2IN1	Unused
C6	MODE	GND
C7	PH4,PHC3IN1,TB5OUT	Unused
C8	PH0,TXD4	Unused
C9	PI7,TDO/SWDIO	Unused
C10	PI6,TMS/SWDIO	Unused
C11	DVDD3A	+3V
D1	AVSS	GND

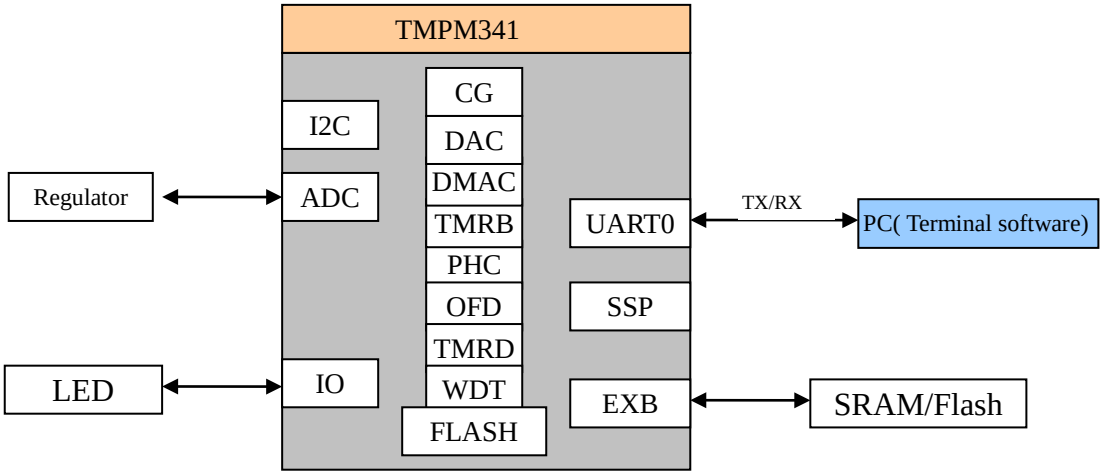
No	Name	Usage
D2	AVREFL	Unused
D3	PK2,AIN10,TB6IN0	Unused
D4	PJ6,AIN06,TB0IN0	Unused
D5	PJ2,AIN02,PHC1IN0	Unused
D6	PG7,INT1,TB9IN1	Unused
D7	PG6,SCLK3,TB9IN0	Unused
D8	PH1,RXD4	Unused
D9	PI2,TRACECLK	Unused
D10	PI3,TRST	Unused
D11	DVSSA	GND
E1	DVSSB	GND
E2	DVDD3B	+3V
E3	PK3,AIN11,INTB, TB6IN1	Unused
E4	DA0	output
E5	DA1	Unused
E8	PH2,SCLK4,CTS4	Unused
E9	PG5,RXD3,TB8IN1	Unused
E10	PI1,TRACEDATA0	Unused
E11	DVSSB	GND
F1	PA0,D0/AD0	AD0
F2	PF0,BOOT,TB6OUT	Unused
F3	PF1,RD	Unused
F4	INTLV	Unused
F8	PG4,TXD3,TB8IN0	Unused
F9	PG3,INT0	Unused
F10	PI0,TRACEDATA1	Unused
G1	PA1,D1/AD1	AD1
G2	PA2,D2/AD2	AD2
G3	PF3,BELL	BELL
G4	PF2,WR	WR
G8	PPG2,SCK0	Unused
G9	PG1,SIO/SCL0,TB7IN1	Unused
G10	PD7,A15,SPFSS,SCOUT	Unused
G11	PD6,A14,SPCLK	Unused
H1	PA3,D3/AD3	AD3
H2	PA4,D4/AD4	AD4
H3	PF4,BELH,INT6,TB5IN0	BELH
H4	PF5,CS1,INT7,TB5IN1	CS1
H5	PC1,A1,RXD1,TB2IN1	SIO1 RXD1
H6	PC3,A3,INT2,TB1OUT	Unused
H7	PC6,A6,SCLK2,TB4IN0	Unused

No	Name	Usage
H8	PC7,A7,INT3,TB4IN1	Unused
H9	PG0,SO0/SDA0	Unused
H10	PD5,A13,SPDI	Unused
H11	PD4,A12,,SPD0	Unused
J1	PA5, D5/AD5	AD5
J2	PA6, D6/AD6	AD6
J3	PF6, CS0	CS0
J4	PF7, ALE	ALE
J5	PC0, A0, TXD1, TB2IN0	SIO1 TXD1
J6	PC2, A2, SCLK1, TB0OUT, CTS1	SIO1 SCLK1
J7	PC5, A5, RXD2, TB3IN1	Unused
J8	PE0, TXD0, A16	A16 / SIO0 TXD0
J9	PE2, SCLK0, A18, TB2OUT, CTS0	LED output/ A18/ SIO0 SCLK0
J10	PD3, A11, INT4, ADTRG	Unused
J11	PD2, A10, SCK1, TB9OUT	Unused
K1	PA7, D7/AD7	AD7
K2	PB1, D9/AD9, A1	AD9
K3	PB3, D11/AD11, A3	AD11
K4	PB5, D13/AD13, A5	AD13
K5	PB7, D15/AD15, A7	AD15
K6	DVDD3B	Unused
K7	PC4, A4, TXD2, TB3IN0	Unused
K8	PE1, RXD0, A17	A17 / SIO0 RXD0
K9	PE3, INT5, A19, TB3OUT	A19
K10	PD1, A9, SI1/SCL1, TB8OUT	Unused
K11	PD0, A8, SO1/SDA1, TB7OUT	PPG Output
L1	BSC	Unused
L2	PB0, D8/AD8, A0	AD8
L3	PB2, D10/AD10, A2	AD10
L4	PB4, D12/AD12, A4	AD12
L5	PB6, D14/AD14, A6	AD14
L6	DVSSB	GND
L7	PE4, A20, TD0OUT0	TMRD PPG phase A0 output/ A20
L8	PE5, A21, TD0OUT1	A21
L9	PE6, A22, TD1OUT0	TMRD PPG phase B0 output/ A22
L10	PE7, A23, TD1OUT1	A23

No	Name	Usage
L11	FTEST3	Unused

5 Development Environment

Following is development environment:



1. Hardware board:
   1) IAR TMPM341-SK Evaluation Board

2. Development tool:
   - IAR:
     2) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
     3) IDE: IAR Embedded workbench 6.30.1 version
   - KEIL:
     1) U-Link: Realview ULINK2
     2) IDE: KEIL uVision 4.21

6 Functions

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

6-1 Operation mode

There are four operation modes for TMPM341: NORMAL, IDLE, STOP1 and STOP2 mode.

IDLE, STOP1 and STOP2 mode are low power mode.

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

- **NORMAL mode:**

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

Note: Only STOP1 mode is demostrated in driver example.

- **STOP1 mode:**

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

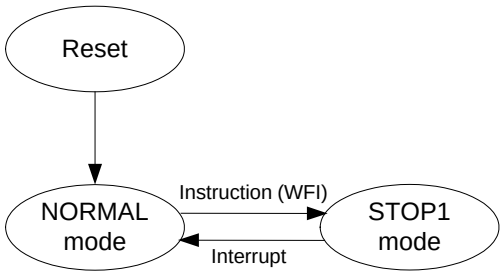
When returning from STOP1 mode, related bits CGPLLSEL<PLLSEL>, CGOSCCR <HWUPSEL>, <OSCSEL>, <XEN2> and <PLLON> are initialized and internal oscillator starts. After elapsed warm-up time, the NORMAL mode is operated.

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

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

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

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



6-2 OFD

This is a simple application based on the Peripheral Driver (OFD).
When the clock exceeds the OFD detection frequency range, OFD will generate a reset for I/O. Then software will be reset. The OFD reset flag will be set. If the flag is detected, OFD will be disabled and LED2 will be blinking.

6-3 ADC

6-3-1 ADC Data Read

There is a potentiometer that connected to PK6/AIN14. The voltage on it will be measured and print to stdout. As stdout is retargeted to UART, connect UART0 with a PC and the AD result can be displayed on terminal software.

6-4 CG

6-4-1 Power mode change

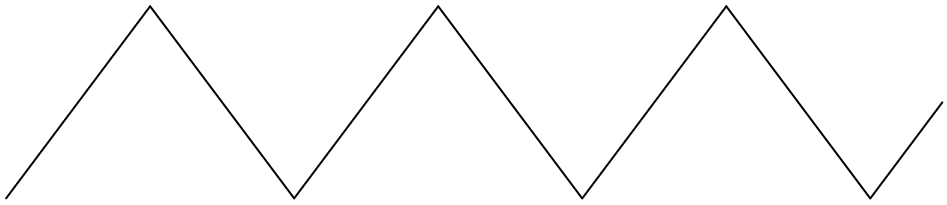
Change the CPU operation mode. Only 2 modes are supported, NORMAL and STOP1. Press the key to switch the power mode. LED0 is turned on in the NORMAL mode and turned off in the STOP1 mode.

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

6-5 DAC

6-5-1 Sawtooth waveform output

Change the output code of DAx from 0 to 1023 at the fixed interval; when it reaches 1023,change the output code of DAx from 1023 to 0 at the fixed interval. So the waveform outputted from the DAx pin will be as the following:



6-6 DMAC

6-6-1 Memory to peripheral

This program implements transmission from UART0 to UART2, the string “TOSHIBA” is sent via UART0 to UART2.
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 UART2. After UART2 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 IAR TMPM341-SK Evaluation Board must be modified: Connect the TX of UART0 and RX of UART2 via wire.

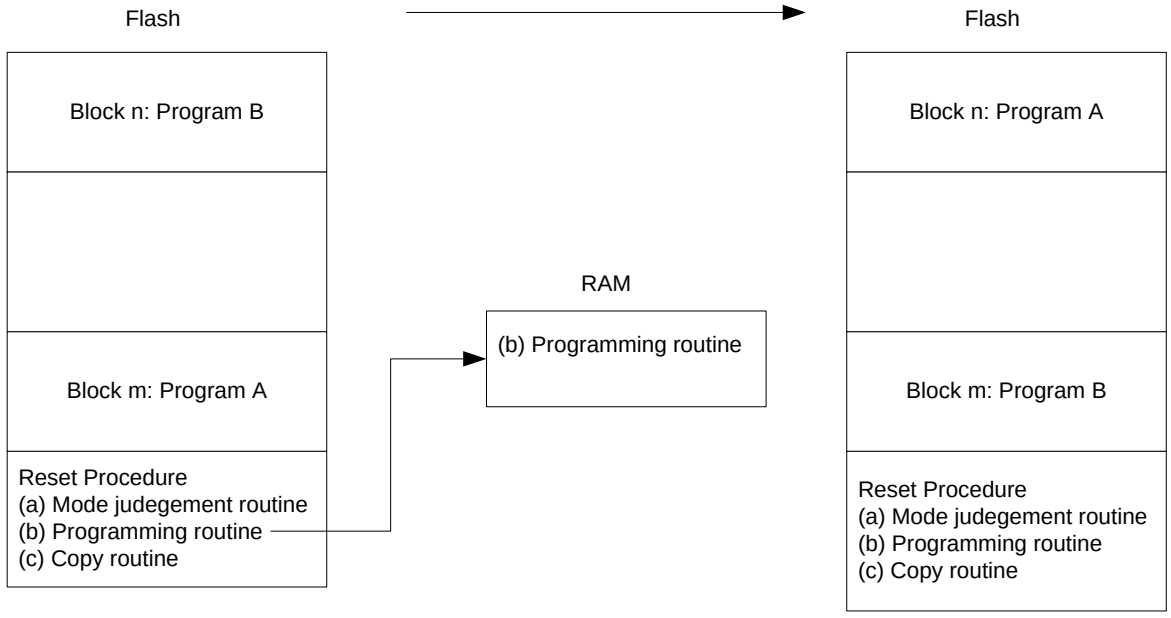
6-7 EXB

6-7-1 SRAM Read/Write

This program implements read/write the external SRAM assembled on TMPM341 evaluation board and EXB is set as 16-bit bus width on multiplex bus. The A.C specifications of SRAM (cycles time) used is for specific SRAM chip. The external SRAM is IS62WV51216BLL with 1Mbyte size.

6-8 FLASH

This program demonstrates the feature of flash APIs such as erasing and writing 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.
By default, the program A will run firstly
－Button SW1 is used for mode judgment in Reset procedure
SW1 is released: high level Normal mode
SW1 is pressed: low level User boot mode

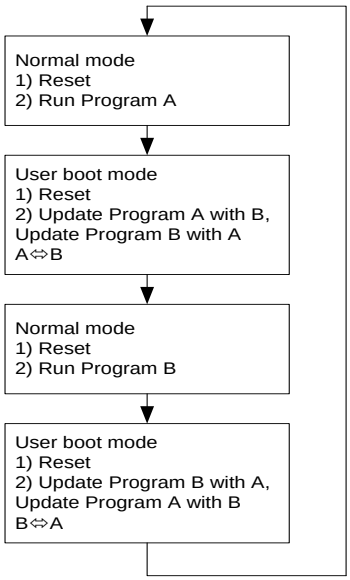


Demo sequence

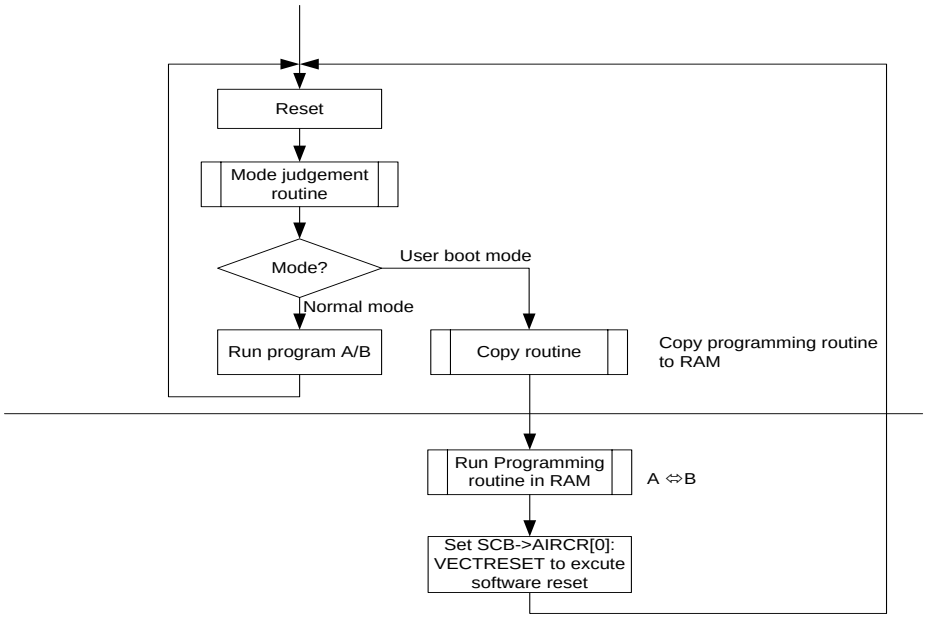
(1) Power on
When the button SW1 is released, then power on or press RESET button. DemoA will run at first. LED2 is always on.

(2) When pressing the button SW1, then press RESET button, there will be two processes:
Exchange step: LED2 will flash 8 times to show it has entered user boot mode; then LED2 will flash 4 times to show RAM transmission of flash API has been completed; then LED2 will flash twice to show copy work of Program A&B to RAM has been completed.
Exchange of successful step : LED2 will flash 12 times

(3) After exchange of successful, when the button "SW1" is released, system will reset automatically to run DemoB.



・**Demo process flow**



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

LED2 flashes for 8 times (user boot mode)

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

LED display sample:
LED2 flashes for 4 times: (RAM transferring)
LED2 flashes for 2 times: (Swap A and B)
LED2 flashes for 12 times: (Swap finished, wait for reset)

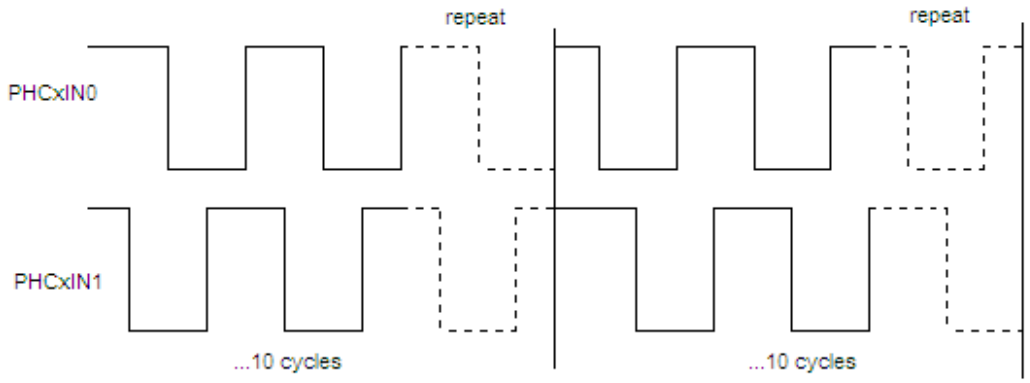
6-9 GPIO

This is a simple application based on the Peripheral Driver (GPIO), use GPIO functions to configure low 4bit of GPIO Port E as output to drive LED, then turn on and turn off the LED connected in PE0 to PE3.

6-10 PHC

This function intends to use the Quadruple mode of PHC.

Using PortJ-Pin2 and PortJ-Pin3 to simulate two-phase input as PHC0IN0 and PHC0IN1 input. The input waveform like following graphic:



The up-and-down counter is increment or decrement when the state transition of the two-phase pulse changes once.

Counter should get 40 times for increment and 40 times for decrement.

Counter results can be checked by RAM variable count_result[] in debugger.
count_result[0] = 0x7FFE,
count_result[1] = 0x7FFD,
count_result[2] = 0x7FFC
…
count_result[38] = 0x7FD8,
count_result[39] = 0x7FD7,
count_result[40] = 0x7FD8,
count_result[41] = 0x7FD9,
count_result[42] = 0x7FDA
…
count_result[79] = 0x7FFF.

Note: In order to run this sample software, use TOSHIBA TMPM341 evaluation board which must be modified.
Connect the pin PJ0 (PHC0IN0) and pin PJ2 (PortJ-Pin2 output).
Connect the pin PJ1 (PHC0IN0) and pin PJ3 (PortJ-Pin3 output).

6-11 SBI

This program intends to support the I2C bus slave mode.
Connect two I2C buses (SBI0 & SBI1) on TOSHIBA TMPM341 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 TMPM341 evaluation board which must be modified.
Connect the pin PG1(SCL0) and pin PD1(SCL1).
Connect the pin PG0(SDA0) and pin PD0(SDA1).

6-12 SIO

6-12-1 Retarget

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

6-12-2 UART FIFO

In this sample program, UART0 sent the data "TMPM3411" to UART1 use FIFO, and at same time UART0 receive the data "TMPM3412" 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 = "TMPM3412",and RxBuffer1 = "TMPM3411".

6-12-3 SIO

This demo will show synchronously transfer and receive usage of SIO module in TMPM341
It use the channel SIO0, SIO1 and transfer data synchronously between them.(connect TXD0 with RXD1, connect TXD1 with RXD0, connect sclk0 with sclk1)

6-13 SSP

6-13-1 SSP Self Loop Back

For M341, there is one SSP channel. Infinite data is sent and checked if the received data is OK or not. And if transmission is set to lowest speed mode(lowest bit rate), what happens will be checked.

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

6-14 TMRB

6-14-1 General Timer

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

6-14-2 PPG Output

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

6-15 TMRD

6-15-1 Interval timer

This program implements a general timer by using TMRD0 16-bit interval timer.
Timer cycle is set to 500us.
Use this timer for LED2 blinking and the period is 1s (500ms ON and 500ms OFF).

6-15-2 Interlock PPG output

This program implements interlock PPG output by using TMRD.
PPG cycle is set to 500us, and duty is 50%.(250us H level, 250us L level).
Phase A is set to faster than phase B, and the phase shift (delay) (θ) is 120 degree.
The phase A0 and B0 can be observed by oscilloscope from PE4/TD0OUT0 and PE6/TD1OUT0.

6-16 WDT

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

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

The driver example step:

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

   DEMO1:

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

   DEMO2:

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

7 Software

This software project creates the sample applications based on IAR TMPM341-SK Evaluation Board which will demonstrate the main feature of TMPM341 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
        │  tmpm341_wdt_int.c
        ├─TX03_CMSIS
        │  │  core_cm3.c
        │  │  core_cm3.h
        │  │  system_TMPM341.c
        │  │  system_TMPM341.h
        │  │  TMPM341.h
        │  │
        │  └─startup
        │        startup_TMPM341.s
        │
        ├─TX03_Periph_Driver
        │  ├─inc
        │  │  tmpm341_wdt.h
        │  │  tmpm341_gpio.h
        │  │  tx03_common.h
```

```
│   │
│   └─src
│         tmpm341_wdt.c
│         tmpm341_gpio.c
└─TMPM341-SK
          led.c
          led.h
```

KEIL MDK:

```
├─WDT_NMI
      ├─APP
      │   main.c
      │   tmpm341_wdt_int.c
      │
      ├─TX03_CMSIS
      │    core_cm3.c
      │    system_TMPM341.c
      │    startup_TMPM341.s
      │
      ├─TX03_Periph_Driver
      │    tmpm341_wdt.c
      │    tmpm341_gpio.c
      │
      └─TMPM341-SK
           led.c
```

7-1 ADC

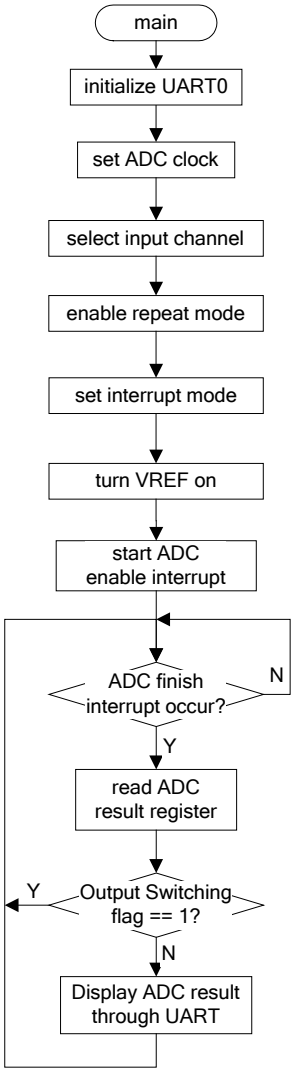
7-1-1 Example: ADC Data Read

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

The example includes:

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

- **Flowchart:**



- **Code and Explanation for the Example**

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

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

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

/* select ADC input channel */
ADC_SetInputChannel(ADC_AN_14);

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

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

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

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

Then start ADC and enable INTAD:

```
ADC_Start();

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

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

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

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

```
ADC_Result result = ADC_GetConvertResult(ADC_REG_00);

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

7-2 CG

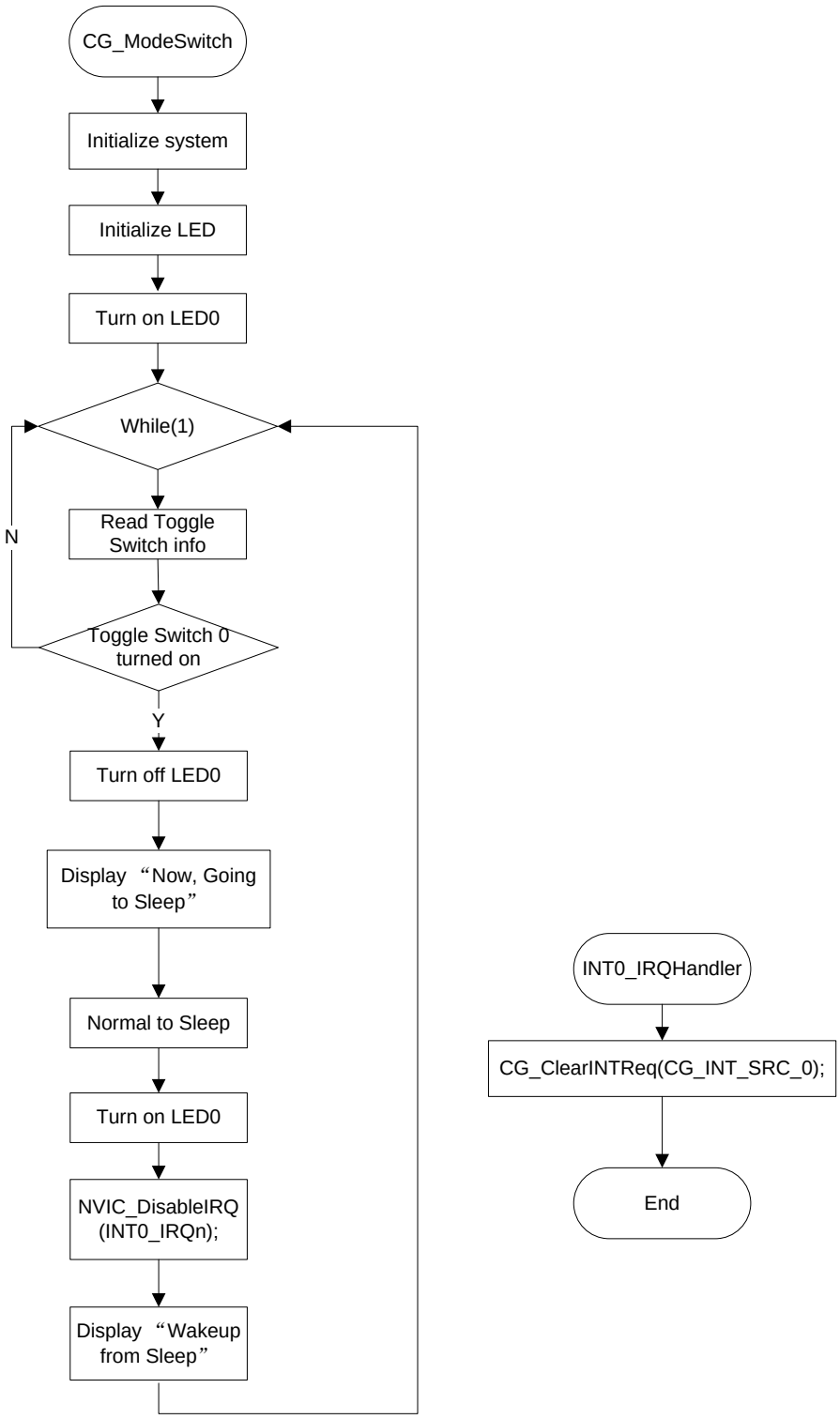
7-2-1 Example: Power mode change

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

The example includes:

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

- **Flowchart**



- **Code and Explanation for the Example**

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

Normal setup for CG (after reset)

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

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();
    CG_ClearINTReq(CG_INT_SRC_0);
    NVIC_ClearPendingIRQ(INT0_IRQn);
    NVIC_EnableIRQ(INT0_IRQn);
    CG_ClearINTReq(CG_INT_SRC_0);
    __enable_irq();

    /* 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_QUARTER_FPLL);
    } else {
        /*Do nothing */
    }

    return retval;
```

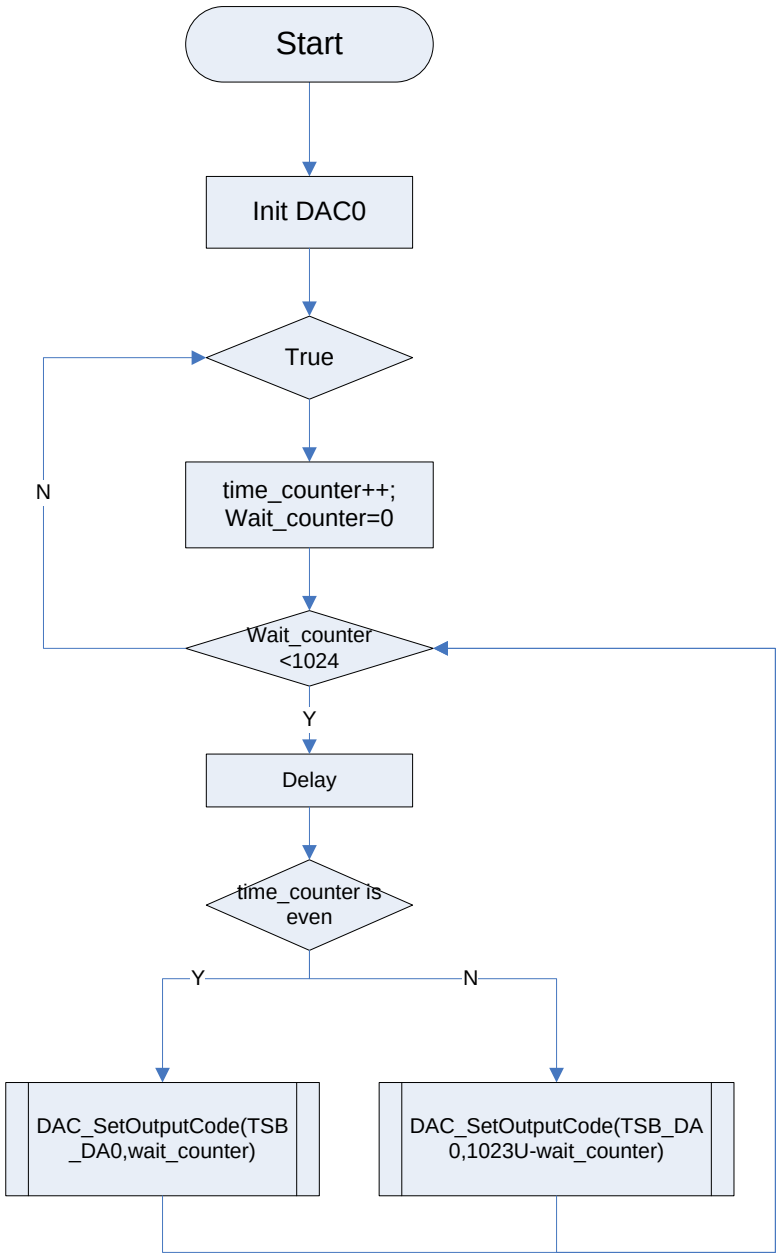
7-3 DAC

7-3-1 Example: Sawtooth waveform output

This is a simple example based on the TX03 Peripheral Driver DAC.

The example includes:
DAC0 sawtooth waveform output

- **Flowchart:**



- **Code and Explanation for the Example**

The following simple example is based on TX03 Peripheral Driver (DAC), which output sawtooth waveform from DA0 pin

Firstly set initial value for DA0 output code and start the operation of it

```
DAC_SetOutputCode(TSB_DA0,0U);
DAC_Start(TSB_DA0);
```

Change the output code at the fixed interval to generate sawtooth waveform.

```
while(1){
        time_counter++;

        for(wait_counter=0U;wait_counter<1024U;wait_counter++){
```

```
            for(i=0U;i<3000U;++i){
               /* Do nothing */
            }
            if( time_counter%2U == 1U){
                  DAC_SetOutputCode(TSB_DA0,wait_counter);
            }
            else{
                  DAC_SetOutputCode(TSB_DA0,1023U-wait_counter);
            }
      }
}
```

7-4 DMAC

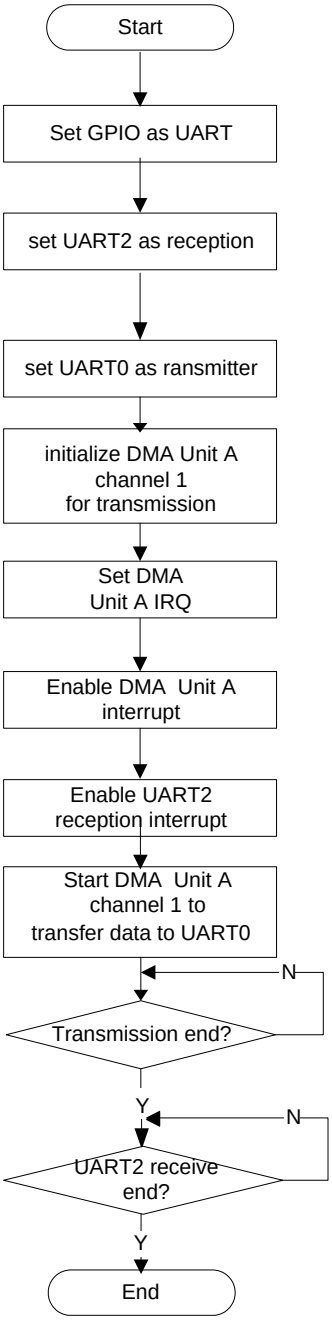
7-4-1 Example: Memory to peripheral

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

The example includes:

1. UART0/2 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 */
    }
```

7-5 EXB

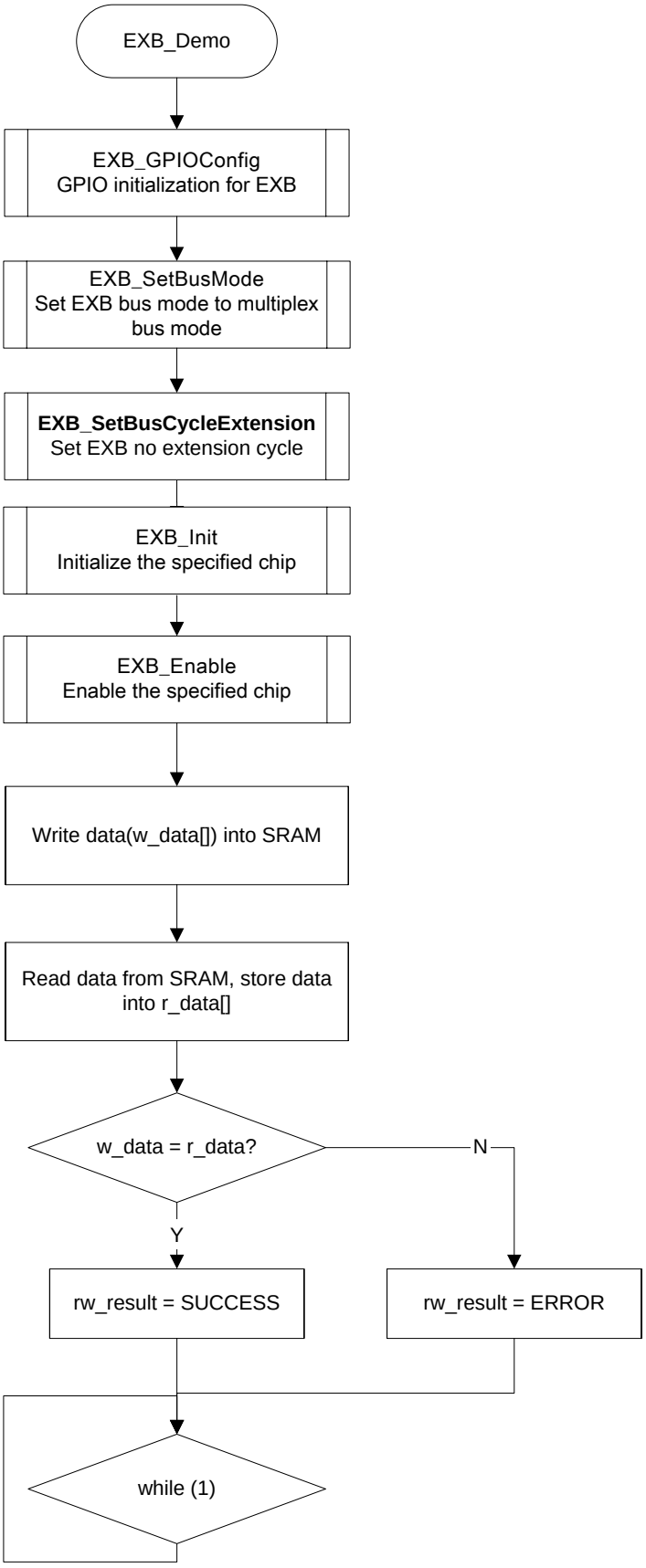
7-5-1 Example: Read/Write SRAM

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

The example includes:

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

- **Flowchart:**


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

- **Code and Explanation for the Example**

Firstly set initial value for EXB.

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

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

    EXB_InitTypeDef InitStruct = { 0U };

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

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

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

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

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

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

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

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

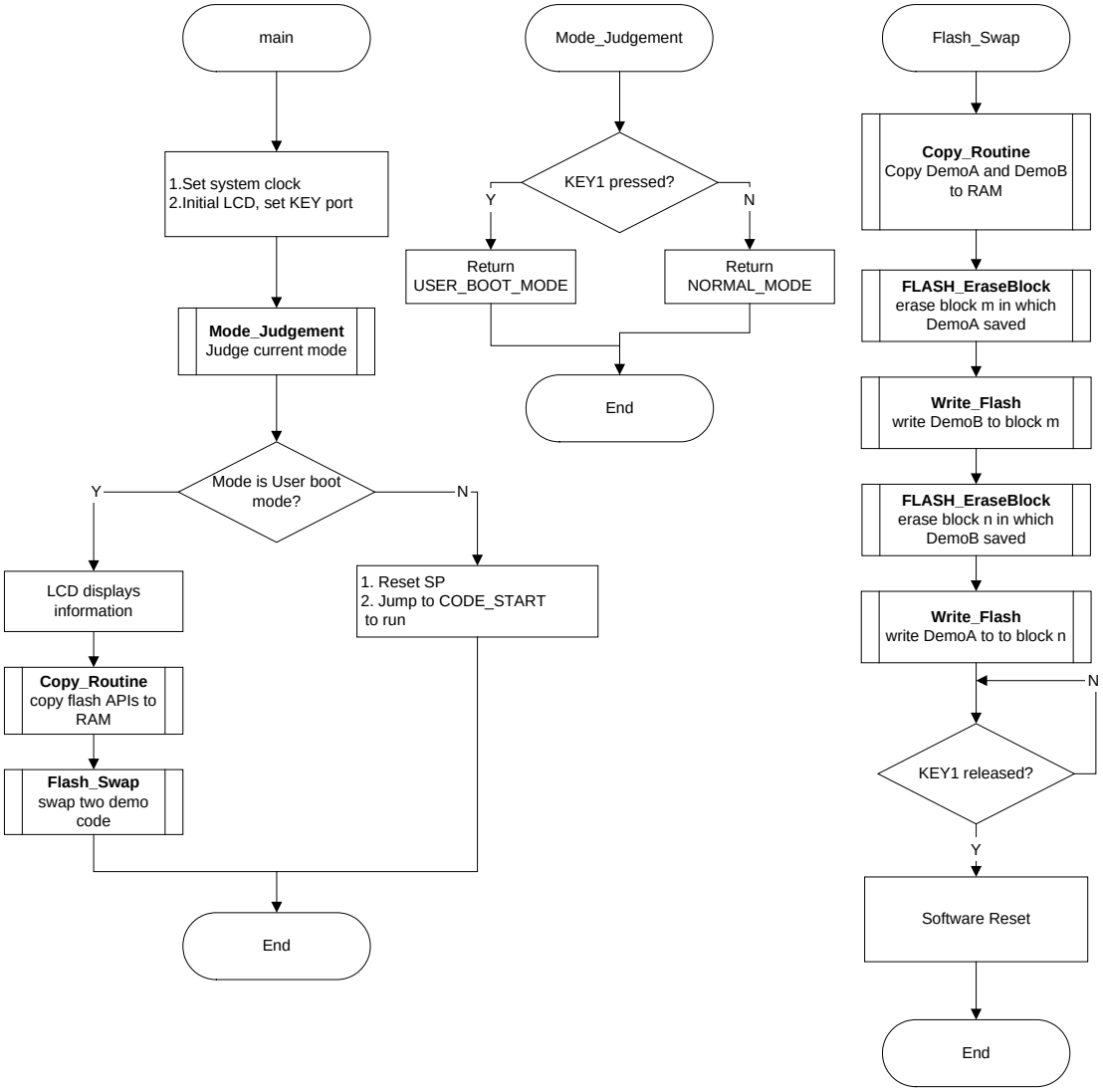
7-6 FLASH

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

The example includes:

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

- **Flowchart**



- **Code and Explanation for the Example**

At first configure system clock and initialize SWITCH port and LED. SWITCH port (GPIO) is used to judge current mode when reset and LED will display current operation information.

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

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

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

If the SWITCH is high 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 show).

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

If the SWITCH is low when reset, the routine will enter user boot mode. LED3 will show to indicate this mode. Then the routine will copy APIs for flash operation from address "FLASH_API_ROM" in Flash memory to address "FLASH_API_RAM" in RAM, because Flash memory cannot erase/program itself when runs the code at the same time. LED2 show to indicate RAM transferring.

```
Status_Display(0x08);      /* status 1: enter user boot mode */
Status_Display(0x04);      /* status 2: RAM transferring */
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(), which has been copied from Flash memory to RAM by using Copy_Routine() above.

In function Flash_Swap(), firstly it will copy Demo A and B from Flash memory to RAM, then call function FC_EraseBlock () and Write_Flash() to erase and program Flash memory. The code of Demo A and B will be exchanged in Flash memory. LED1 show to indicate swapping two demos.

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

Status_Display(0x02);            /* status 2: swap the demo */

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

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

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

/* write A to block n */
if (FC_SUCCESS == Write_Flash(DEMO_B_FLASH, DEMO_A_RAM,
```

```
    SIZE_DEMO_A)) {
        /* Do nothing */
    } else {
        return ERROR;
    }
```

LED3 and LED2 will show to indicate the swap operation has completed. After SWITCH changed to high, it will execute software reset by using SCB->AIRCR register. Then this demo will run again to enter normal mode, because Demo A and B have been exchanged, the address “CODE_START” has become the start address of Demo B, Demo B will run (LED1 show).

```
    Status_Display(0x0C);                    /* status 3: complete and restart */

    while (GPIO_ReadDataBit(GPIO_PD, GPIO_BIT_0) == GPIO_BIT_VALUE_0) {
                /* wait for KEY1 release */
    }
    reg_value = SCB->AIRCR;        /* software reset */
    reg_value = (uint32_t) 0x05FA0001;
    SCB->AIRCR = reg_value;
```

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

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

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

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

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

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

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

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

```
        source++;
    }
```

7-7 GPIO

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

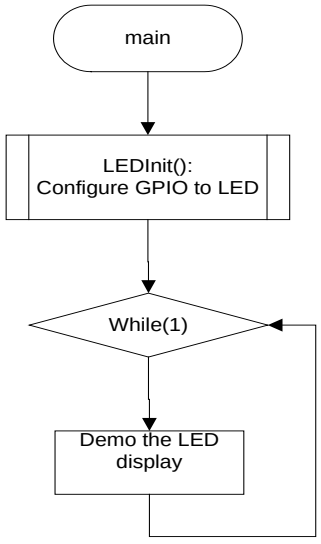
The example includes:

1. GPIO initialization
2. Write data to GPIO

Note:

1. Current demo software is tested on IAR TMPM341-SK board., use PE2 to control LED2 On/Off. The jumper E JP11 must be connected.

- **Flow chart:**



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

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

```
GPIO_SetOutput(LED_DATA_PORT,0X04U);
GPIO_WriteData(LED_DATA_PORT,0X04U);
```

In the While process, do the LED demo: LED on and LED off.

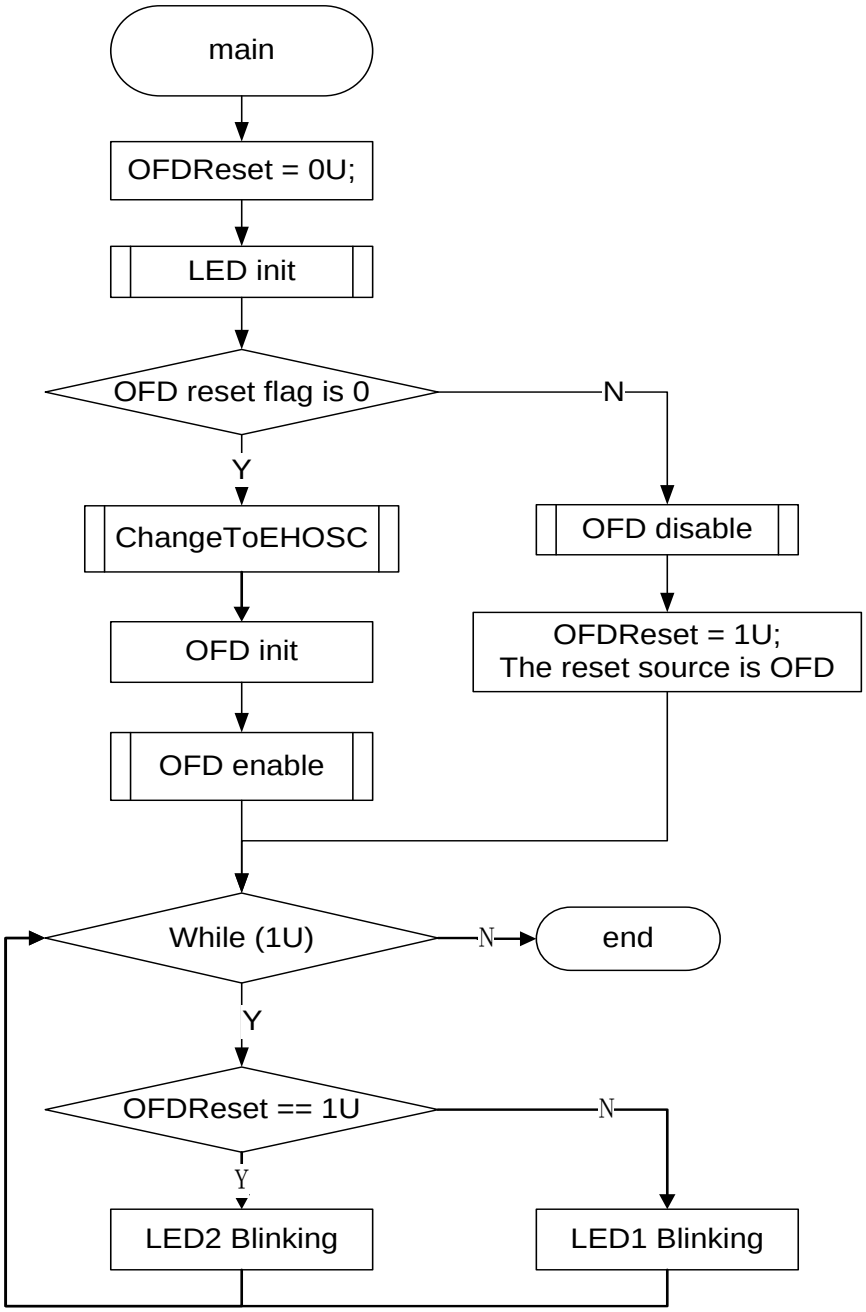
7-1 OFD

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

The example includes:

1. OFD initialization (set OFD detect frequency range)
2. OFD reset flag checking (in CG)
3. OFD enable and disable.

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize LED.

```
LEDInit();
```

Check the OFD reset flag.

```
resetFlag = CG_GetResetFlag();
if (resetFlag.Bit.OFDReset == 0U) {
  /* reset source isn't OFD */
}
```

In normal reset, for example, when the MCU is reset by power up, the OFD flag will be '0', then the program change to use the external oscillation and initialize OFD to set OFD detect frequency range.

Below code is to set up the CG to use the external oscillation.

```
void ChangeToEHOSC(void)
{
    /* Use the external oscillation */
    TSB_CG->OSCCR &= 0x000FFFFFUL;
    TSB_CG->OSCCR |= 0xFFF00000UL;                /*Set the warm up time WUODR[13:2]*/
    TSB_CG_OSCCR_EHOSCSEL = 1U;/*Select external oscillator */
    TSB_CG_OSCCR_XEN1 = 1U;/*Enable external oscillator */
    TSB_CG_OSCCR_HWUPSEL = 1U;/*Select warm-up clock */
    TSB_CG_OSCCR_WUEON = 1U;      /*Start warm up */
    while (TSB_CG_OSCCR_WUEF) {}/* Warm-up */
    TSB_CG_OSCCR_OSCSEL = 1U;     /*Use the external oscillation */
    TSB_CG_OSCCR_XEN2 = 1U;       /*Enable internal oscillator for ofd */
}
```

To configurate the OFD, set to enable to write its register at first.

```
    OFD_SetRegWriteMode(ENABLE);
```

Then set detection frequency.

```
    OFD_SetDetectionFrequency(OFD_HIGHER_COUNT,
                                                OFD_LOWER_COUNT);
```

If define DEMO2, the abnormal frequency range is set.

```
    OFD_SetDetectionFrequency(OFD_HIGHER_COUNT_ABNORMAL,
                                            OFD_LOWER_COUNT_ABNORMAL);
```

Enable OFD after initialization.

```
    OFD_Enable();
```

After above setting, OFD will be ready to detect the frequency of external clock. If the clock exceeds the detection frequency range (for example, take the external oscillator out, or make it short, or define DEMO2), OFD will generate a reset signal in the reset source register which is in CG module. Then the MCU will be reset automatically.

If the OFD reset flag is detected, MCU will run under the default inner oscillator, then OFD function will be disabled and the variable OFDReset will be set to 1.

```
    OFD_Disable();
    OFDReset = 1U;
```

LED1 and LED2 indicate the system clock status as below.

LED status	Meaning
LED1 blinking	MCU run normally with external oscillator works OK, OFD function is enabled and ready to detect oscillator abnormal status
LED2 blinking	External oscillator failure, causes the OFD reset, MCU use the default setting to run under inner oscillator. OFD function is disabled.

```
while (1U) {
    if (OFDReset == 1U) {
        LedOn (LED2);
        Delay();
        LedOff (LED2);
        Delay();
    } else {
        LedOn(LED1);
        Delay();
        LedOff (LED1);
        Delay();
    }
}
```

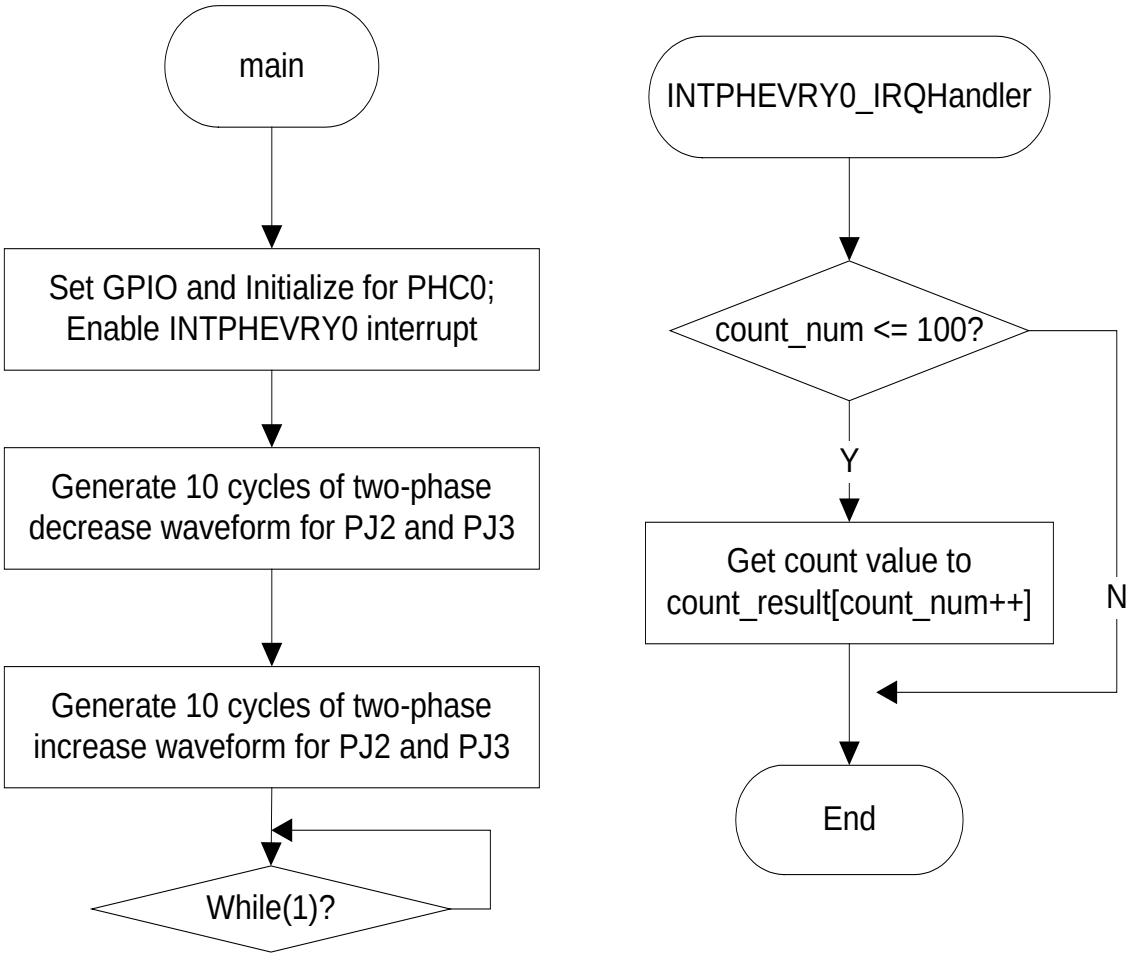
7-2 PHC

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

The example includes:

1. PHC up-and-down counter works in quadruple mode

- **Flow Chart**



- **Code and Explanation for the Example**

At first, initialize GPIO Function,

PJ0 - PHC0IN0 function, input

PJ1 - PHC0IN1 function, input

PJ2 - GPIO, output

PJ3 - GPIO, output

```
    GPIO_SetInput(GPIO_PJ, (GPIO_BIT_1 | GPIO_BIT_0));
    GPIO_SetOutput(GPIO_PJ, (GPIO_BIT_2 | GPIO_BIT_3));
    GPIO_EnableFuncReg(GPIO_PJ,     GPIO_FUNC_REG_3,     (GPIO_BIT_1     |
GPIO_BIT_0));
    GPIO_DisableFuncReg(GPIO_PJ,     GPIO_FUNC_REG_3,     (GPIO_BIT_2     |
GPIO_BIT_3));
    GPIO_SetInputEnableReg(GPIO_PJ, (GPIO_BIT_1 | GPIO_BIT_0), ENABLE);
    GPIO_SetOutputEnableReg(GPIO_PJ, (GPIO_BIT_2 | GPIO_BIT_3), ENABLE);
```

Prepare PHC initialization structure, fill PHC mode, noise filter controller and couner clear controller. This demo sets PHC quadruple mode, with noise filter and clear the counter.

```
    PHC_InitTypeDef InitStruct;
```

```
InitStruct.Mode = PHC_CR_MODE_4TIMES; /* Quadruple mode */
InitStruct.NoiseFilterCtrl = PHC_CR_NOISEFILTER_ON; /* Noise Filter on */
InitStruct.CountClearCtrl = PHC_COUNT_CLR; /* Clear counter */
```

Enable the PHC module and set the initialization structure to specified registers. Enable INTPHEVRY0 interrupt, this interrupt is triggered every counter change, in the end, start the PHC to run.

```
PHC_Enable(TSB_PHC0); /* Enable PHC Operation */
PHC_Init(TSB_PHC0, &InitStruct); /* Set InitStruct to PHC */
PHC_DisableInterrupt(TSB_PHC0, PHC_CR_INT_ALL); /* Disable all interrupt */
PHC_EnableInterrupt(TSB_PHC0,PHC_CR_INT_EVERY); /* Enable INTPHEVRY0 interrupt */
NVIC_EnableIRQ(INTPHEVRY0_IRQn);
PHC_SetRunState(TSB_PHC0, PHC_RUN); /* Run PHC0 */
```

Then the main routine will enter “While(1)” to wait the interrupt happens. In interrupt routine, use a get the count value.

```
void INTPHEVRY0_IRQHandler(void)
{
    if(count_num <= 100) {          /*Judge if overflow count_result[] or not*/
        count_result[count_num++] = PHC_GetPulseCntValue(TSB_PHC0); /* get count value*/
    } else {
        //Do nothing
    }
}
```

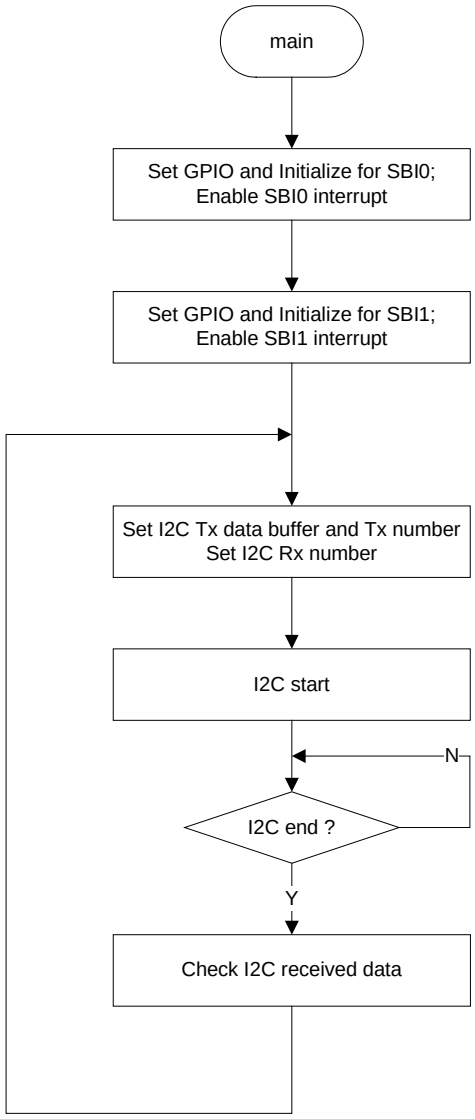
7-3 SBI

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_SBI2) is used to to start I2C process.

```
        /* Check I2C bus state and start TRx */
    case MODE_SBI_I2C_START:
        i2c_state = SBI_GetI2CState(TSB_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 INTSBI2.
The data receive is handled in INTSBI0.

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 */
    }
}
```

7-4 SIO

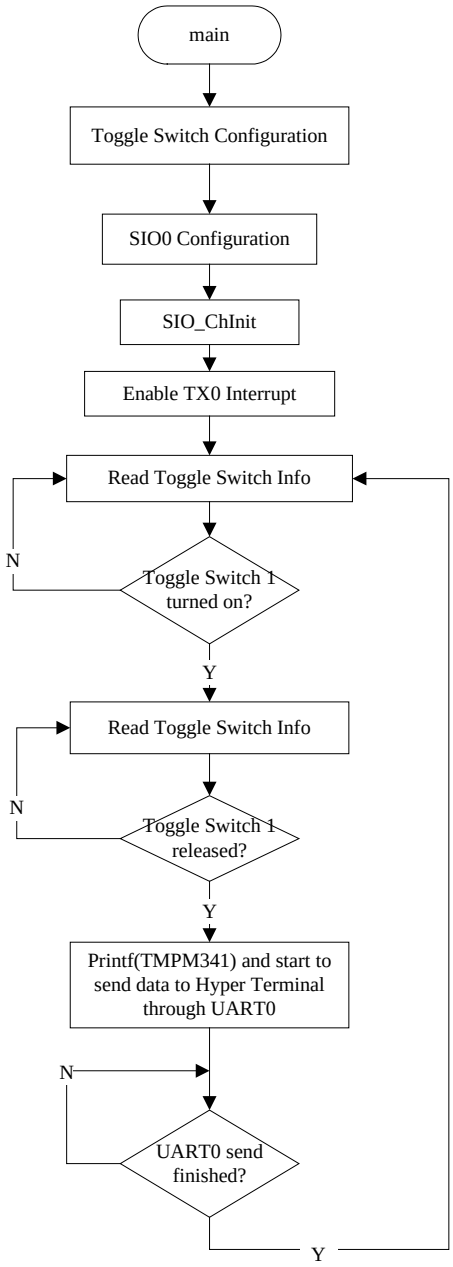
7-4-1 Example: Retarget

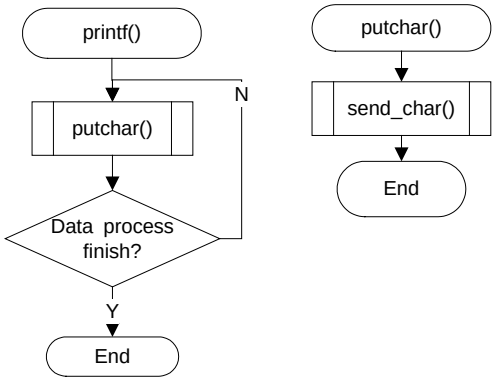
This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:

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

- **Flowchart**





- **Code and Explanation for the Example**

At first, Configure Toggle Switch and SIO0, then Initialize UART0.
Use GPIO peripheral drivers configure GPIO for Toggle Switch.

```
GPIO_SetPullUp(TSW_PORT, GPIO_BIT_0, ENABLE);
GPIO_SetPullUp(TSW_PORT, GPIO_BIT_1, ENABLE);
GPIO_SetPullUp(TSW_PORT, GPIO_BIT_2, ENABLE);
GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_0, ENABLE);
GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_1, ENABLE);
GPIO_SetInputEnableReg(TSW_PORT, GPIO_BIT_2, ENABLE);
```

Use GPIO peripheral drivers configure GPIO for UART0.

```
GPIO_SetOutputEnableReg(GPIO_PC, GPIO_BIT_4, ENABLE);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_2, GPIO_BIT_4);
GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_2, GPIO_BIT_1);
GPIO_SetInputEnableReg(GPIO_PC, GPIO_BIT_1, ENABLE);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_0, ENABLE);
GPIO_SetPullUp(GPIO_PC, GPIO_BIT_1, ENABLE);
```

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

```
UART_InitTypeDef myUART;

myUART.BaudRate = 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;
```

Use UART peripheral drivers enable and initialize UART0.

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

After above setting, enable UART0 TX interrupt.

```
NVIC_EnableIRQ(INTTX0_IRQn);
```

Then Read Toggle Switch info:

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

Then Judge whether Toggle Switch 1 has been turned on, if Toggle Switch is turned on, then wait for Toggle Switch 1 released:

```
if (TSW_info == TSW1) {
```

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

After Toggle Switch 1 has been released, start to send data by printf() through UART0.

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

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

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

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

      return ch;
}
```

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

UART0 TX interrupt routine:

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

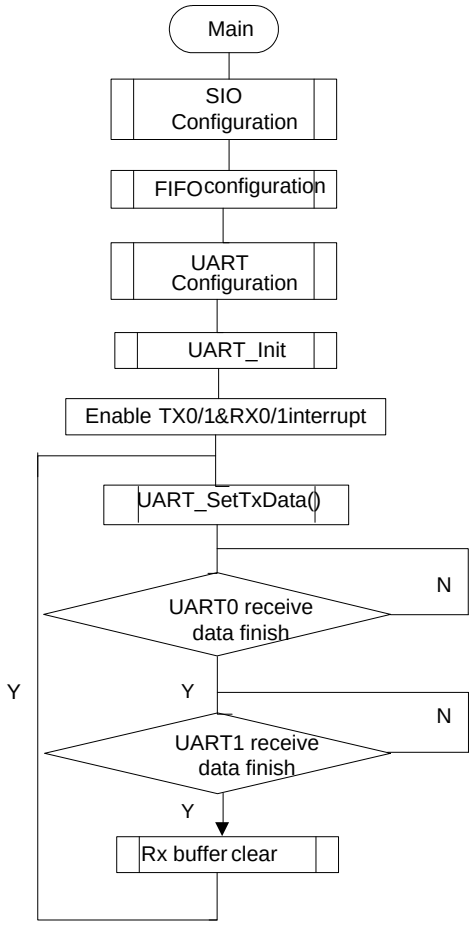
```
        }
    }
```

7-4-2 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->FR2 |= GPIO_BIT_0;
          TSB_PC->FR2 |= 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;
```

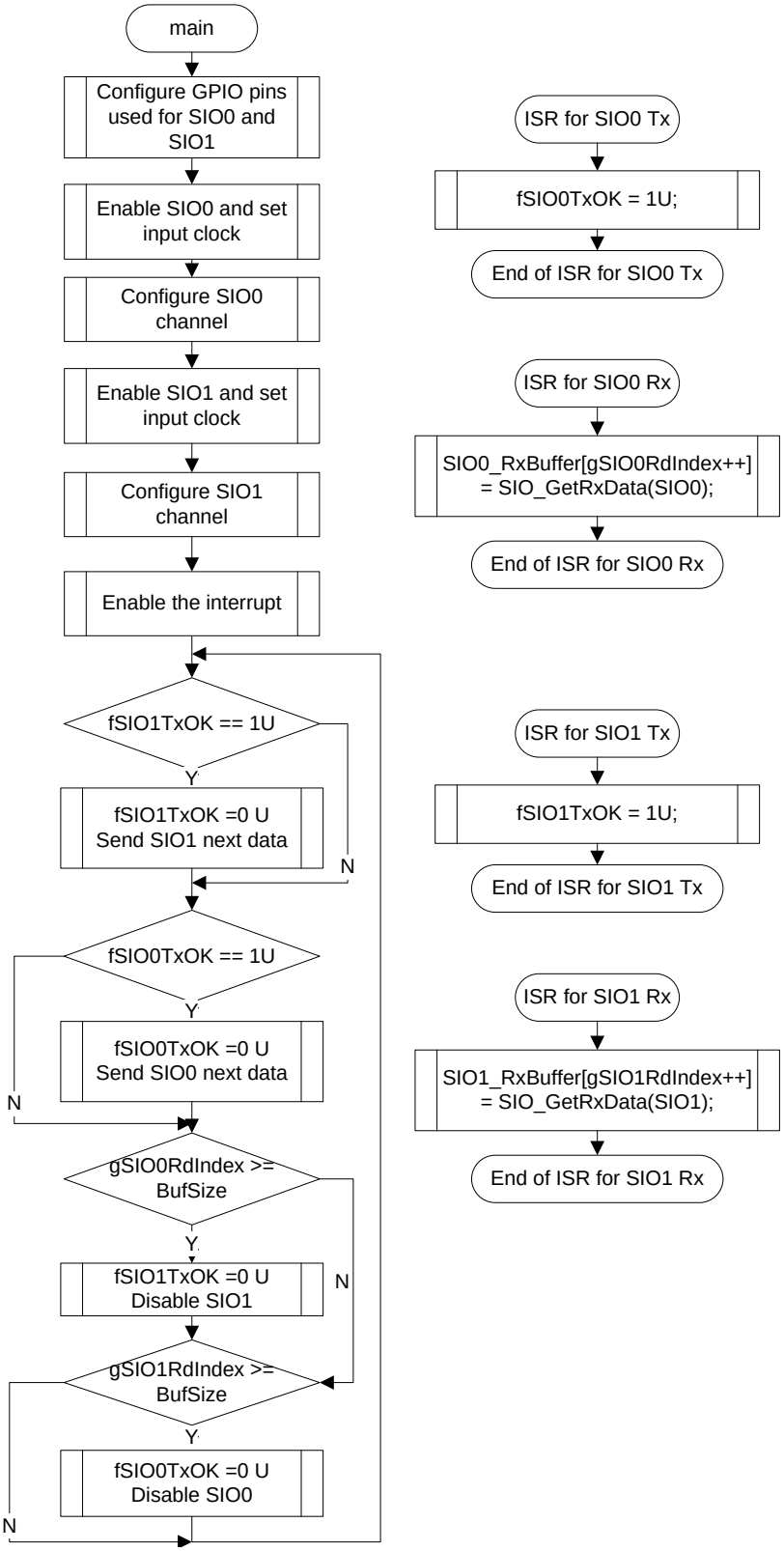
```
    err = UART_GetErrState(UART1);
    if (UART_NO_ERR == err) {
        RxBuffer1[RxCounter1++] = (uint8_t) UART_GetRxData(UART1);
    }
}
```

7-4-3 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_1;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_T4;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;

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

Enable SIO1 configure the input clock and SIO1 initialization structure.

```
/*Enable the SIO1 channel */
SIO_Enable(SIO1);

/*initialize the SIO1 struct */
SIO1_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO1_Init.IntervalTime = SIO_SINT_TIME_SCLK_1;
SIO1_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO1_Init.TransferDir = SIO_LSB_FRIST;
SIO1_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO1_Init.DoubleBuffer = SIO_WBUF_ENABLE;

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

Enable the interrupt of SIO TX, RX.

```
/* Enable SIO0 Channel TX interrupt */
NVIC_EnableIRQ(INTTX0_IRQn);
/* Enable SIO1 Channel RX interrupt */
NVIC_EnableIRQ(INTRX1_IRQn);

/* Enable SIO1 Channel TX interrupt */
NVIC_EnableIRQ(INTTX1_IRQn);
/* Enable SIO0 Channel RX interrupt */
NVIC_EnableIRQ(INTRX0_IRQn);
```

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

```
while (1) {
        /* SIO1 send data from TXD1*/
        if (fSIO1TxOK == 1U) {
            fSIO1TxOK = 0U;
            SIO_SetTxData(SIO1, SIO1_TxBuffer[gSIO1WrIndex++]);
        } else {
            /*Do Nothing */
        }
        /* SIO0 send data from TXD0*/
        if (fSIO0TxOK == 1U) {
            fSIO0TxOK = 0U;
            SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
        } else {
            /*Do Nothing */
        }
```

```
        /*SIO0 receive data end */
        if (gSIO0RdIndex >= BufSize) {
            fSIO1TxOK = 0U;
            SIO_Disable(SIO1);
        } else {
            /*Do Nothing */
        }
        /*SIO1 receive data end */
        if (gSIO1RdIndex >= BufSize) {
            fSIO0TxOK = 0U;
            SIO_Disable(SIO0);
        } else {
            /*Do Nothing */
        }    }
```

In ISR of SIO0 Tx, set transfer is ok.

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

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

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

In ISR of SIO1 Tx, set transfer is ok.

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

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

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

7-5 SSP

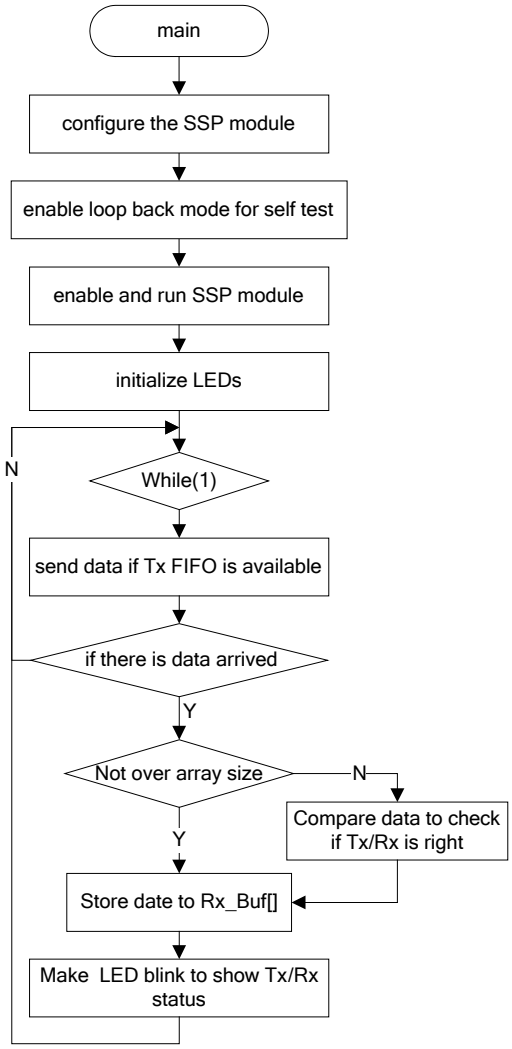
7-5-1 Example: SSP Self Loop Back

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

The example includes:

1. Configuration and initialization for SSP module
2. Enable its loop back mode to do self Tx/Rx

- **Flowchart**



- **Code and Explanation for the Example**

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

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

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

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

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

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

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

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

	/* initialize LEDs on M341-SK board before display something */
	LEDInit();

	while (1) {

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

		/* check if there is data arrived */
		fifoState = SSP_GetFIFOState(SSP_RX);
		if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
			receive = SSP_GetRxData(TSB_SSP);
			if (cntRx < MAX_BUFSIZE) {
				Rx_Buf[cntRx] = receive;
				cntRx++;
			} else {
				/* Place a break point here to check if receive data is right.
*/
				/*                    Success                    Criteria:
*/
				/*          Every data transmited from Tx_Buf is received in
Rx_Buf.	*/
				/* When the line "#define BITRATE_MIN" is commented, the SSP is
run in maxium */
				/*  bit rate, so we can find there is enough time to transmit date from
1 to	*/
				/*  MAX_BUFSIZE one by one. but if we uncomment that line, SSP is
run in	*/
				/*  minimum bit rate, we will find that receive data can't catch
```

```
"datTx++",     */
                    /*   in this so slow bit rate, when the Tx FIFO is available, the cntTx
has      */
                    /*                    been          increased          so          much.
*/
                    __NOP();
                    result = Buffercompare( Tx_Buf, Rx_Buf, MAX_BUFSIZE);
                }

            } else {
                /* do nothing */
            }

            DisplayLED(receive);
        }
}
```

7-6 TMRB

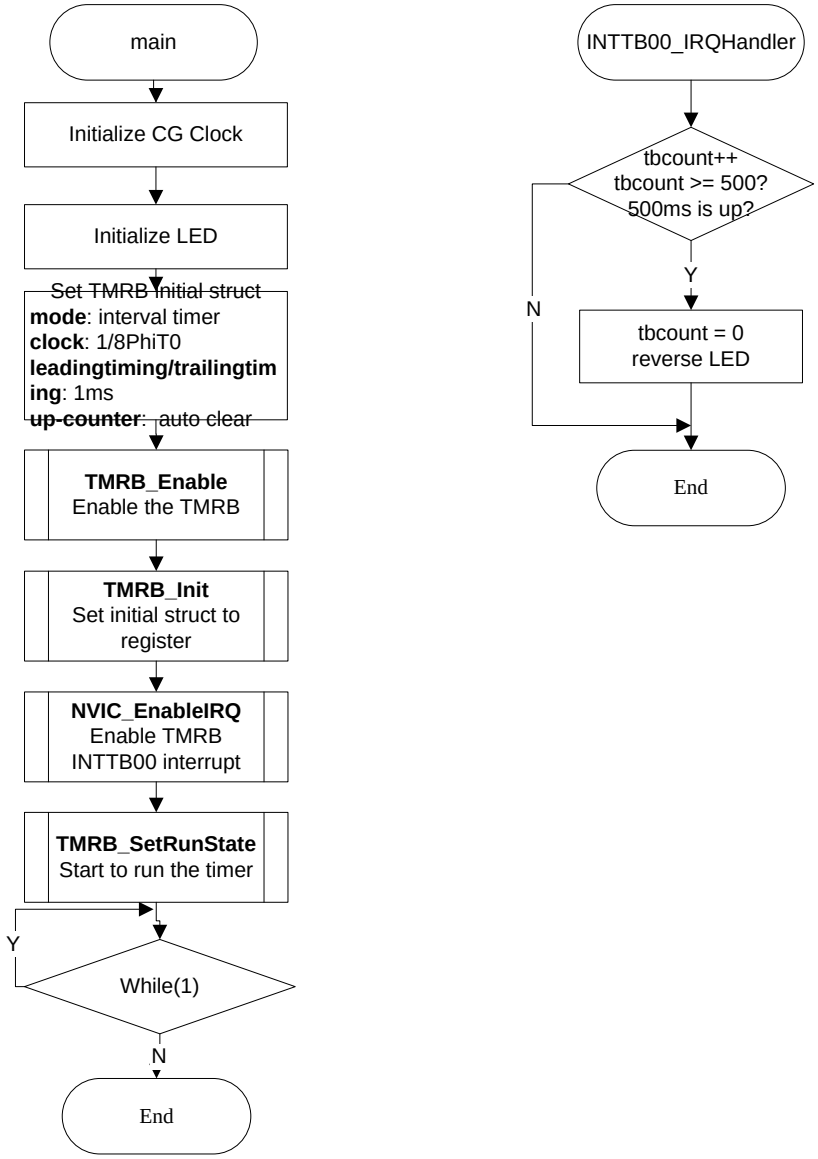
7-6-1 Example: General Timer

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

The example includes:

1. TMRB0 initialization
2. General Timer for 1ms

- **Flow Chart**



- **Code and Explanation for the Example**

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

```
CG_InitSystem();                    /* CG_SetSystem */
LEDInit();                          /* LED initialize */
LedDisable(LED0 | LED1 | LED2);
LedOn(LED3);                                    /* Turn on LED1 */
```

Prepare TMRB initialization structure, fill TMRB mode, clock, up-counter clear method, trailingtiming and leadingtiming. This demo will set trailingtiming and leadingtiming to 1ms, macro TMRB_1MS equals 0x1388, because fphiT0 = fsys = fc = 10MHz * PLL* 1/4= 40MHz, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us, 1ms/0.2us = 5000 = 0x1388 (Please see CG part for more detail information about the clock setting)

```
TMRB_InitTypeDef m_tmrb;

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

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

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

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

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

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

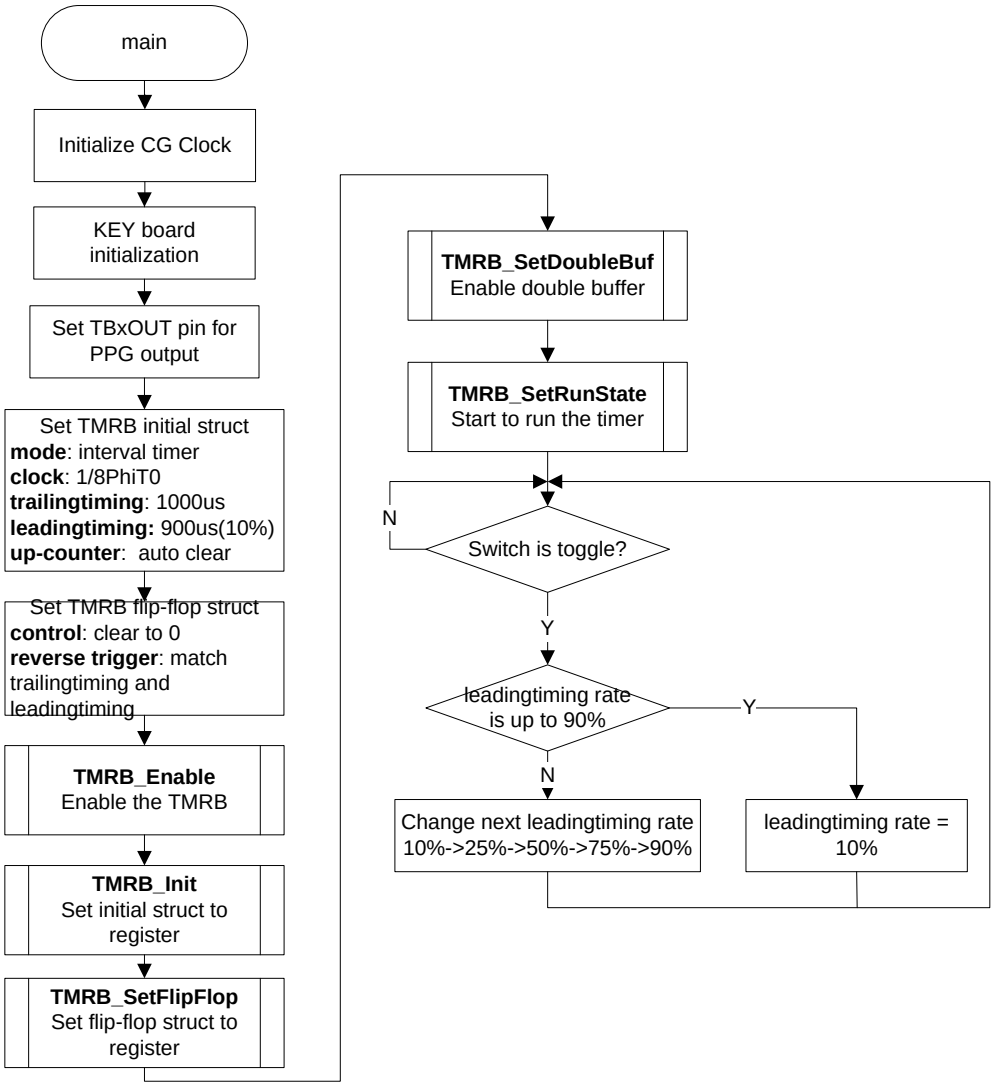
7-6-2 Example: PPG Output

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

The example includes:

1. TMRB7 initialization
2. PPG process setting and start
3. PPG leadingtiming adjustment

- **Flow Chart**



- **Code and Explanation for the Example**

At first, initialize KEY board, set PD0 as TB7OUT for PPG output.

```
    GPIO_SetInput(KEYPORT, GPIO_BIT_3);                    /* set KEY port to input */
    /* Set PD0 as TB7OUT for PPG output */
    GPIO_SetOutput(GPIO_PD, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_3, GPIO_BIT_0);
```

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 1000us, macro TMRB7TIME equals 0x09C4, because fphiT0 = fsys = fc = 10MHz * PLL * 1/4 = 20MHz, ftmrb = 1/8 fphiT0 = 2.5MHz, Ttmrb = 0.4us, 1000us/0.4us = 2500 = 0x09C4 (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 = TMRB7TIME;              /* trailingtiming is 500us
*/
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;     /* up-counter auto clear */
```

```
    m_tmrb.LeadingTiming  =  LeadingTiming[Rate];                    /*
leadingtiming, initial value 10% */
```

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

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

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

```
    TMRB_Enable(TSB_TB7);
    TMRB_Init(TSB_TB7, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB7, &PPGFFInital);
    TMRB_SetDoubleBuf(TSB_TB7, ENABLE);     /* enable double buffer */
    TMRB_SetRunState(TSB_TB7, TMRB_RUN);
```

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

```
    do {                          /* wait if switch is Low */
        keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_3);
      } while (GPIO_BIT_VALUE_0 == keyvalue);
    delay(0xFFFU);                /* noise cancel */
```

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

```
    do {
        keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_3);
      } while (GPIO_BIT_VALUE_1 == keyvalue);
    delay(0xFFFU);                /* noise cancel */

    Rate++;
    if (Rate >= LEADINGTIMINGMAX) {
      Rate = LEADINGTIMINGINIT;
      } else {
          /* Do nothing */
      }
    TMRB_ChangeLeadingTiming(TSB_TB7,  LeadingTiming[Rate]);      /* switch is
High again */
```

The calculation method of leadingtiming value:
TrailingTiming = 1ms, ftmrb = 1/8 fphiT0 = 2.5MHz, Ttmrb = 0.4us (these time parameters will be different if use different CG setting)

LeadingTiming = 10%: high-level time is 1000*10% = 100us, low-level time is 1000-100 = 900us, counter value = 900us/Ttmrb = 0x8CA
LeadingTiming = 25%: high-level time is 1000*25% = 250us, low-level time is 1000-250 = 750us, counter value =750us/Ttmrb = 0x753

LeadingTiming = 50%: high-level time is 1000*50% = 500us, low-level time is 1000-500 = 500us, counter value = 500us/Ttmrb = 0x4E2

LeadingTiming = 75%: high-level time is 1000*75% = 750us, low-level time is 1000-750 = 250us, counter value = 250us/Ttmrb = 0x271

LeadingTiming = 90%: high-level time is 1000*90% = 900us, low-level time is 1000-900 = 100us, counter value = 100us/Ttmrb = 0xFA

That is the calculation method of leadingtiming array

```
uint32_t LeadingTiming[5] = { 0x8CAU, 0x753U, 0x4E2U, 0x271U, 0xFAU };
/* leadingtiming: 10%, 25%, 50%, 75%, 90% */
```

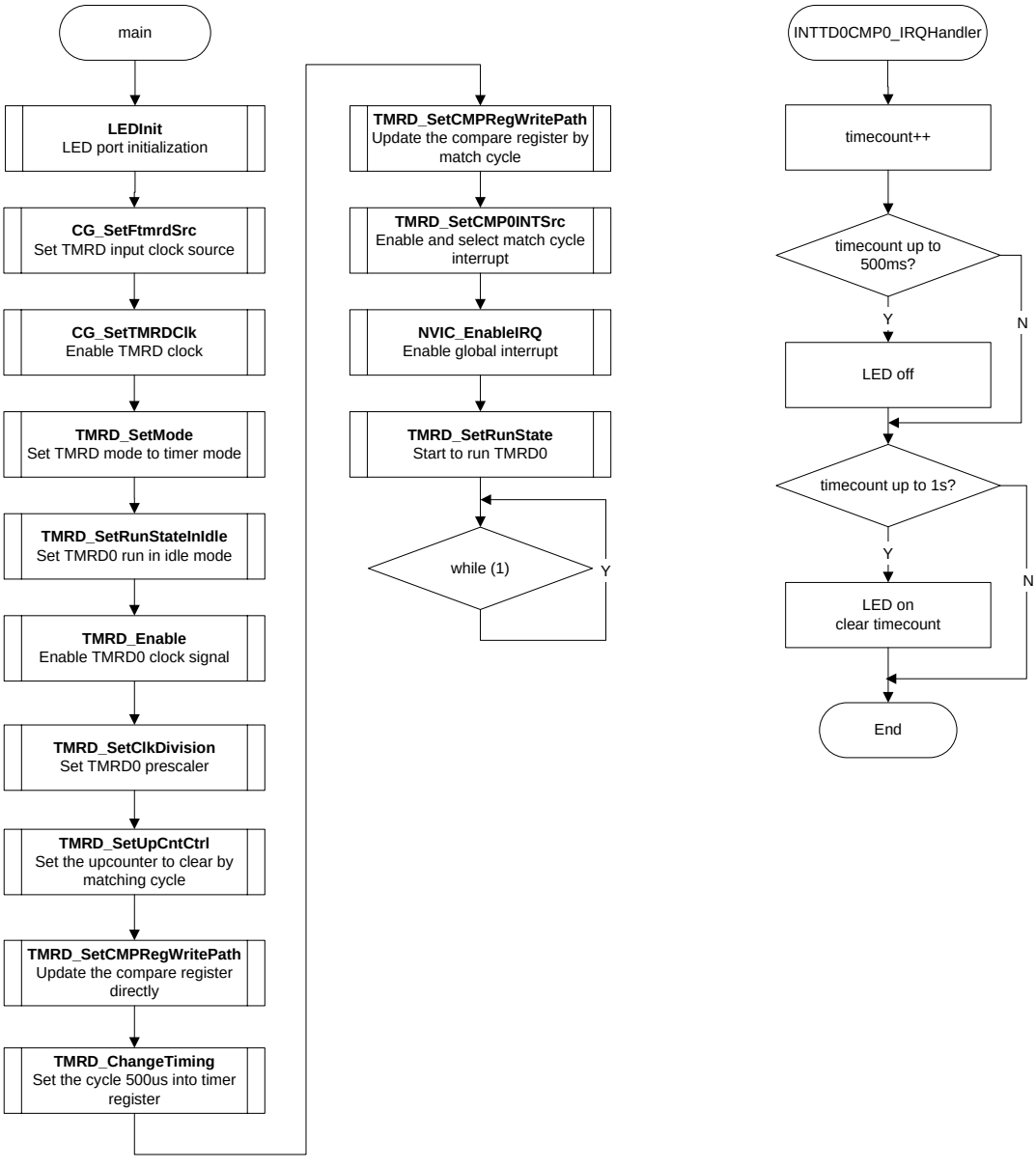
7-7 TMRD

7-7-1 Interval timer

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

The example includes:
4. TMRD interval timer mode.
5. TMRD initialization and operation process.
6. TMRD interrupt handling.

- **Flow Chart**



- **Code and Explanation for the Example**

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

```
LEDInit();
```

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

```
/* TMRD CG setting */
CG_SetFtmrdSrc(CG_FTMRD_SRC_FPLL);        /* clock is set to fpll */
CG_SetTMRDClk(ENABLE);                    /* enable the TMRDCLK */
```

Set TMRD mode by using TMRD_SetMode(). In this demo, both TMRD0 and TMRD1 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(TSB_TD0, TMRD_RUN);
```

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

```
/* TMRD enable clock signal */
TMRD_Enable(TSB_TD0);

/* Set the TMRD prescaler */
TMRD_SetClkDivision(TSB_TD0, TMRD_CLK_DIV_1);
```

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

```
/* Set the upcounter clear mode */
TMRD_SetUpCntCtrl(TSB_TD0, 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 = 10MHz(external high-speed oscillator)
fpll = 8 * fosc = 80MHz(in this demo, PLL multiplying value is set to 8 multiplying)
ftmrd = fpll = 80MHz(in this demo, ftmrd = fpll, please see CG_SetFtmrdSrc())
So the value of TIME_500US is 500us*ftmrd = 40000 = 0x9C40

```
/* Update the compare register directly */
TMRD_SetCMPRegWritePath(TSB_TD0, 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(TSB_TD0, 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(TSB_TD0, TMRD_INT_MATCH_CYCLE);
```

```
    NVIC_EnableIRQ(INTTD0CMP0_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(TSB_TD0, TMRD_RUN);
```

When 500us is up, the compare 0 interrupt occurs. In IRQ INTTD0CMP0_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.

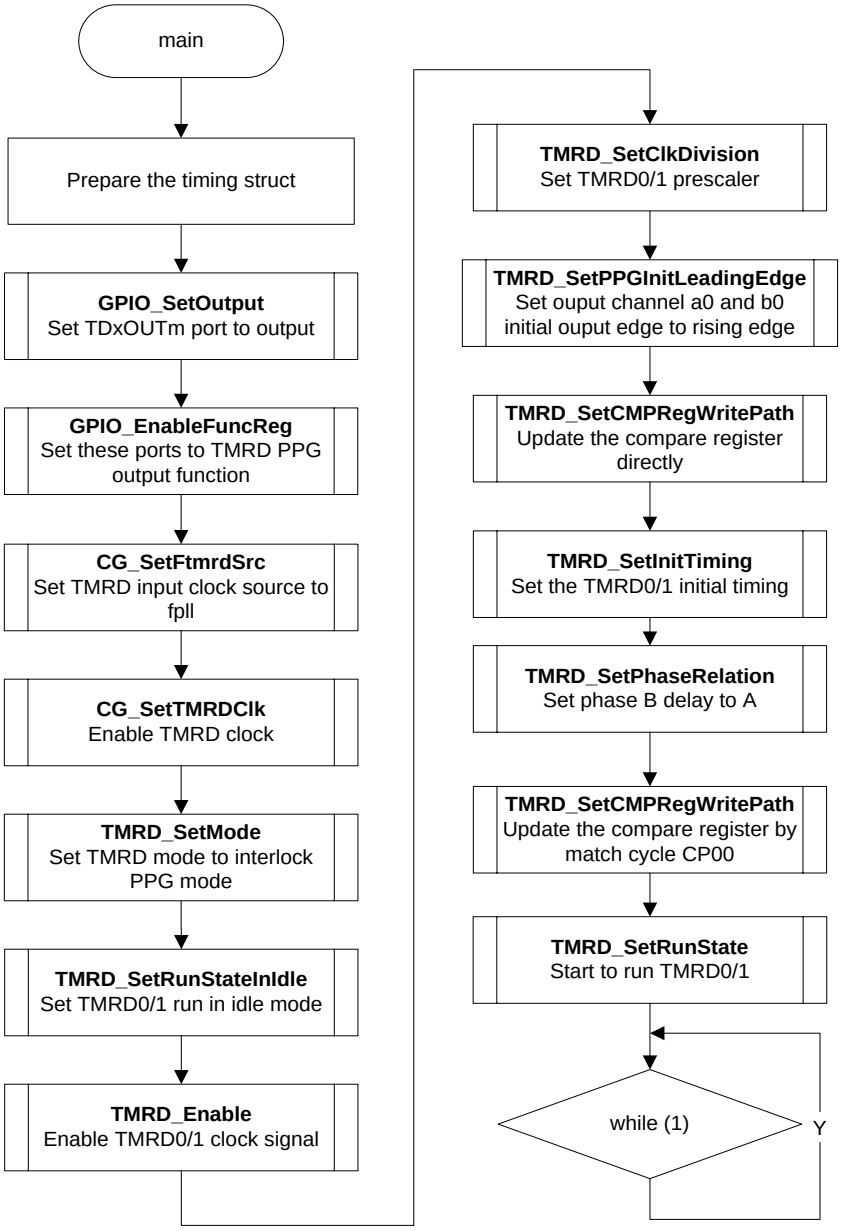
```
void INTTD0CMP0_IRQHandler(void)
{
    static uint16_t timecount = 0U;

    timecount++;
    if (timecount == 1000U) {     /* 500ms */
        LedOff(0x00U);
    } else if (timecount == 2000U) {       /* 1s */
        timecount = 0U;
        LedOn(0x02U);
    } else {
        /* Do nothing */
    }
}
```

7-7-2 Interlock PPG output

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

The example includes:
1. TMRD interlock PPG mode.
2. TMRD initialization and operation process.
3. TMRD PPG mode parameters calculation method.

- **Flow Chart**


main
Prepare the timing struct
GPIO_SetOutput
Set TDxOUTm port to output
GPIO_EnableFuncReg
Set these ports to TMRD PPG output function
CG_SetFtmrdSrc
Set TMRD input clock source to fpll
CG_SetTMRDClk
Enable TMRD clock
TMRD_SetMode
Set TMRD mode to interlock PPG mode
TMRD_SetRunStateInIdle
Set TMRD0/1 run in idle mode
TMRD_Enable
Enable TMRD0/1 clock signal
TMRD_SetClkDivision
Set TMRD0/1 prescaler
TMRD_SetPPGInitLeadingEdge
Set ouput channel a0 and b0 initial ouput edge to rising edge
TMRD_SetCMPRegWritePath
Update the compare register directly
TMRD_SetInitTiming
Set the TMRD0/1 initial timing
TMRD_SetPhaseRelation
Set phase B delay to A
TMRD_SetCMPRegWritePath
Update the compare register by match cycle CP00
TMRD_SetRunState
Start to run TMRD0/1
while (1)
Y

- **Code and Explanation for the Example**

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

Cycle is 500us:
fosc = 10MHz(external high-speed oscillator)
fpll = 8 * fosc = 80MHz(in this demo, PLL multiplying value is set to 8 multiplying)
ftmrd = fpll = 80MHz(in this demo, ftmrd = fpll, please see CG_SetFtmrdSrc())
So the value is 500us*ftmrd = 40000 = 0x9C40

Duty is 50%:
Set LeadingTiming0 to 0x0000, then TrailingTiming0 is Cycle/2 = 0x4E20(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 = 0x3415(167us)

```
#define TIME_CYCLE              ((uint16_t)0x9C40U)     /* 500us */
#define TIME_DUTY               ((uint16_t)0x4E20U)     /* 250us */
#define TIME_PHASE_SHIFT        ((uint16_t)0x3415U)     /* 500/3= 167us */
…
TMRD_TimingTypeDef timestruct = { 0U };

timestruct.Cycle = TIME_CYCLE;
timestruct.TrailingTiming0 = TIME_DUTY;
timestruct.PhaseShiftTiming = TIME_PHASE_SHIFT;
```

Configure the TDxOUTm port to TMRD PPG output. In this demo, these ports are port E 4,5,6,7.

```
/* TD0OUT0, TD0OUT1, TD1OUT0, TD1OUT1 port setting */
GPIO_SetOutput(GPIO_PE, GPIO_BIT_4 | GPIO_BIT_5 | GPIO_BIT_6 |
                    GPIO_BIT_7);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_3, GPIO_BIT_4 |
                        GPIO_BIT_5 | GPIO_BIT_6 | GPIO_BIT_7);
```

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

```
/* TMRD CG setting */
CG_SetFtmrdSrc(CG_FTMRD_SRC_FPLL);      /* clock is set to fpll */
CG_SetTMRDClk(ENABLE);                  /* enable the TMRDCLK */
```

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

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

Set TMRD0/1 to run in idle mode by using API TMRD_SetRunStateInIdle().

```
/* TMRD IDLE mode */
TMRD_SetRunStateInIdle(TSB_TD0, TMRD_RUN);
TMRD_SetRunStateInIdle(TSB_TD1, 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(TSB_TD0);
TMRD_Enable(TSB_TD1);
```

```
    /* Set the TMRD prescaler */
    TMRD_SetClkDivision(TSB_TD0, 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_A0,
                               TMRD_WAVE_EDGE_RISING);
    TMRD_SetPPGInitLeadingEdge(TMRD_PPG_CHANNEL_B0,
                               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(TSB_TD0, TMRD_CMP_WRITE_DIRECT);
    TMRD_SetCMPRegWritePath(TSB_TD1, TMRD_CMP_WRITE_DIRECT);

    /* Set the value to timer register */
    TMRD_SetInitTiming(TSB_TD0, &timestruct);
    TMRD_SetInitTiming(TSB_TD1, &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(TSB_TD0, TMRD_CMP_WRITE_INDIRECT);
    TMRD_SetCMPRegWritePath(TSB_TD1, 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 TMRD0, 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(TSB_TD0, TMRD_RUN);
```

7-8 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();
```