

REPORT

Title: SysTick and External Interrupt



Submission Date	2021.10.22	Major	Mechanical and control engineering
Subject	Embedded Controller	Professor	Young-Keun Kim
Name	Yeong-Won Song	Student Number	21700375
Partner Name	Jin-su Yu	Partner Number	21700469

Contents

I. Introduction	3
1.1 Purpose	3
1.2 Parts List.....	3
 II. Procedure	 4
2.1 Create HAL, API driver	4
2.2 LED Toggle with EXTI Button.....	9
2.3 7-Segment Display with EXTI Button	10
2.4 Create User API	12
 III. Conclusion	 14
 Appendix	

I. Introduction

1.1 Purpose

In this lab, we are required to create a simple program that toggle multiple LEDs with a push-button input. Create HAL drivers for GPIO digital in and out control and use these APIs for the lab. the number by 1 second and display it on 7-segment led. Also, we will learn how to configure external interrupt (EXTI) to reset the number with push-button.

1.2 Parts List

Parts	Specification	Quantity
7-Segment Array Register	5101ASR	1
	330[Ω]	1
breadboard	-	1
NUCLEO-F411RE	Arm®(a) Cortex®-M4 with FPU	1
M-F Jumper Wire	-	5
M-M Jumper Wire	-	5

Table 1. Part List

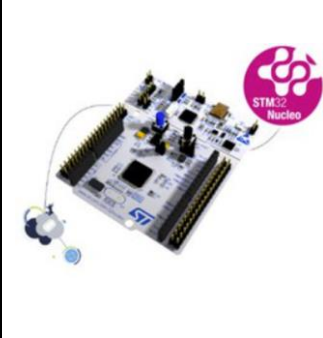




				
NUCLEO-F411RE	7-segment	Array Register	Breadboard	M-F/M-M jumper

Table 2. experiment equipment

II. Procedure

2.1 Create HAL, API driver

Below are the examples of functions for Digital In and Out.

Include File	Function
ecEXTI.h, c	void EXTI_init(GPIO_TypeDef *port, int pin, int trig_type, int priority);
	void EXTI_enable(uint32_t pin); // mask in IMR
	void EXTI_disable(uint32_t pin); // unmask in IMR
	uint32_t is_pending_EXTI(uint32_t pin);
esSysTick.h, c	void clear_pending_EXTI(uint32_t pin);
	void SysTick_init(uint32_t msec);
	void delay_ms(uint32_t msec);
	uint32_t SysTick_val(void);
	void SysTick_reset (void);
	void SysTick_enable(void);
	void SysTick_disable (void);
	void EXTIx_IRQHandler(void); // in main()
	void SysTick_Handler (void); // in main() or SysTick.h

2.2 LED-Toggle with EXTI Button

Use your HAL library to toggle LED2 with User button. MUST use External Interrupt.

```

/**
 * @author Name : Song Yeong Won
 * @Mod 2021-10-22 by YWSONG
 * @brief Embedded Controller: LAB SysTick and External Interrupt
 *
 */

#include "stm32f4llxe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "exEXTI.h"

#define LED_PIN 5

void setup(void);
void LED_toggle(GPIO_TypeDef *Port, uint32_t pin);
void EXTI15_10_IRQHandler(void);

int main(void){
    setup();

    while(1){}
}

void setup(void){
    RCC_PLL_init();
    GPIO_init(GPIOA, LED_PIN, OUTPUT);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pudr(GPIOC, BUTTON_PIN, PD);
}

void LED_toggle(GPIO_TypeDef *Port, uint32_t pin){
    Port->ODR ^= (1<<pin);
}

void EXTI15_10_IRQHandler(void){
    if(is_pending_EXTI(BUTTON_PIN)){
        LED_toggle(GPIOA, LED_PIN);
        clear_pending_EXTI(BUTTON_PIN);
    }
}

```

Figure 1. EXTI Button main.c

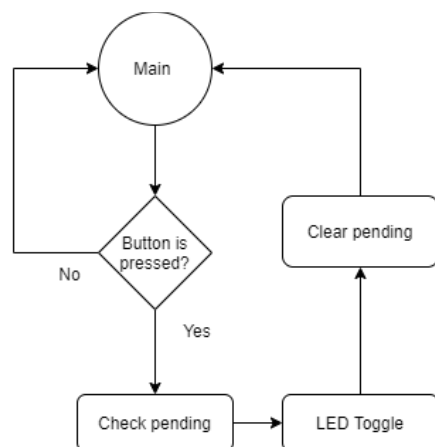


Figure2. Flow chart of LED-Toggle

2.3 7-Segment Display with EXTI Button

Create a new project named as “LAB_EXTI_SysTick”.

You MUST write your name in the top of the source file, inside the comment section.

- **Configure Input and Output pins**

Digital In: Button	Digital Out:
GPIOC, Pin 13	PA5, PA6, PA7, PC7, PA9, PA8, PB10
Digital Input	Digital Output
Set PULL-UP	Push-Pull, No Pull-up Pull-down

- **Display a number in sequence with timer**

Display the number 0 to 9 on the 7-segment LED at the rate of 1 sec. After displaying up to 9, then it should display ‘0’ and continue counting. When the button is pressed it should reset ‘0’ and start counting.

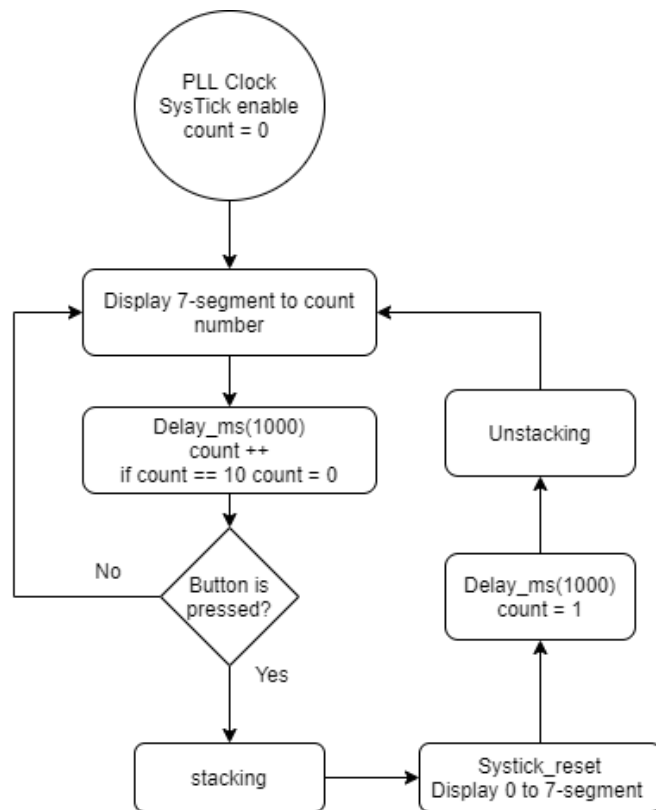


Figure 3. Flow chart

```

/**
*****
* @author Name : Song Yeong Won
* @Mod 2021-10-22 by YWSONG
* @brief Embedded Controller: LAB SysTick and External Interrupt
*
*****
*/

#include "stm32f4llxe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "exEXTI.h"
#include "ecSysTick.h"

//Initialization
static volatile uint32_t count = 0;
static volatile uint32_t PLL_CLK = 84000000;
void EXTI15_10_IRQHandler(void);

void setup(void)
{
    RCC_PLL_init();
    SysTick_init(PLL_CLK);
    sevensegment_init();
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
    GPIO_pudr(GPIOC, BUTTON_PIN, PU);
}

int main (void){
    setup();

    while(1){
        sevensegment_decoder(count);
        delay_ms(1000);
        count++;
        if(EX == 1){
            delay_ms(1000);
            EX=0;
            count = 1;
        }
        if(count == 10)count =0;
    }
}

void EXTI15_10_IRQHandler(void){
    if(is_pending_EXTI(BUTTON_PIN)){
        SysTick_reset();
        sevensegment_decoder(0);
        EX = 1;
        clear_pending_EXTI(BUTTON_PIN);
    }
}

```

Figure 4. 7-Segment Display with EXTI Button main.c

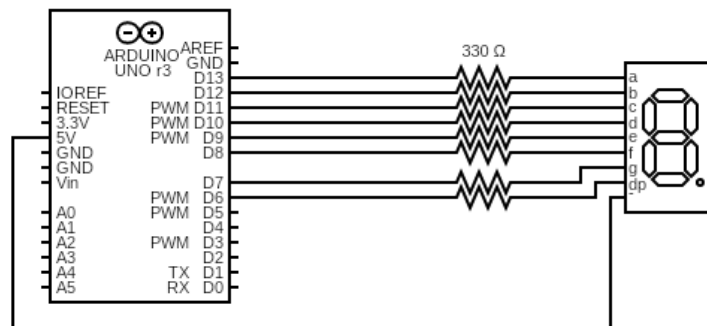


Figure 5. External circuit of 7-Segment Display

- Discussion

- 1) To detect an external signal, we can use two different methods: polling and interrupt. What are the advantages and disadvantages of each approach?

Polling is a method of periodically inspecting the occurrence of an event and executing it when receiving the signal. Polling is easy to implement on software. However, there is a disadvantage in that the response time is slow and takes up a lot of resources because the system is constantly loaded. In addition, because the signal is checked at a specific period, it cannot respond immediately to the exact timing. The reaction is slow and there is an error according to the cycle.

Interrupt is a method of executing a command through Handler only when an event occurs. Interrupt responds quickly to the exact timing when an event occurs at that moment. Since the process is performed only when the interrupt is occurred, the load on the system can be reduced. However, since interrupt must be implemented through Handler, there are difficulties in implementation.

- 2) What would happen if the EXTI interrupt handler dose not clear the interrupt pending flags? Check with your code

When a new interrupt occurs, the CPU executes a high priority interrupt first. Low priority Interrupt waits for the next execution sequence in the pending state. After executing Interrupt, write 1 on the pending bit to clear the pending state. If clear pending is not performed, the same operation will be repeated. We compared when clear pending was performed and when not performed in the LED-Toggle part. First, if clear pending is performed, interrupt occurs when Button_Pin is pressed, and once the pending bit is cleared, the same interrupt is not repeatedly executed and another interrupt is waited. Therefore, when the button is pressed, the LED turns on and when pressed again, the LED turns off. However, if clear pending is not performed, the LED will turn on when the button is pressed and will not be able to leave the pending state continuously. In the end, since it cannot respond to another interrupt, even if the button is pressed again, the LED does not turn off and remains on.

<pre>void EXTI15_10_IRQHandler(void) { if (is_pending_EXTI(BUTTON_PIN)) { SysTick_reset(); sevensegment_decoder(0); EX = 1; clear_pending_EXTI(BUTTON_PIN); } }</pre>	<pre>void EXTI15_10_IRQHandler(void) { if (is_pending_EXTI(BUTTON_PIN)) { SysTick_reset(); sevensegment_decoder(0); EX = 1; //clear_pending_EXTI(BUTTON_PIN); } }</pre>
Execute clear pending	Not Execute clear pending

Figure 6. Compare execute clear pending or not

2.4 Create User API

Below are the examples of functions.

Include file	Function
GPIOC, Pin 13	PA5, PA6, PA7, PC7, PA9, PA8, PB10
Digital Input	Digital Output
Set PULL-UP	Push-Pull, No Pull-up Pull-down

```

/**
*****
* @author   Name : Song Yeong Won
* @Mod      2021-10-22 by YWSONG
* @brief    Embedded Controller:  LAB SysTick and External Interrupt
*
*****
*/

#include "EC_API.h"
EC_Ticker tick(1);
volatile uint32_t count = 0;

//Initialization
void setup(void)
{
    RCC_PLL_init();
    sevensegment_init();
}

int main (void){
    setup();
    while(1){
        sevensegment_decoder(count);
        tick.Delay_ms(1000);
        count++;
        if(count == 10) count =0;
        tick.reset();
    }
}

```

Figure 7. User API main.cpp

III. Conclusion

Through this lab, we learned how to use it for application directly through programming based on an understanding of the concepts of polling and external interrupt. The difference in interrupt behavior according to the presence or absence of pending was checked by coding. The start and end count values can be measured using the SysTick Val function to measure the time when the function is executed. If this is used as a function, it will be useful for many applications.

- Troubleshooting

Q. what algorithm was applied to operate exactly like a reset button?

A. When the reset button is pressed, 7-segment immediately indicates 0, and counts again for 1 second from the time when the button is pressed. Therefore, when the external interrupt of the Button pin occurs, the current count value has been initialized and 0 is immediately indicated in the 7-segment. Then, 1000ms delay was given and the 7-segment count value was increased by 1 so that 7-segment appeared sequentially again from 1 to 9 after unstacking.

Appendix

- **Video demo Link**

Click [here](#) watch video.

- **Source code**

<ecEXTI.h>

```

#ifndef __EC_EXTI_H
#define __EC_EXTI_H

#include "stm32f411xe.h"

//EXTI interrupt
#define PA_PIN 0
#define PB_PIN 1
#define PC_PIN 2
#define PD_PIN 3
#define PE_PIN 4
#define PH_PIN 7

#define FALL 0
#define RISE 1
#define BOTH 2

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

void EXTI_init(GPIO_TypeDef *Port, uint32_t pin, int trig,int priority);
void EXTI_clearpending(uint32_t pin);
void EXTI_enable(uint32_t pin);
void EXTI_disable(uint32_t pin);
uint32_t is_pending_EXTI(uint32_t pin);
void clear_pending_EXTI(uint32_t pin);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif

```

Figure 8. ecEXTI.h code

<ecEXTI.c>

```

#include "exEXTI.h"

void EXTI_init(GPIO_TypeDef *port, uint32_t pin, int trig_type, int priority){
    //RCC->APB2ENR |= 1UL << 14;
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    //EXTIx connection GPIO Pinx
    SYSCFG->EXTICR[pin/4] &= ~ 15UL<<4*(pin%4);

    if(port == GPIOA){
        SYSCFG->EXTICR[pin/4] = PA_PIN<<4*(pin%4);
    }
    if(port == GPIOB){
        SYSCFG->EXTICR[pin/4] = PB_PIN<<4*(pin%4);
    }
    if(port == GPIOC){
        SYSCFG->EXTICR[pin/4] = PC_PIN<<4*(pin%4);
    }
    if(port == GPIOD){
        SYSCFG->EXTICR[pin/4] = PD_PIN<<4*(pin%4);
    }
    if(port == GPIOE){
        SYSCFG->EXTICR[pin/4] = PE_PIN<<4*(pin%4);
    }
    if(port == GPIOH){
        SYSCFG->EXTICR[pin/4] = PH_PIN<<4*(pin%4);
    }

    //triger type (Falling, rising, both)
    if(trig_type == FALL){
        EXTI->FTSR |= 1<<pin;
    }
    if(trig_type == RISE){
        EXTI->RTSR |= 1<<pin;
    }
    if(trig_type == BOTH){
        EXTI->FTSR |= 1<<pin;
        EXTI->RTSR |= 1<<pin;
    }

    EXTI_enable(pin);

    // priority
    uint32_t IRQN = 0;
    if(pin<=4){
        IRQN += 6;
    }
    else if(pin<=9){
        IRQN = 23;
    }
    else if(pin<=15){
        {
            IRQN = 40;
        }
        NVIC_SetPriority(IRQN, priority); // Set EXTI priority as 0
        NVIC_EnableIRQ(IRQN);           // Enable EXTI
    }
}

void EXTI_enable(uint32_t pin){ // unmask in IMR
    EXTI -> IMR |= 1<<pin;
    uint32_t EN = 0;
    if(pin<=4){
        EN +=6;
        NVIC_EnableIRQ(EN);
    }
}

```

```

else if(pin<=9){
    EN = 23;
    NVIC_EnableIRQ(EN);
}
else if(pin<=15){
    EN = 40;
    NVIC_EnableIRQ(EN);
}
}

void EXTI_disable(uint32_t pin){// mask in IMR
    EXTI->IMR &= 0UL<<pin;
}

uint32_t is_pending_EXTI(uint32_t pin){
    return (EXTI->PR & 1UL<<pin) == (1<<pin);
}

void clear_pending_EXTI(uint32_t pin){
    EXTI->PR |= 1UL<<pin;
}

```

Figure 9. ecEXTI.c code

<ecSysTick.h>

```

#ifndef __EC_SYSTICK_H
#define __EC_SYSTICK_H

#include "stm32f411xe.h"
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

void SysTick_init(uint32_t msec);
void delay_ms(uint32_t msec);
uint32_t SysTick_val(void);
void SysTick_reset(void);
void SysTick_enable(void);
void SysTick_disable(void);
void SysTick_Handler(void);
static volatile uint32_t TimeDelay = 0;
static volatile uint32_t EX = 0;
#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif

```

Figure 10. ecSysTick.h code

< ecSysTick .c>

```
#include "ecSysTick.h"

void SysTick_init(uint32_t msec){
    SysTick->CTRL &= 0<<0; //Disable SysTick
    SysTick->LOAD = msec/1000 -1; //Set Reload register
    SysTick->VAL = 0; //Reset the SysTick counter value
    SysTick->CTRL |= 1<<2; //Select processor clock
    SysTick->CTRL |= 1<<1; //Enable SysTick interrupt
    SysTick->CTRL |= 1<<0; //Enable SysTick
    NVIC_SetPriority(SysTick_IRQn,3); //Set interrupt priority
}

uint32_t SysTick_val(void){
    return SysTick->VAL &= 0xffffffff;
}

void SysTick_reset (void){
    TimeDelay = 0;
    SysTick->VAL = 0;
}

void SysTick_enable(void){
    SysTick->CTRL |= 1<<0;
}

void SysTick_disable (void){
    SysTick->CTRL &= 0<<0;
}

void SysTick_Handler(void){
    if(TimeDelay >0){
        TimeDelay --;
    }
}

void delay_ms(uint32_t msec){
    TimeDelay = msec;
    while(TimeDelay !=0);
}
```

Figure 11. ecSysTick.c code

<EC_API.h>

```
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecSysTick.h"
#include "exEXTI.h"

#ifndef __EC_GPIO_API_H
#define __EC_GPIO_API_H

#define EC_DOUT 1
#define EC_DIN 0

#define EC_PU 1
#define EC_PD 0
#define EC_NONE 0

#define EC_LOW 0
#define EC_MEDIUM 1
#define EC_FAST 2
#define EC_HIGH 3

/* System CLOCK is HSI by default */
class EC_DigitalIn
{
public:
    EC_DigitalIn(GPIO_TypeDef *Port, int pin)
    {
        uint8_t mode=EC_DIN; // mode=0
        GPIO_init(Port, pin, mode);
        Port_t=Port;
        pin_t=pin;
        mode_t=mode;
    }
}
```

```

~EC_DigitalIn()
{
    delete[] Port_t;
}

int read();

void pupdr(int _pupdr);

operator int()
{
    return read();
}

private:
    GPIO_TypeDef *Port_t;
    int pin_t;
    int mode_t;
    int val_t;
};

class EC_DigitalOut
{
public:
    EC_DigitalOut(GPIO_TypeDef *Port, int pin);
    ~EC_DigitalOut()
    {
        delete[] Port_t;
    }

    void write(int _outVal);
    void pupdr(int _pupdr);
    void otype(int _type);
    void ospeed(int _speed);

    EC_DigitalOut &operator= (int value)
    {
        write(value);
        return *this;
    }

    int read()
    {
        return GPIO_read(Port_t, pin_t);
    }

    operator int()
    {
        // Underlying call is thread safe
        return read();
    }

private: //only access in class
    GPIO_TypeDef *Port_t;
    int pin_t; // _t -> internal
    int mode_t;
};

```

```

class EC_Ticker
{
public:
    EC_Ticker(uint32_t freq)
    {
        frequency = freq * 84000000;
        SysTick_init(frequency);
    }
    void Delay_ms(uint32_t msec);
    uint32_t read_ms(void);
    void reset (void);

private:
    uint32_t frequency ;
};

#endif

```

Figure 12. EC_API.h code

<EC_API.c>

```

#include "EC_API.h"

/* System CLOCK is HSI by default */
int EC_DigitalIn::read()
{
    val_t = GPIO_read(Port_t, pin_t);
    return val_t;
}

void EC_DigitalIn::pupdr(int _pupdr) {
    GPIO_pupdr(Port_t, pin_t, _pupdr);
}

void EC_DigitalOut::write(int _outVal)
{
    GPIO_write(Port_t, pin_t, _outVal);
}

void EC_DigitalOut::pupdr(int _pupdr) {
    GPIO_pupdr(Port_t, pin_t, _pupdr);
}

void EC_DigitalOut::otype(int _type) {
    GPIO_otype(Port_t, pin_t, _type);
}

void EC_DigitalOut::ospeed(int _speed) {
    GPIO_ospeed(Port_t, pin_t, _speed);
}

void EC_Ticker::Delay_ms(uint32_t msec) {
    delay_ms(msec);
}

uint32_t EC_Ticker::read_ms(void) {
    return SysTick->VAL &= 0xffffffff;
}

void EC_Ticker::reset(void) {
    SysTick_reset();
}

```

Figure 13. EC_API.c code

- Reference

6.3.12 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved											SPI5EN	Reserved	TIM11EN	TIM10EN	TIM9EN	
											rw		rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved	SYSCFGEN	SPI4EN	SPI1EN	SDIOEN	Reserved			ADC1EN	Reserved			USART6EN	USART1EN	Reserved		TIM1EN
	rw	rw	rw	rw				rw				rw	rw			rw

Bits 31:21 Reserved, must be kept at reset value.

Bit 20 **SPI5EN**: SPI5 clock enable

This bit is set and cleared by software

0: SPI5 clock disabled

1: SPI5 clock enabled

Bit 19 Reserved, must be kept at reset value.

Bit 18 **TIM11EN**: TIM11 clock enable

Set and cleared by software.

0: TIM11 clock disabled

1: TIM11 clock enabled

Bit 17 **TIM10EN**: TIM10 clock enable

Set and cleared by software.

0: TIM10 clock disabled

1: TIM10 clock enabled

Bit 16 **TIM9EN**: TIM9 clock enable

Set and cleared by software.

0: TIM9 clock disabled

1: TIM9 clock enabled

Bit 15 Reserved, must be kept at reset value.

Bit 14 **SYSCFGEN**: System configuration controller clock enable

Set and cleared by software.

0: System configuration controller clock disabled

1: System configuration controller clock enabled

Bit 13 **SPI4EN**: SPI4 clock enable

Set and reset by software.

0: SPI4 clock disabled

1: SPI4 clock enabled

Bit 12 **SPI1EN**: SPI1 clock enable

Set and cleared by software.

0: SPI1 clock disabled

1: SPI1 clock enabled

7.2.3 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
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 0 to 3)

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

0000: PA[x] pin

0001: PB[x] pin

0010: PC[x] pin

0011: PD[x] pin

0100: PE[x] pin

0101: Reserved

0110: Reserved

0111: PH[x] pin

7.2.4 SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 4 to 7)

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

0000: PA[x] pin

0001: PB[x] pin

0010: PC[x] pin

0011: PD[x] pin

0100: PE[x] pin

0101: Reserved

0110: Reserved

0111: PH[x] pin

7.2.5 SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 8 to 11)

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

0000: PA[x] pin

0001: PB[x] pin

0010: PC[x] pin

0011: PD[x] pin

0100: PE[x] pin

0101: Reserved

0110: Reserved

0111: PH[x] pin

7.2.6 SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 12 to 15)

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

0000: PA[x] pin

0001: PB[x] pin

0010: PC[x] pin

0011: PD[x] pin

0100: PE[x] pin

0101: Reserved

0110: Reserved

0111: PH[x] pin

10.3.3 Rising trigger selection register (EXTI_RTSTR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	Reserved		TR18	TR17	TR16
									r/w	r/w			r/w	r/w	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx**: Rising trigger event configuration bit of line x

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

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

10.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	TR22		TR21	20	19	18	17	16
Reserved									TR22	TR21	Reserved		TR18		TR17	TR16
									r/w	r/w			r/w	r/w	r/w	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **TRx**: Falling trigger event configuration bit of line x

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

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

Note: The external wakeup lines are edge triggered, no glitch must be generated on these lines. If a falling edge occurs on the external interrupt line while writing to the EXTI_FTSR register, the pending bit is not set.

Rising and falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

10.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved									MR22	MR21	Reserved			MR18	MR17	MR16
									rw	rw				rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **MRx**: Interrupt mask on line x

0: Interrupt request from line x is masked

1: Interrupt request from line x is not masked

10.3.6 Pending register (EXTI_PR)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PR22	PR21	Reserved		PR18	PR17	PR16
									rc_w1	rc_w1			rc_w1	rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 **PRx**: Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line.

This bit is cleared by programming it to '1'.

Figure 14. STM32F411xC/E Reference Manual

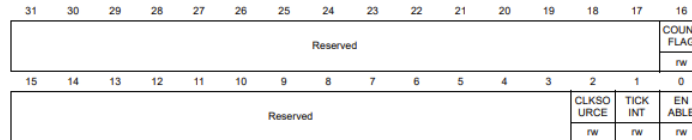
4.5.1 SysTick control and status register (STK_CTRL)

Address offset: 0x00

Reset value: 0x0000 0000

Required privilege: Privileged

The SysTick CTRL register enables the SysTick features.



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 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

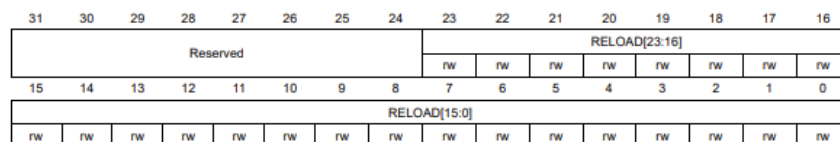
1: Counter enabled

4.5.2 SysTick reload value register (STK_LOAD)

Address offset: 0x04

Reset value: 0x0000 0000

Required privilege: Privileged



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:

- I 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.
- I 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.

4.5.3 SysTick current value register (STK_VAL)

Address offset: 0x08

Reset value: 0x0000 0000

Required privilege: Privileged

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

Bits 31:24 Reserved, must be kept cleared.

Bits 23:0 **CURRENT**: Current counter value

The VAL register contains the current value of the SysTick counter.

Reads return the current value of the SysTick counter.

A write of any value clears the field to 0, and also clears the COUNTFLAG bit in the STK_CTRL register to 0.

Figure 15. PM0214 Programming manual

- Documentation

EXTI_init()

Initializes external interrupt pins with default setting and Enables SYSCFG peripheral Clock.

```
void EXTI_init(GPIO_TypeDef *port, uint32_t pin, int trig_type, int priority);
```

Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **Trig_type**: FALL(0), RISE(1), BOTH(2)
- **priority**: interrupt priority

Example code

```
EXTI_init(GPIOC, BUTTON_PIN, FALL, 0) //BUTTON_PIN = 13 , priority = 0
```

EXTI_enable()

Enable external interrupt. unmasking corresponding bit in IMR register.

```
void EXTI_enable(uint32_t pin);
```

Parameters

- **pin**: pin number (int) 0~15

Example code

```
EXTI_enable(BUTTON_PIN) //BUTTON_PIN = 13
```

EXTI_disable()

Disable external interrupt. masking corresponding bit in IMR register.

```
void EXTI_disable(uint32_t pin);
```

Parameters

- **pin**: pin number (int) 0~15

Example code

```
EXTI_disable(BUTTON_PIN) //BUTTON_PIN = 13
```

is_pending_EXTI()

check the interrupt pending bit to excute interrupt

```
uint32_t is_pending_EXTI(uint32_t pin);
```

Parameters

- **pin:** pin number (int) 0-15

Example code

```
is_pending_EXTI(BUTTON_PIN) //BUTTON_PIN = 13
```

clear_pending_EXTI()

clear pending bit when external interrupt is finished.

```
void clear_pending_EXTI(uint32_t pin);
```

Parameters

- **pin:** pin number (int) 0-15

Example code

```
clear_pending_EXTI(BUTTON_PIN) //BUTTON_PIN = 13
```

Figure 16. documentation of EXTI

SysTick_init()

Initializes SysTick with default setting including reload counting value, interrupt priority, processor clock and so on.

```
void SysTick_init(uint32_t msec);
```

Parameters

- **msec:** Clock frequency

Example code

```
SysTick_init(PLL_CLK) //PLL_CLK = 84000000 (84MHz)
```

delay_ms()

delay msec interval.

```
void delay_ms(uint32_t msec);
```

Parameters

- **msec:** count value

Example code

```
delay_ms(1000) // down count 999 to 0
```

SysTick_val()

Read and return the current value of the SysTick counter.

```
uint32_t SysTick_val(void);
```

Parameters

- void

Example code

```
Start_time = SysTick_val(); // read start count value  
Stop_time = SysTick_val(); // read stop count value
```

SysTick_reset()

Reset the count value to be zero.

```
void SysTick_reset(void);
```

Parameters

- void

Example code

```
SysTick_reset();
```

SysTick_enable()

Enable the counter.

```
void SysTick_enable(void);
```

Parameters

- void

Example code

```
SysTick_enable();
```

SysTick_disable(void)

Disable the counter.

```
void SysTick_disable(void);
```

Parameters

- void

Example code

```
SysTick_disable();
```

Figure 17. documentation of SysTick

EC_Ticker::Delay_ms()

Give a delay milliseconds.

```
void EC_Ticker::Delay_ms(uint32_t msec);
```

Parameters

- msec : delay time

Example code

```
EC_Ticker tick(1);
volatile uint32_t count = 0;

//Initialization
void setup(void)
{
    RCC_PLL_init();
    sevensegment_init();
}

int main (void){
    setup();
    while(1){
        sevensegment_decoder(count);
        tick.Delay_ms(1000);
        count++;
        if(count == 10)count =0;
        tick.reset();
    }
}
```

EC_Ticker::read_ms()

Read and return the current value of the SysTick counter.

```
uint32_t EC_Ticker::read_ms(void);
```

Parameters

- void

Example code

```
Start_time = tick.read_ms(); // read start count value
Stop_time = tick.read_ms(); // read stop count value
```

EC_Ticker::reset()

Reset the count value to be zero.

```
void EC_Ticker::reset(void);
```

Parameters

- void

Example code

```
tick.reset();
```

Figure 18. documentation of API