King Mongkut's University of Technology Thonburi

Department of Electronics and Telecommunication Engineering Faculty of Engineering

EIE/ENE 335 Digital Circuit and Microprocessor Lab for the 3rd year student

Experiment: Interrupt, Timer, and WDT

Objectives

- How to use
  - the NuMicro™ NUC100 series driver to do the fast application software development
  - Interrupt
  - Timer
  - WDT

Background Theory

Cortex™-M0

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Interrupt Name	Source IP	Interrupt description
0 ~ 15	-	-	-	System exceptions
16	0	**BOD_OUT**	Brown-Out	Brown-Out low voltage detected interrupt
17	1	**WDT_INT**	WDT	Watchdog Timer interrupt
18	2	**EINT0**	GPIO	External signal interrupt from PB.14 pin
19	3	**EINT1**	GPIO	External signal interrupt from PB.15 pin
20	4	**GPAB_INT**	GPIO	External signal interrupt from PA[15:0]/PB[13:0]
21	5	**GPCDE_INT**	GPIO	External interrupt from PC[15:0]/PD[15:0]/PE[15:0]
22	6	**PWMA_INT**	PWM0~3	PWM0, PWM1, PWM2 and PWM3 interrupt
23	7	**PWMB_INT**	PWM4~7	PWM4, PWM5, PWM6 and PWM7 interrupt
24	8	**TMR0_INT**	TMR0	Timer 0 interrupt
25	9	**TMR1_INT**	TMR1	Timer 1 interrupt

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Interrupt Name	Source IP	Interrupt description
26	10	**TMR2_INT**	TMR2	Timer 2 interrupt
27	11	**TMR3_INT**	TMR3	Timer 3 interrupt
28	12	**UART02_INT**	UART0/2	UART0 and UART2 interrupt
29	13	**UART1_INT**	UART1	UART1 interrupt
30	14	**SPI0_INT**	SPI0	SPI0 interrupt
31	15	**SPI1_INT**	SPI1	SPI1 interrupt
32	16	**SPI2_INT**	SPI2	SPI2 interrupt
33	17	**SPI3_INT**	SPI3	SPI3 interrupt
34	18	**I2C0_INT**	I^2C0	I^2C0 interrupt
35	19	**I2C1_INT**	I^2C1	I^2C1 interrupt
36	20	**CAN0_INT**	CAN0	CAN0 interrupt
37	21	**Reserved**	Reserved	Reserved
38	22	**Reserved**	Reserved	Reserved
39	23	**USB_INT**	USBD	USB 2.0 FS Device interrupt
40	24	**PS2_INT**	PS/2	PS/2 interrupt
41	25	**ACMP_INT**	ACMP	Analog Comparator-0 or Comaprator-1 interrupt
42	26	**PDMA_INT**	PDMA	PDMA interrupt
43	27	**I2S_INT**	I^2S	I^2S interrupt
44	28	**PWRWU_INT**	CLKC	Clock controller interrupt for chip wake-up from power down state
45	29	**ADC_INT**	ADC	ADC interrupt
46	30	**Reserved**	Reserved	Reserved
47	31	**RTC_INT**	RTC	Real time clock interrupt

System Interrupt Map

Cortex-M0 status and operating mode control are managed by System Control Registers. Including CPUID, Cortex-M0 interrupt priority and Cortex-M0 power management can be controlled through these system control register

For more detailed information, please refer to the documents “ARM® Cortex™-M0 Technical Reference Manual” and “ARM® v6-M Architecture Reference Manual”.

Register/Instruction / Mode	PWR_DOWN_EN	PD_WAIT_CPU	CPU run WFI instruction	Clock Disable
Normal operation	0	0	NO	All clocks are disabled by control register
Idle mode (CPU entry sleep mode)	0	0	YES	Only CPU clock is disabled
Power down mode	1	0	NO	Most clocks are disabled except 10 kHz/32.768 kHz, only RTC/WDT/Timer/PWM peripheral clock are still enabled.
Power down mode (CPU entry deep sleep mode)	1	1	YES	Most clocks are disabled except 10 kHz/32.768 kHz, only RTC/WDT/Timer/PWM peripheral clock are still enabled.

Power down Mode Control Table

Equipment required

- Nu_LB-002 (Nuvoton learning board)

Reference:

1. Nu_LB-002 Rev 2.1 User's Manual
2. NuMicro™ NUC130_140 Technical Reference Manual EN V2.02
3. NuMicro™ NUC100 Series Driver Reference Guide V1.05.002

Procedure 1: interrupt

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**Interrupt**' lab file.
2. Compile the project, and run the program.
3. Study the program and answer the following questions.

```
16  #define DELAY300ms  300000    // The maximal delay time is 335000 us.
17
18  void EINT1Callback(void) {
19    clr_all_pannal();
20    print_lcd(0, "Int1 !!!!");
21    DrvSYS_Delay(DELAY300ms);   // delay
22    }
23
24  void Delay(uint32_t counter) {
25    while(counter--);
26    }
27
28  int main (void) {
29    Initial_pannel();  //call initial pannel function
30    clr_all_pannal();
31
32    /* Configure general GPIO interrupt */
33    DrvGPIO_Open(E_GPB, 15, E_IO_INPUT);
34
35    /* Configure external interrupt */
36    DrvGPIO_EnableEINT1(E_IO_BOTH_EDGE, E_MODE_EDGE, EINT1Callback);
37
38    /* Waiting for interrupts */
39    while(1) {
40      print_lcd(0, "Deep Sleep");
41      DrvSYS_Delay(DELAY300ms);    // delay
42
43      UNLOCKREG();
44      SCB->SCR = 4;      // System Control Register (5.2.8: Tech.Ref.)
45      SYSCLK->PWRCON.PD_WU_INT_EN = 0;
46      SYSCLK->PWRCON.PD_WAIT_CPU = 1;
47      SYSCLK->PWRCON.PWR_DOWN_EN = 1;
48      LOCKREG();
49      __WFI();
50
51      // check if SW_INT is pressed
52      while((GPIOB->PIN &= 0x8000) == 0) {
53        GPIOA->DOUT &= 0x8FFF;  // turn on only RGB_LED GPA_12,13, and 14
54        Delay(DELAY300ms);
55        GPIOA->DOUT |= 0x7000;  // turn off only RGB_LED GPA_12,13, and 14
56        Delay(DELAY300ms);
57       }
58      }
59    }
```

Questions (interrupt)

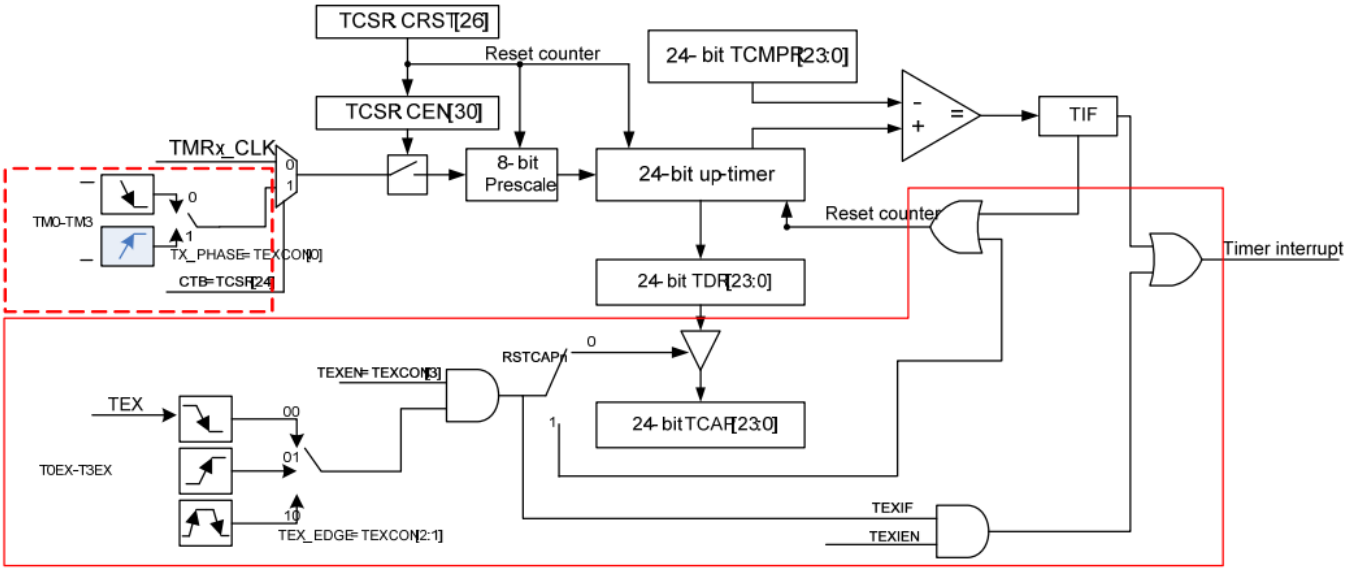
1. What does the code on line 44 (SCB->SCR = 4;) do?

2. What does the code on line 45 (SYSCLK->PWRCON.PD_WU_INT_EN = 0;) do?

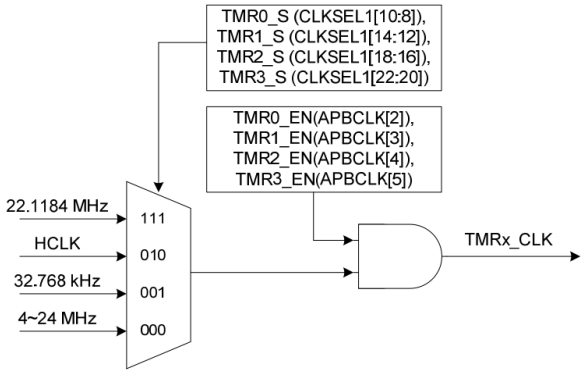
Timer Controller (TMR)

The timer controller includes four 32-bit timers, TIMER0~TIMER3, which allows user to easily implement a timer control for applications. The timer can perform functions like frequency measurement, event counting, interval measurement, clock generation, delay timing, and so on. The timer can generate an interrupt signal upon timeout, or provide the current value during operation.

- 4 sets of 32-bit timers with 24-bit up-timer and one 8-bit pre-scale
- Provides one-shot, periodic, toggle and continuous counting operation modes
- Time out period = (Period of timer clock input) * (8-bit pre-scale counter + 1) * (24-bit TCMP)
- Maximum counting cycle time = (1 / T MHz) * (2^8) * (2^{24}), T is the period of timer clock
- 24-bit timer value is readable through TDR (Timer Data Register)
- Support event counting function to count the event from external pin
- Support input capture function to capture or reset counter value



Timer Controller Block Diagram



Clock Source of Timer Controller

Procedure 2: Timer

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**Timer**' lab file.
2. Compile the project, and run the program.
3. Study the program and answer the following questions.

```
#include <stdio.h>
#include <string.h>
#include "NUC1xx.h"
#include "LCD_Driver.h"

static uint16_t Timer0Counter=0;

void InitTIMER0(void) {
  /* Step 1. Enable and Select Timer clock source */
  SYSCLK->CLKSEL1.TMR0_S = 0; // Select 12Mhz for Timer0 clock source
  // 0 = 12 MHz, 1 = 32 kHz, 2 = HCLK, 7 = 22.1184 MHz
  SYSCLK->APBCLK.TMR0_EN = 1; // Enable Timer0 clock source

  /* Step 2. Select Operation mode */
  TIMER0->TCSR.MODE = 1;      // 1 -> Select periodic mode
  // 0 = One shot, 1 = Periodic, 2 = Toggle, 3 = continuous counting mode

  /* Step 3. Select Time out period
  = (Period of timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP)*/
  TIMER0->TCSR.PRESCALE = 11; // Set Prescale [0~255]
  TIMER0->TCMPR = 1000000;    // Set TCMPR [0~16777215]
  // (1/12000000)*(11+1)*(1000000)= 1 sec or 1 Hz

  /* Step 4. Enable interrupt */
  TIMER0->TCSR.IE = 1;
  TIMER0->TISR.TIF = 1;       // Write 1 to clear the interrupt flag
  NVIC_EnableIRQ(TMR0_IRQn);  // Enable Timer0 Interrupt

  /* Step 5. Enable Timer module */
  TIMER0->TCSR.CRST = 1;      // Reset up counter
  TIMER0->TCSR.CEN = 1;       // Enable Timer0
  }

void TMR0_IRQHandler(void)    // Timer0 interrupt subroutine
{
  char lcd2_buffer[18] = "Timer0:";
  Timer0Counter += 1;
  sprintf(lcd2_buffer+7, " %d s.", Timer0Counter);
  print_lcd(2, lcd2_buffer);
  TIMER0->TISR.TIF = 1;       // Write 1 to clear the interrupt flag
}

int32_t main (void) {
  UNLOCKREG();
  SYSCLK->PWRCON.XTL12M_EN = 1;
  SYSCLK->CLKSEL0.HCLK_S = 0;
  LOCKREG();

  Initial_pannel();  //call initial pannel function
  clr_all_pannal();

  InitTIMER0();

  while(1) {
    __NOP();
    }
  }
```

Questions (Timer)

1. On line 19, what is the memory location of the ‘SYSCLK->CLKSEL1’?

2. How many bit(s) are there for the TMR0_S on line 19?

3. How can we use the Timer to measure frequency?

4. If we decide to use 22.1184 MHz instead of 12 MHz, to get 1 s. time out period what is the new values of pre-scale and TCMP?

5. What does the code on line 60 (SYSCLK->CLKSEL0.HCLK_S = 0;) do?

Procedure 3: Timer (toggle)

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**TimerToggle**' lab file.
2. Compile the project, and run the program.
3. Study the program and answer the following questions.

```
15  static uint16_t Timer0Counter=0;
16
17  //-------------------------------------------TIMER
18  void InitTIMER0(void) {
19    /* Step 0. GPIO initial */
20    SYS->GPBMFP.TM0 = 1;          // System Manager Control Registers
21
22    /* Step 1. Enable and Select Timer clock source */
23    SYSCLK->CLKSEL1.TMR0_S = 0; // Select 12Mhz for Timer0 clock source
24    // 0 = 12 MHz, 1 = 32 kHz, 2 = HCLK, 7 = 22.1184 MHz
25    SYSCLK->APBCLK.TMR0_EN = 1; // Enable Timer0 clock source
26
27    /* Step 2. Select Operation mode */
28    TIMER0->TCSR.MODE = 2;        // 2 -> Select Toggle mode
29    // 0 = One shot, 1 = Periodic, 2 = Toggle, 3 = continuous counting mode
30
31    /* Step 3. Select Time out period
32    = (Period of timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP)*/
33    TIMER0->TCSR.PRESCALE = 11; // Set Prescale [0~255]
34    TIMER0->TCMPR = 1000000;     // Set TCMPR [0~16777215]
35    // (1/12000000)*(11+1)*(1000000)= 1 sec or 1 Hz
36
37    /* Step 4. Enable interrupt */
38    TIMER0->TCSR.IE = 1;
39    TIMER0->TISR.TIF = 1;         // Write 1 to clear the interrupt flag
40    NVIC_EnableIRQ(TMR0_IRQn);  // Enable Timer0 Interrupt
41
42    /* Step 5. Enable Timer module */
43    TIMER0->TCSR.CRST = 1;        // Reset up counter
44    TIMER0->TCSR.CEN = 1;         // Enable Timer0
45    }
46
47  void TMR0_IRQHandler(void) {  // Timer0 interrupt subroutine
48    char lcd2_buffer[18] = "Timer0:";
49    Timer0Counter += 1;
50    sprintf(lcd2_buffer+7, " %d s.", Timer0Counter);
51    print_lcd(2, lcd2_buffer);
52    TIMER0->TISR.TIF = 1;         // Write 1 to clear the interrupt flag
53    }
54
55  int32_t main (void) {
56    UNLOCKREG();
57    SYSCLK->PWRCON.XTL12M_EN = 1;
58    SYSCLK->CLKSEL0.HCLK_S = 0;
59    LOCKREG();
60
61    Initial_pannel();  //call initial pannel function
62    clr_all_pannal();
63
64    InitTIMER0();
65
66    while(1) {
67      __NOP();
68      }
69    }
```

Questions (Timer (toggle))

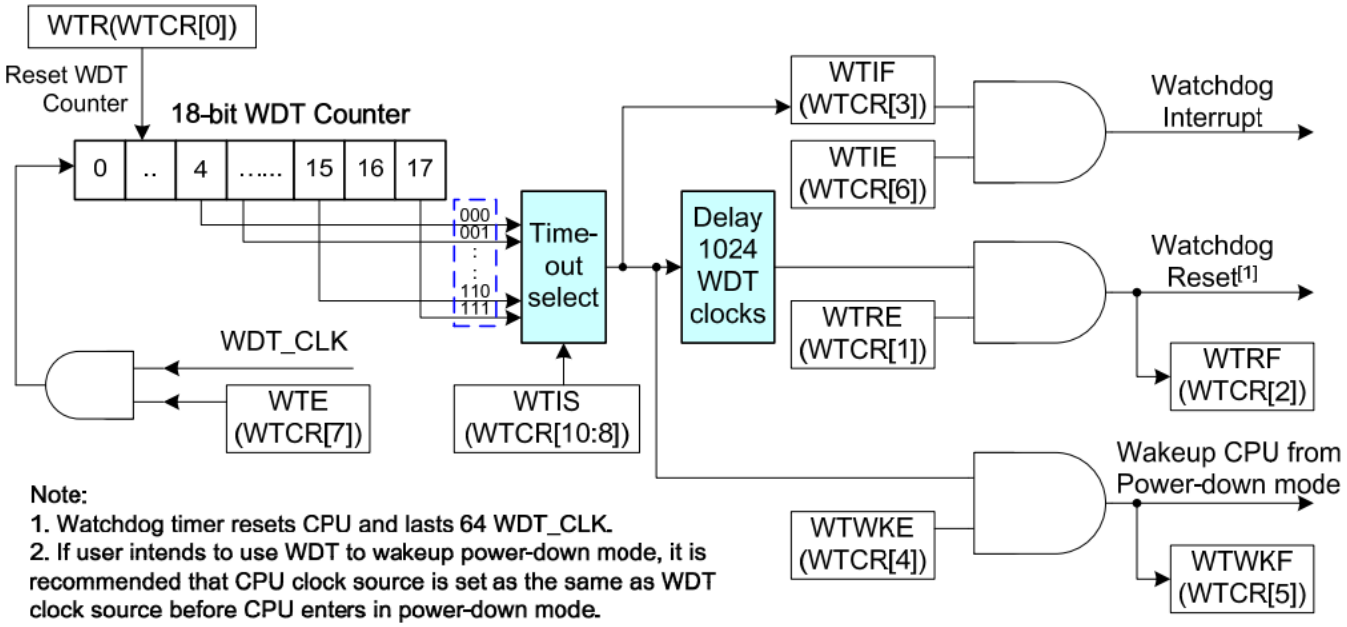
1. If we decide to use Timer1 in steads of Timer0, what is the new code on line 20? And which GPIO pin for the toggle output?

2. Draw the toggle output signal:

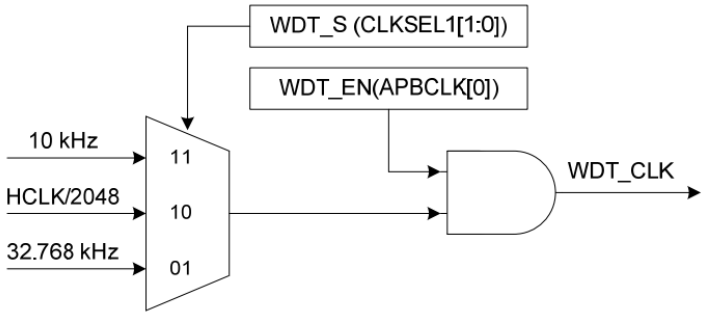
WDT

The purpose of Watchdog Timer is to perform a system reset when system runs into an unknown state. This prevents system from hanging for an infinite period of time. Besides, this Watchdog Timer supports another function to wake-up chip from power down mode. The watchdog timer includes an 18-bit free running counter with programmable time-out intervals.

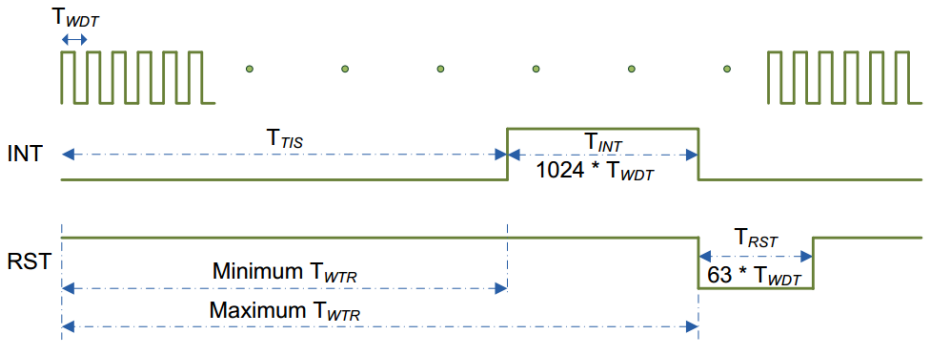
- 18-bit free running counter to avoid chip from Watchdog timer reset before the delay time expires.
- Selectable time-out interval (2^4 ~ 2^18) and the time out interval is 104 ms ~ 26.3168 s (if WDT_CLK = 10 kHz).
- Reset period = (1 / 10 kHz) * 63, if WDT_CLK = 10 kHz.



Watchdog Timer Block Diagram



Watchdog Timer Clock Control



- T_{WDT} : *Watchdog Engine Clock Time Period*
- T_{TIS} : *Watchdog Timeout Interval Selection Period*
- T_{INT} : *Watchdog Interrupt Period*
- T_{RST} : *Watchdog Reset Period*
- T_{WTR} : *Watchdog Timeout Interval Period*

Timing of Interrupt and Reset Signal

WTIS	Timeout Interval Selection T_{TIS}	Interrupt Period T_{INT}	WTR Timeout startingInterval (WDT_CLK=10 kHz) MIN. T_{WTR} ~ Max. T_{WTR}
000	$2^4 * T_{WDT}$	$1024 * T_{WDT}$	1.6 ms ~ 104 ms
001	$2^6 * T_{WDT}$	$1024 * T_{WDT}$	6.4 ms ~ 108.8 ms
010	$2^8 * T_{WDT}$	$1024 * T_{WDT}$	25.6 ms ~ 128 ms
011	$2^{10} * T_{WDT}$	$1024 * T_{WDT}$	102.4 ms ~ 204.8 ms
100	$2^{12} * T_{WDT}$	$1024 * T_{WDT}$	409.6 ms ~ 512 ms
101	$2^{14} * T_{WDT}$	$1024 * T_{WDT}$	1.6384 s ~ 1.7408 s
110	$2^{16} * T_{WDT}$	$1024 * T_{WDT}$	6.5536 s ~ 6.656 s
111	$2^{18} * T_{WDT}$	$1024 * T_{WDT}$	26.2144 s ~ 26.3168 s

Procedure 4: WDT

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**WDT**' lab file.
2. Compile the project, and run the program.
3. Study the program, answer the following question, and do the assignment in the class.

```
15  static uint16_t Timer0Counter = 0;
16
17  //---------------------------------------------------------------------------WDT
18  void InitWDT(void) {
19    UNLOCKREG();
20    /* Step 1. Enable and Select WDT clock source */
21    SYSCLK->CLKSEL1.WDT_S = 3;  // Select 10kHz for WDT clock source
22    SYSCLK->APBCLK.WDT_EN = 1;  // Enable WDT clock source
23
24    /* Step 2. Select Timeout Interval */
25    WDT->WTCR.WTIS = 6;         // 2^16 * (1/10k) = 6.5536 sec.
26
27    /* Step 3. Disable Watchdog Timer Reset function */
28    WDT->WTCR.WTRE = 0;
29
30    /* Step 4. Enable WDT interrupt */
31    WDT->WTCR.WTIF = 1;         // Write 1 to clear flag
32    WDT->WTCR.WTIE = 1;
33    NVIC_EnableIRQ(WDT_IRQn);
34
35    /* Step 5. Enable WDT module */
36    WDT->WTCR.WTE = 1;          // Enable WDT
37    WDT->WTCR.WTR = 1;          // Clear WDT counter
38    LOCKREG();
39    }
40
41  void WDT_IRQHandler(void) {
42    UNLOCKREG();
43    WDT->WTCR.WTIF = 1;
44    WDT->WTCR.WTR = 1;
45    UNLOCKREG();
46    print_lcd(3, "WDT interrupt");
47    }
```

```
49  //------------------------------------------------------------------------TIMER
50  void InitTIMER0(void) {
51    /* Step 0. GPIO initial */
52    SYS->GPBMFP.TM0 = 1;         // System Manager Control Registers
53
54    /* Step 1. Enable and Select Timer clock source */
55    SYSCLK->CLKSEL1.TMR0_S = 0; // Select 12Mhz for Timer0 clock source
56    // 0 = 12 MHz, 1 = 32 kHz, 2 = HCLK, 7 = 22.1184 MHz
57    SYSCLK->APBCLK.TMR0_EN = 1; // Enable Timer0 clock source
58
59    /* Step 2. Select Operation mode */
60    TIMER0->TCSR.MODE = 2;       // 2 -> Select Toggle mode
61    // 0 = One shot, 1 = Periodic, 2 = Toggle, 3 = continuous counting mode
62
63    /* Step 3. Select Time out period
64    = (Period of timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP)*/
65    TIMER0->TCSR.PRESCALE = 11; // Set Prescale [0~255]
66    TIMER0->TCMPR = 1000000;     // Set TCMPR [0~16777215]
67    // (1/12000000)*(11+1)*(1000000)= 1 sec or 1 Hz
68
69    /* Step 4. Enable interrupt */
70    TIMER0->TCSR.IE = 1;
71    TIMER0->TISR.TIF = 1;        // Write 1 to clear the interrupt flag
72    NVIC_EnableIRQ(TMR0_IRQn);  // Enable Timer0 Interrupt
73
74    /* Step 5. Enable Timer module */
75    TIMER0->TCSR.CRST = 1;       // Reset up counter
76    TIMER0->TCSR.CEN = 1;        // Enable Timer0
77    }
78
79  void TMR0_IRQHandler(void) {  // Timer0 interrupt subroutine
80    char lcd2_buffer[18] = "Timer0:";
81    Timer0Counter += 1;
82    sprintf(lcd2_buffer+7, " %d s.", Timer0Counter);
83    print_lcd(2, lcd2_buffer);
84    TIMER0->TISR.TIF = 1;        // Write 1 to clear the interrupt flag
85    }
86
87  //-------------------------------------------------------------------------MAIN
88  int32_t main (void) {
89    UNLOCKREG();
90    SYSCLK->PWRCON.XTL12M_EN = 1;
91    SYSCLK->CLKSEL0.HCLK_S = 0;
92    LOCKREG();
93
94    Initial_pannel();  // call initial pannel function
95    clr_all_pannal();
96
97    InitTIMER0();
98    InitWDT();
99
100   while(1) {
101     __NOP();
102     }
103   }
```

Questions (WDT)

1. If we decide to use 32.768 kHz instead of 10 kHz, what is the value of time-out interval?

Assignment(s)

Summarize what you suppose to learn in this class.