ARM Interrupt

Part of this document is by
Prof. Shiao-Li Tsao
CS Dept., NCTU

SysTick Control

P246

4.5.1 SysTick control and status register (STK_CTRL)

Address offset: 0x00

Reset value: 0x0000 0000

Required privilege: Privileged

The SysTick CTRL register enables the SysTick features.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															COUNT FLAG
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													CLKSOURCE	TICKINT	ENABLE
													rw	rw	rw

SysTick Control

Bits 31:17 Reserved, must be kept cleared.

Bit 16 COUNTFLAG:
Returns 1 if timer counted to 0 since last time this was read.

Bits 15:3 Reserved, must be kept cleared.

Bit 2 **CLKSOURCE:** Clock source selection
Selects the clock source.
0: AHB/8
1: Processor clock (AHB)

Bit 1 **TICKINT**: SysTick exception request enable
0: Counting down to zero does not assert the SysTick exception request
1: Counting down to zero to asserts the SysTick exception request.
Note: Software can use COUNTFLAG to determine if SysTick has ever counted to zero.

Bit 0 **ENABLE**: Counter enable
Enables the counter. When ENABLE is set to 1, the counter loads the RELOAD value from the LOAD register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.
0: Counter disabled
1: Counter enabled

Values for SysTick Control Register

timer.c | helper_functions.c | main.c | stm32l476xx.h | startup_stm32.s | core_cm4.h

```
742   \brief  Structure type to access the System Timer (SysTick).
743  */
744 typedef struct
745 {
746   __IOM uint32_t CTRL;                    /*!< Offset: 0x000 (R/W)  SysTick Control and Status Register */
747   __IOM uint32_t LOAD;                    /*!< Offset: 0x004 (R/W)  SysTick Reload Value Register */
748   __IOM uint32_t VAL;                     /*!< Offset: 0x008 (R/W)  SysTick Current Value Register */
749   __IM  uint32_t CALIB;                   /*!< Offset: 0x00C (R/ )  SysTick Calibration Register */
750 } SysTick_Type;
```

r.c | helper_functions.c | main.c | stm32l476xx.h | startup_stm32.s | core_cm4.h

```
#define SysTick_CTRL_CLKSOURCE_Pos          2U                                            /*!< SysTick CTRL: CLKSOUR
#define SysTick_CTRL_CLKSOURCE_Msk         (1UL << SysTick_CTRL_CLKSOURCE_Pos)            /*!< SysTick CTRL: CLKSOUR

#define SysTick_CTRL_TICKINT_Pos            1U                                            /*!< SysTick CTRL: TICKINT
#define SysTick_CTRL_TICKINT_Msk           (1UL << SysTick_CTRL_TICKINT_Pos)              /*!< SysTick CTRL: TICKINT

#define SysTick_CTRL_ENABLE_Pos             0U                                            /*!< SysTick CTRL: ENABLE
#define SysTick_CTRL_ENABLE_Msk            (1UL /*<< SysTick_CTRL_ENABLE_Pos*/)           /*!< SysTick CTRL: ENABLE
```

P247

SysTick Load

4.5.2 SysTick reload value register (STK_LOAD)

Address offset: 0x04

Reset value: 0x0000 0000

Required privilege: Privileged

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								RELOAD[23:16]							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOAD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bits 23:0 **RELOAD:** RELOAD value

The LOAD register specifies the start value to load into the STK_VAL register when the counter is enabled and when it reaches 0.

Calculating the RELOAD value

The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use:

- To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.
- To deliver a single SysTick interrupt after a delay of N processor clock cycles, use a RELOAD of value N. For example, if a SysTick interrupt is required after 100 clock pulses, set RELOAD to 99.

Interrupt mask register (EXTI)

P330

12.5.1 Interrupt mask register 1 (EXTI_IMR1)

Address offset: 0x00

Reset value: 0xFF82 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IMx:** Interrupt Mask on line x (x = 31 to 0)
0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked

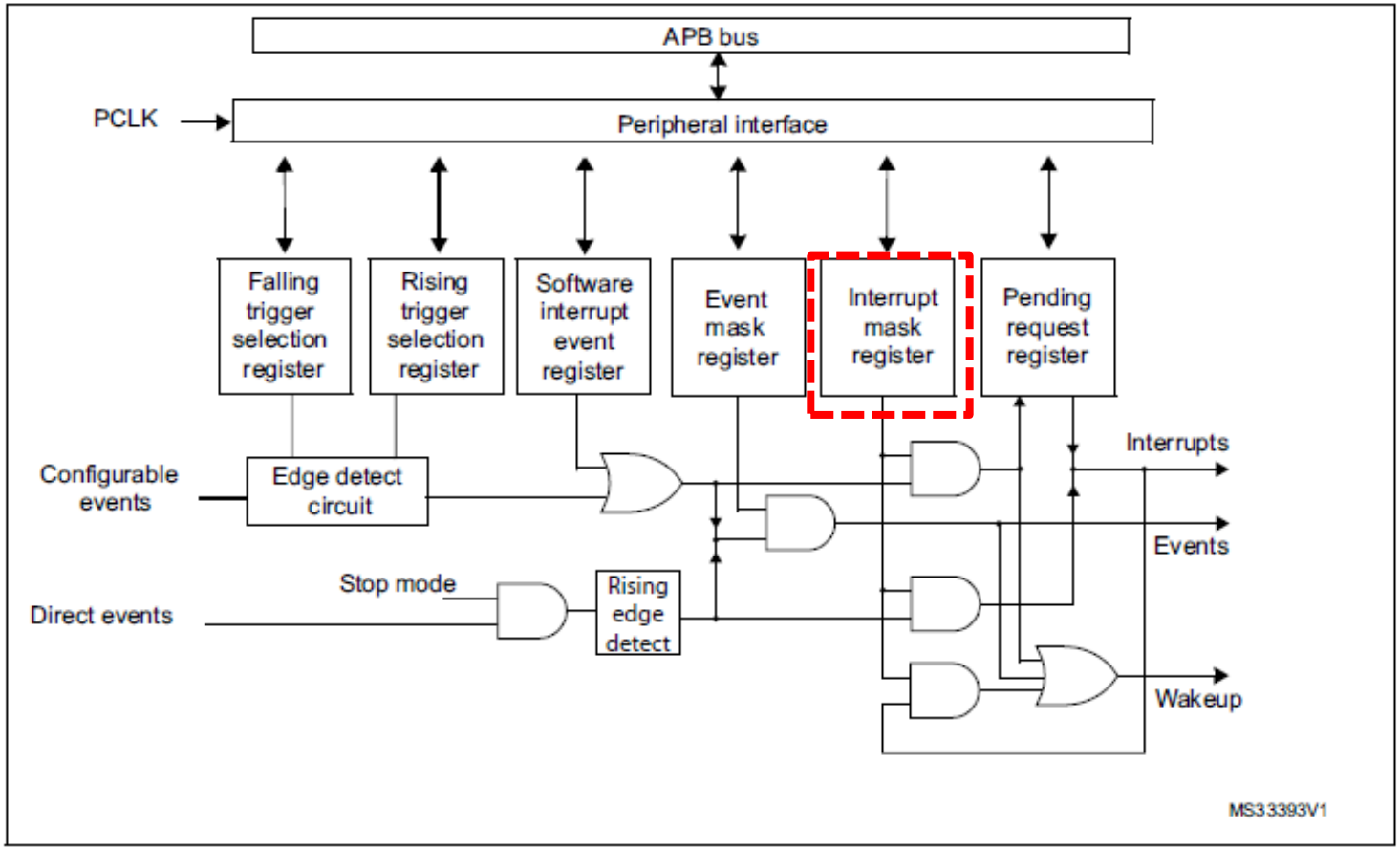
Note: The reset value for the direct lines (line 17, lines from 23 to 34, line 39) is set to '1' in order to enable the interrupt by default.

Interrupt mask register (EXTI)

12.3.1 EXTI block diagram

The extended interrupt/event block diagram is shown on *Figure 26*.

Figure 26. Configurable interrupt/event block diagram



EXTI

c | helper_functions.c | main.c | stm32l476xx.h | startup_stm32.s | core_cm4.h | main.c

```
typedef struct
{
  __IO uint32_t IMR1;       /*!< EXTI Interrupt mask register 1,           Address offset: 0x00 */
  __IO uint32_t EMR1;       /*!< EXTI Event mask register 1,               Address offset: 0x04 */
  __IO uint32_t RTSR1;      /*!< EXTI Rising trigger selection register 1, Address offset: 0x08 */
  __IO uint32_t FTSR1;      /*!< EXTI Falling trigger selection register 1,Address offset: 0x0C */
  __IO uint32_t SWIER1;     /*!< EXTI Software interrupt event register 1, Address offset: 0x10 */
  __IO uint32_t PR1;        /*!< EXTI Pending register 1,                  Address offset: 0x14 */
  uint32_t      RESERVED1;  /*!< Reserved, 0x18                                                 */
  uint32_t      RESERVED2;  /*!< Reserved, 0x1C                                                 */
  __IO uint32_t IMR2;       /*!< EXTI Interrupt mask register 2,           Address offset: 0x20 */
  __IO uint32_t EMR2;       /*!< EXTI Event mask register 2,               Address offset: 0x24 */
  __IO uint32_t RTSR2;      /*!< EXTI Rising trigger selection register 2, Address offset: 0x28 */
  __IO uint32_t FTSR2;      /*!< EXTI Falling trigger selection register 2,Address offset: 0x2C */
  __IO uint32_t SWIER2;     /*!< EXTI Software interrupt event register 2, Address offset: 0x30 */
  __IO uint32_t PR2;        /*!< EXTI Pending register 2,                  Address offset: 0x34 */
} EXTI_TypeDef;
```

Lab6 Interrupt
範例程式

Code provided by NCTU CS 謝明恩 吳赫倫
Slide made by NCTU ME 助教 林穎毅
20170502

Lab6.1: SysTick timer interrupt (30%)

- 作一個SysTick interrupt handler，當中斷發生時toggle LED燈明暗。
- 當使用者按下user button開啟或關閉SysTick timer。
- Notes: 設定SysTick clock source為10MHz，SysTick timer每 0.5秒 interrupt一次。
- 請利用操作NVIC的interrupt mask register的方式開啟或關閉Systick timer (not disable all interrupts)，NVIC相關register請參閱Reference manual section 4.3。

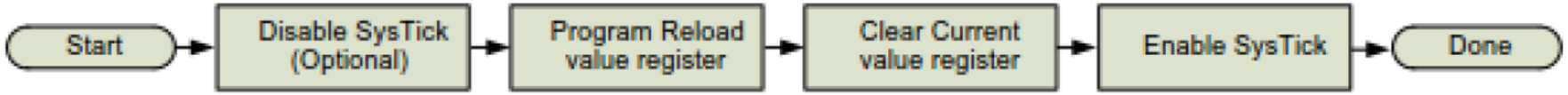


Figure 10.4:
Setup sequence for SysTick timer.

```
void SystemClock_Config(){
    //TODO: Setup system clock and SysTick timer interrupt
}
void SysTick_Handler(void) {
    //TODO: Toggle the LED
}
int main(){
    SystemClock_Config();
    GPIO_init();
    while(1){
        if(user_press_button())
        {
            //TODO: Enable or disable Systick timer
        }
    }
}
```

starup

- 把原本starup/starup_stm32.s換成我們給的starup_stm32.s

include 標頭檔

```
1 #include "stm32l476xx.h"
2 #include "helper_functions.h"
3 #include "7seg.h"
4 #include "keypad.h"
5 #include "led_button.h"
6 #include "timer.h"
```

定義pin角跟timer

```
// Define pins for led (default use on-board led PA5)
#define LED_gpio GPIOA
#define LED_pin 5

// Define pins for button (default use on-board button PC13)
#define BUTTON_gpio GPIOC
#define BUTTON_pin 13

// Define Counter timer
#define COUNTER_timer TIM2
```

SysTick_Handler

每隔一段時間執行一次interrupt，讓LED閃爍

```
void SysTick_Handler() {


    if(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk){
        // Toggle LED display
        toggle_output(LED_gpio, LED_pin);
    }
```

用現在的OUTPUT來改

位段	名称	类型	复位值	描述
16	COUNTFLAG	R	0	如果在上次读取本寄存器后，SysTick 已经计到了 0，则该位为 1。如果读取该位，该位将自动清零
2	CLKSOURCE	R/W	0	0=外部时钟源(STCLK) 1=内核时钟(FCLK)
1	TICKINT	R/W	0	1=SysTick 倒数计数到 0 时产生 SysTick 异常请求 0=数到 0 时无动作
0	ENABLE	R/W	0	SysTick 定时器的使能位

表8.10　SysTick重装载数值寄存器（地址：0xE000_E014）

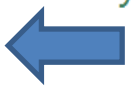
main

```
int main(){
    // Cause we want to use floating points we need to init FPU
    FPU_init();


    if(init_led(LED_gpio, LED_pin) != 0){
        // Fail to init led
        return -1;
    }
    if(init_button(BUTTON_gpio, BUTTON_pin) != 0){
        // Fail to init button
        return -1;
    }
```

main

```
// Configure SysTick clk to 10 Mhz and interrupt every 0.5s
SystemClock_Config_Interrupt( 1   5000000);
                              0，
// button_press_cycle_per_second (How many button press segments in a second)
int button_press_cycle_per_second = 20;
// Use to state how many cycles to check per button_press_cycle
int debounce_cycles = 50;
// Use to state the threshold when we consider a button press
int debounce_threshold = debounce_cycles*0.7;
// Used to implement negative edge trigger 0=not-presses 1=pressed
int last_button_state=0;
```



設定clock的參數，控制閃爍速度

main

```
while(1){
    for(int a=0;a<button_press_cycle_per_second;a++){
        // Simple Debounce without interrupt
        int pos_cnt=0;
        for(int a=0;a<debounce_cycles;a++){
            // If button press add count
            if(read_gpio(BUTTON_gpio, BUTTON_pin)==0){
                pos_cnt++;
            }
            delay_without_interrupt(1000/(button_press_cycle_per_second*debounce_cycles));
        }
```

main

```
// Check if need to change state
if(pos_cnt>debounce_threshold){
    if(last_button_state==0){
        // Pressed button - Pos edge
        SysTick->CTRL ^= (1 << SysTick_CTRL_ENABLE_Pos);//exclusive or
    }
    else{
        // Pressed button - Continued pressing
        // Do nothing
    }
    last_button_state = 1;
}
```

把計時器關掉/開啟

main

```
            else{
                if(last_button_state==0){
                    // Released button - Not pressing
                    // Do nothing
                }
                else{
                    // Released button - Neg edge
                    // Do nothing
                }
                last_button_state = 0;
            }
        }
    }
    while(1){}

    return 0;
}
```

SystemClock_Config_interrupt

```
void SystemClock_Config_Interrupt(int speed, int load){
    SystemClock_Config(speed);
    SysTick->LOAD = load;
    SysTick->CTRL |= (1 << SysTick_CTRL_CLKSOURCE_Pos);
    SysTick->CTRL |= (1 << SysTick_CTRL_TICKINT_Pos);
    SysTick->CTRL |= (1 << SysTick_CTRL_ENABLE_Pos);
}
```

位段	名称	类型	复位值	描述
16	COUNTFLAG	R	0	如果在上次读取本寄存器后，SysTick 已经计到了 0，则该位为 1。如果读取该位，该位将自动清零
2	CLKSOURCE	R/W	0	0=外部时钟源(STCLK) 1=内核时钟(FCLK)
1	TICKINT	R/W	0	1=SysTick 倒数计数到 0 时产生 SysTick 异常请求 0=数到 0 时无动作
0	ENABLE	R/W	0	SysTick 定时器的使能位

表8.10　SysTick重装载数值寄存器（地址：0xE000_E014）

位段	名称	类型	复位值	描述
23:0	RELOAD	R/W	0	当倒数计数至零时，将被重装载的值

表8.11　SysTick当前数值寄存器（地址：0xE000_E018）

SysteClock_Config

```
void SystemClock_Config(int speed){
    // system clock -> MSI
    RCC->CFGR &= ~RCC_CFGR_SW_Msk;
    RCC->CFGR |= RCC_CFGR_SW_MSI;

    while(!(((RCC->CFGR & RCC_CFGR_SWS_Msk)>> RCC_CFGR_SWS_Pos) == 0));

    RCC->CR &= ~RCC_CR_PLLON;
    while((RCC->CR & RCC_CR_PLLRDY) != 0);

    // Set PLL to MSI
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC_Msk;
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_MSI;
```

```
int set_R=0, set_N=0, set_M=0;
// Change R N M
if(speed==40){
    set_R = 1;
    set_N = 40;
    set_M = 0;
}
else if(speed==16){
    set_R = 0;
    set_N = 8;
    set_M = 0;
}
else if(speed==10){
    set_R = 0;
    set_N = 5;
    set_M = 0;
}
```

```
else if(speed==6){
    set_R = 0;
    set_N = 12;
    set_M = 3;
}
else if(speed==1){
    set_R = 3;
    set_N = 8;
    set_M = 3;
}
else{
    // Default 4 MHz
    set_R = 3;
    set_N = 8;
    set_M = 0;
}
```

```
    // Set PLLR
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLR_Msk;
    RCC->PLLCFGR |= (set_R << RCC_PLLCFGR_PLLR_Pos);
    // Set PLLN
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLN_Msk;
    RCC->PLLCFGR |= (set_N << RCC_PLLCFGR_PLLN_Pos);
    // Set PLLM
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLM_Msk;
    RCC->PLLCFGR |= (set_M << RCC_PLLCFGR_PLLM_Pos);

    // Enable PLLR
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN;

    // Enable PLL
    RCC->CR |= RCC_CR_PLLON;

    // system clock -> PLL
    RCC->CFGR &= ~RCC_CFGR_SW_Msk;
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    while(!(((RCC->CFGR & RCC_CFGR_SWS_Msk)>>RCC_CFGR_SWS_Pos) == 3));
}
```

Lab6.2: Keypad external interrupt (30%)

- 這部分的實驗主要請同學將Lab6中所實作的鍵盤掃描程式改成利用SysTick與外部中斷EXTI完成(無須掃描迴圈)。主要原理由以下3個部分完成。
- 將Column output掃描由SysTick interrupt handler完成，中斷時間間隔0.1s，當SysTick中斷發生時更改scan column。
- 在SysTick interrupt handler中設定並啟動keypad row的4個input腳為負邊緣觸發(Negative trigger)的外部中斷
- 當EXIT中斷發生時讀取4個input的值，並根據目前column掃描狀態判斷是哪個鍵按下。
- 在主程式中依使用者所按下的按鍵值利用lab6的display()顯示至7段顯示器上。

Lab6.2: Keypad external interrupt (30%)

	X0	X1	X2	X3
Y0	1	2	3	10
Y1	4	5	6	11
Y2	7	8	9	12
Y3	15	0	14	13

請依以下TODO說明完成程式碼

```
char key_value = 0;
void EXIT_Setup(){
    //TODO: Setup EXTI interrupt
}
void SystemClock_Config(){
    //TODO: Setup system clock and SysTick timer interrupt
}
void SysTick_Handler(void) {
    //TODO: Scan the keypad column
}
void EXTIx_IRQHandler(void){
    //TODO: Read the keypad row value
}
int main(){
    SystemClock_Config();
    GPIO_init();
    EXTI_Setup();
    while(1){
        display(key_value, 2);
    }
}
```

include 標頭檔

```
1 #include "stm32l476xx.h"
2 #include "helper_functions.h"
3 #include "7seg.h"
4 #include "keypad.h"
5 #include "led_button.h"
6 #include "timer.h"
```

```c
// Define pins for 7seg
#define SEG_gpio GPIOC
#define DIN_pin 3
#define CS_pin 4
#define CLK_pin 5

// Define pins for keypad
// If need to change need to also change EXTI_Setup and IRQHandler
#define COL_gpio GPIOA
#define COL_pin 6       // 6 7 8 9
#define ROW_gpio GPIOB
#define ROW_pin 3       // 3 4 5 6

// Define Counter timer
#define COUNTER_timer TIM2


int now_col = 3;
// Used for debounce
int keyCnt=0, keyValue=-1;
```

SysTick_Handler

每隔一段時間執行一次interrupt，改變讀取的column

```
void SysTick_Handler() {

        if(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk){
            reset_push(COL_gpio, now_col+COL_pin);
            now_col = (now_col+1)%4;
            set_push(COL_gpio, now_col+COL_pin);
        }
    }
```

EXTI_Setup

```
void EXTI_Setup(){
    // Enable SYSCFG CLK
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    // Select output bits
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3_Msk;
    SYSCFG->EXTICR[0] |= (1 << SYSCFG_EXTICR1_EXTI3_Pos);
    SYSCFG->EXTICR[1] &= ~SYSCFG_EXTICR2_EXTI4_Msk;
    SYSCFG->EXTICR[1] |= (1 << SYSCFG_EXTICR2_EXTI4_Pos);
    SYSCFG->EXTICR[1] &= ~SYSCFG_EXTICR2_EXTI5_Msk;
    SYSCFG->EXTICR[1] |= (1 << SYSCFG_EXTICR2_EXTI5_Pos);
    SYSCFG->EXTICR[1] &= ~SYSCFG_EXTICR2_EXTI6_Msk;
    SYSCFG->EXTICR[1] |= (1 << SYSCFG_EXTICR2_EXTI6_Pos);
```

設定interrupt的pin角
PB3456

8.2.3 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	EXTI3[2:0]			Res	EXTI2[2:0]			Res	EXTI1[2:0]			Res	EXTI0[2:0]		
	rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw	rw

Bits 14:12 **EXTI3[2:0]**: EXTI 3 configuration bits

These bits are written by software to select the source input for the EXTI3 external interrupt.

000: PA[3] pin
001: PB[3] pin
010: PC[3] pin
011: PD[3] pin
100: PE[3] pin
101: PF[3] pin
110: PG[3] pin
111: Reserved

EXTI_Setup

```
// Enable interrupt
EXTI->IMR1 |= EXTI_IMR1_IM3;
EXTI->IMR1 |= EXTI_IMR1_IM4;
EXTI->IMR1 |= EXTI_IMR1_IM5;
EXTI->IMR1 |= EXTI IMR1 IM6;

// Enable Falling Edge
EXTI->FTSR1 |= EXTI_FTSR1_FT3;
EXTI->FTSR1 |= EXTI_FTSR1_FT4;
EXTI->FTSR1 |= EXTI_FTSR1_FT5;
EXTI->FTSR1 |= EXTI_FTSR1_FT6;

// Enable NVIC**
NVIC_EnableIRQ(EXTI3_IRQn);
NVIC_EnableIRQ(EXTI4_IRQn);
NVIC_EnableIRQ(EXTI9_5_IRQn);

}
```

Unmask interrupt

設falling edge為我們判斷的依據

Interrupt 設為有效

12.5.1 Interrupt mask register 1 (EXTI_IMR1)

Address offset: 0x00

Reset value: 0xFF82 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IMx:** Interrupt Mask on line x (x = 31 to 0)

0: Interrupt request from Line x is masked

1: Interrupt request from Line x is not masked

12.5.4 Falling trigger selection register 1 (EXTI_FTSR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FT22	FT21	FT20	FT19	FT18	Res.	FT16
									rw	rw	rw	rw	rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FT15	FT14	FT13	FT12	FT11	FT10	FT9	FT8	FT7	FT6	FT5	FT4	FT3	FT2	FT1	FT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 16:0 **FTx:** Falling trigger event configuration bit of line x (x = 16 to 0)

0: Falling trigger disabled (for Event and Interrupt) for input line

1: Falling trigger enabled (for Event and Interrupt) for input line

EXTIKeypadHandler

```
void EXTIKeypadHandler(int r){
    int nowKey = keypad[r][(now_col + 3) % 4];
    // A simple debounce
    if(nowKey == keyValue){
        keyCnt++;
    }
    else{
        keyCnt = 0;
    }
    keyValue = nowKey;
    if(keyCnt >= 5){
        keyCnt = 5;

        display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, keyValue, 2);
     }
}
```

把讀到的值輸出

```
void EXTI3_IRQHandler(){
    if(EXTI->PR1&EXTI_PR1_PIF3_Msk){
        EXTIKeypadHandler(0);
        EXTI->PR1 = EXTI_PR1_PIF3_Msk;//?
    }
}

void EXTI4_IRQHandler(){
    if(EXTI->PR1 & EXTI_PR1_PIF4_Msk){
        EXTIKeypadHandler(1);
        EXTI->PR1 = EXTI_PR1_PIF4_Msk;
    }
}

void EXTI9_5_IRQHandler(){
    if(EXTI->PR1 & EXTI_PR1_PIF5_Msk){
        EXTIKeypadHandler(2);
        EXTI->PR1 = EXTI_PR1_PIF5_Msk;
    }
    if(EXTI->PR1 & EXTI_PR1_PIF6_Msk){
        EXTIKeypadHandler(3);
        EXTI->PR1 = EXTI_PR1_PIF6_Msk;
    }
}
```

當讀到按鈕時進行中斷

清掉 interrupt

main

```
int main(){

  if(init_7seg_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin) != 0){
      // Fail to init 7seg
      return -1;
  }
  if(init_keypad(ROW_gpio, COL_gpio, ROW_pin, COL_pin) != 0){
      // Fail to init keypad
      return -1;
  }

  // Set 10MHz 0.001s interrupt
  SystemClock_Config_Interrupt(10, 10000);
  // Init Interrupts
  EXTI_Setup();

    while(1){}

    return 0;
}
```

SystemClock_Config_interrupt

```
void SystemClock_Config_Interrupt(int speed, int load){
    SystemClock_Config(speed);
    SysTick->LOAD = load;
    SysTick->CTRL |= (1 << SysTick_CTRL_CLKSOURCE_Pos);
    SysTick->CTRL |= (1 << SysTick_CTRL_TICKINT_Pos);
    SysTick->CTRL |= (1 << SysTick_CTRL_ENABLE_Pos);
}
```

SysteClock_Config

```
void SystemClock_Config(int speed){
    // system clock -> MSI
    RCC->CFGR &= ~RCC_CFGR_SW_Msk;
    RCC->CFGR |= RCC_CFGR_SW_MSI;

    while(!(((RCC->CFGR & RCC_CFGR_SWS_Msk)>> RCC_CFGR_SWS_Pos) == 0));

    RCC->CR &= ~RCC_CR_PLLON;
    while((RCC->CR & RCC_CR_PLLRDY) != 0);

    // Set PLL to MSI
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC_Msk;
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_MSI;
```

```
int set_R=0, set_N=0, set_M=0;
// Change R N M
if(speed==40){
    set_R = 1;
    set_N = 40;
    set_M = 0;
}
else if(speed==16){
    set_R = 0;
    set_N = 8;
    set_M = 0;
}
else if(speed==10){
    set_R = 0;
    set_N = 5;
    set_M = 0;
}
```

```
else if(speed==6){
    set_R = 0;
    set_N = 12;
    set_M = 3;
}
else if(speed==1){
    set_R = 3;
    set_N = 8;
    set_M = 3;
}
else{
    // Default 4 MHz
    set_R = 3;
    set_N = 8;
    set_M = 0;
}
```

```
    // Set PLLR
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLR_Msk;
    RCC->PLLCFGR |= (set_R << RCC_PLLCFGR_PLLR_Pos);
    // Set PLLN
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLN_Msk;
    RCC->PLLCFGR |= (set_N << RCC_PLLCFGR_PLLN_Pos);
    // Set PLLM
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLM_Msk;
    RCC->PLLCFGR |= (set_M << RCC_PLLCFGR_PLLM_Pos);

    // Enable PLLR
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN;

    // Enable PLL
    RCC->CR |= RCC_CR_PLLON;

    // system clock -> PLL
    RCC->CFGR &= ~RCC_CFGR_SW_Msk;
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    while(!(((RCC->CFGR & RCC_CFGR_SWS_Msk)>>RCC_CFGR_SWS_Pos) == 3));
}
```

HW Lab6.3: 製作簡單鬧鐘 (40%)

- 利用SysTick timer、User button和蜂鳴器設計一個簡單的鬧鐘，
- 利用keypad輸入計時鬧鐘倒數時間並即時顯示至7-Seg LED，每一個數字代表設定幾秒(2為2秒)
- 輸入為0時則沒反應，繼續等待下次輸入，
- 按下User button則代表時間輸入完畢
- 啟動一秒觸發一次interrupt的Systick timer開始倒數，
- 利用7-seg LED顯示目前倒數的時間秒數
- 當時間到後，蜂鳴器便會響起(在SysTick interrupt handler中利用while loop讓蜂鳴器持續發出聲音，頻率自訂)
- 直到使用者按下User button後才會停止發出聲音並回到等待使用者輸入狀態，注意SysTick開始計時到使用者關閉蜂鳴器的期間，keypad不會有任何作用。(程式會由user button觸發一個nested interrupt)
- Note:
- 1.注意SysTick timer中斷和User button外部中斷的Priority關係。
- 2. SysTick clock source 設定為10MHz

HW Lab6.3: 製作簡單鬧鐘 (40%)

	X0	X1	X2	X3
Y0	1	2	3	10
Y1	4	5	6	11
Y2	7	8	9	12
Y3	15	0	14	13