LAB: EXTI & SysTick

Date: 2023-10-16

Author: EunChan Kim

ID: 21801017

Demo Video: EXIT, SysTick

Introduction

In this experiment, we will create an interrupt program using EXIT and SysTick to increment the number on a 7-segment display either when a button is pressed or every second.

Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - 4 LEDs and load resistance
  - 7-segment display(5101ASR)
  - Array resistor (330 ohm)
  - breadboard

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: Counting numbers on 7-Segment using EXTI Button

Procedure

- Complete the EXTI code.
- Use the decoder chip (**74LS47**). Connect it to the bread board and 7-segment display.
- Check if every number, 0 to 9, can be displayed properly on the 7-segment.
- Program the numbers to increase from 0 to 9 when the button is pressed. This should be done using EXTI library functions.

Configuration

Digital In for Button (B1)	Digital Out for 7-Segment decoder
Digital In	Digital Out
PC13	PA7, PB6, PC7, PA9
PULL-UP	Push-Pull, No Pull-up-Pull-down, Medium Speed

Circuit Diagram

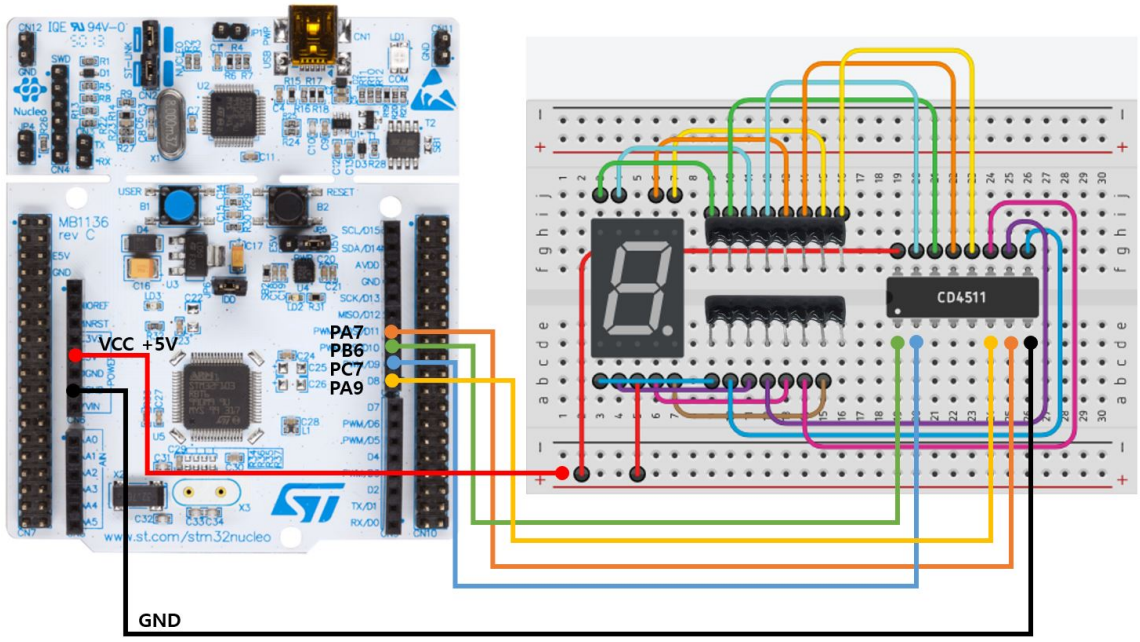


Figure1. 7-Segment Circuit with Decoder

Discussion

1. We can use two different methods to detect an external signal: polling and interrupt. What are the advantages and disadvantages of each approach?

Polling and Interrupt both share the common aspect of detecting external signals, but they also have differences.

Polling

Advantages:

- It has a simple and easily understandable design.
- You can predict exactly when data will be processed at a specific time.

Disadvantages:

- It consumes a lot of CPU time and uses resources inefficiently because the CPU needs to continuously check the status.
- Since the exact arrival time of the signal cannot be predicted, there can be significant delays between signal arrivals.

Interrupt

Advantages:

- It only notifies the CPU when there is an external signal, allowing the CPU to focus on other tasks.
- It offers good real-time performance, enhancing responsiveness to critical tasks that require immediate processing.

Disadvantages:

- It has a complex design and management. Additional code, such as ISRs, needs to be managed.
- If multiple interrupt requests occur simultaneously, additional logic, including priority determination, is necessary

2. What would happen if the EXTI interrupt handler does not clear the interrupt pending flag? Check with your code

When an EXTI interrupt handler does not clear the interrupt pending flag, the interrupt remains in a pending state. This prevents the system from recognizing that the interrupt service routine (ISR) has been completed. Therefore, even after the ISR has terminated, if the corresponding flag is still set, the system will attempt to call the ISR for the same interrupt.

This can lead to unnecessary CPU cycles being consumed and can potentially disrupt the processing of new interrupt events in other parts of the system. In severe cases, this repetitive handling of interrupts can result in the system not functioning properly.

Code

LAB_EXTI.c

Each time the button is pressed, `EXTI15_10_IRQHandler(void)` is executed, and it displays the corresponding number based on the count. `EXTI15_10_IRQHandler(void)` checks whether there is a pending EXTI event for the button. If there is a pending event, it increments the count and activates the 7-segment display. Finally, it clears the pending EXTI event using `clear_pending_EXTI(BUTTON_PIN)`, allowing the next interrupt to proceed.

```
int cnt = 0;

void EXTI15_10_IRQHandler(void);
void setup(void);

int main(void) {
    // Initialiization --------------------------------------------------------
    setup();

    // Inifinite Loop ---------------------------------------------------------
    while(1){}
}

// Initialiization
void setup(void)
```

```
{
    RCC_HSI_init();
    sevensegment_display_init();
// LED configuration
    EXTI_init(GPIOC,BUTTON_PIN,FALL,0);             // EXTI initialization ,
button pin, priority : 0
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);            // BUTTON_PIN : INPUT
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);            // BUTTON_PIN : PULL_UP

}

void EXTI15_10_IRQHandler(void) {

    if (is_pending_EXTI(BUTTON_PIN)) {

        cnt ++;
        sevensegment_display(cnt%10);
        for(volatile int i = 0; i < 500000; i++){}
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }

}
```

ecEXTI.c

EXTI functions have been completed by consulting the programming manual. `EXTI_init()` configures and initializes an EXTI for a specified GPIO pin. It sets the trigger type, enables the EXTI interrupt, and specifies the priority for NVIC. It connects the EXTI line to the appropriate GPIO port and handles SYSCFG peripheral configuration.

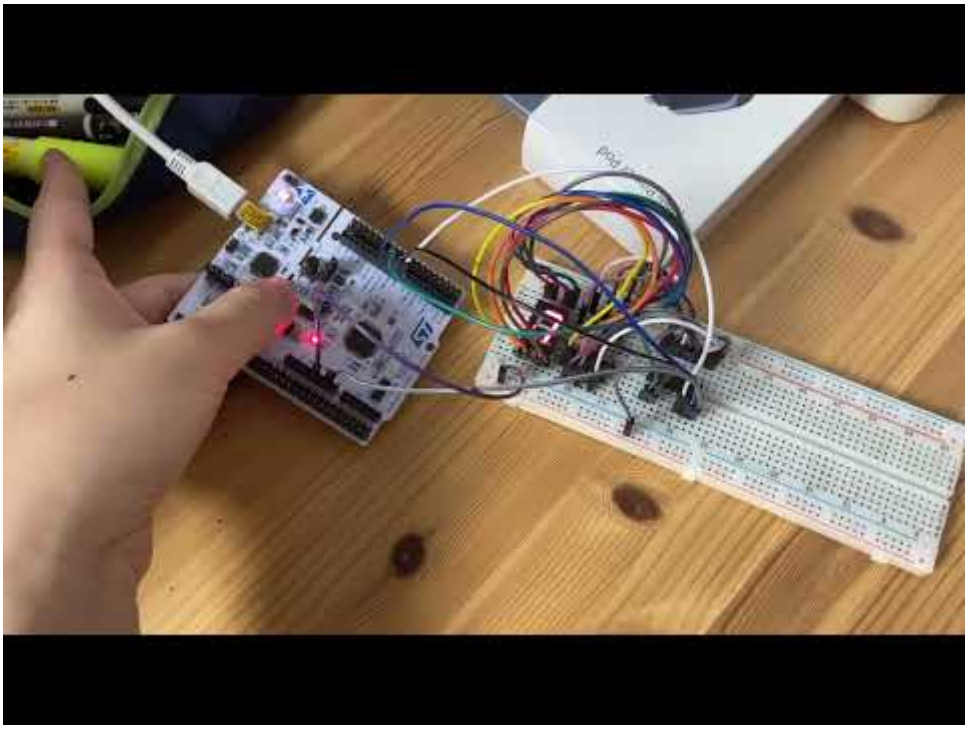
```
void EXTI_init(GPIO_TypeDef *Port, int Pin, int trig_type,int priority){

    // SYSCFG peripheral clock enable
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Connect External Line to the GPIO
    unsigned int EXTICR_port;
    if      (Port == GPIOA) EXTICR_port = 0;
    else if (Port == GPIOB) EXTICR_port = 1;
    else if (Port == GPIOC) EXTICR_port = 2;
    else if (Port == GPIOD) EXTICR_port = 3;
    else                                        EXTICR_port = 4;

    SYSCFG->EXTICR[Pin/4] &= ~(15<<(4*(Pin%4)));            // clear 4 bits
    SYSCFG->EXTICR[Pin/4] |= EXTICR_port << 4*(Pin%4);          // set 4 bits,
connect port number

    // Configure Trigger edge
    if (trig_type == FALL) EXTI->FTSR |= 1UL << Pin;   // Falling trigger enable
    else if (trig_type == RISE) EXTI->RTSR |= 1UL << Pin;   // Rising trigger
enable
    else if (trig_type == BOTH) {           // Both falling/rising trigger
enable
        EXTI->RTSR |= 1UL << Pin;
        EXTI->FTSR |= 1UL << Pin;
    }
```

```
    // Configure Interrupt Mask (Interrupt enabled)
    EXTI->IMR  |= 1UL << Pin;     // not masked

    // NVIC(IRQ) Setting
    int EXTI_IRQn = 0;

    if (Pin < 5)    EXTI_IRQn = Pin + EXTI0_IRQn;
    else if (Pin < 10)  EXTI_IRQn = EXTI9_5_IRQn;
    else            EXTI_IRQn = EXTI15_10_IRQn;

    NVIC_SetPriority(EXTI_IRQn, priority);  // EXTI priority
    NVIC_EnableIRQ(EXTI_IRQn);  // EXTI IRQ enable
}

void EXTI_enable(uint32_t pin) {
    EXTI->IMR |= 1UL << pin;     // not masked (i.e., Interrupt enabled)
}
void EXTI_disable(uint32_t pin) {
    EXTI->IMR &= ~(1 << pin);      // masked (i.e., Interrupt disabled)
}

uint32_t is_pending_EXTI(uint32_t pin){
    uint32_t EXTI_PRx = 1 << pin;       // check  EXTI pending
    return ((EXTI->PR & EXTI_PRx) == EXTI_PRx);
}

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

Results

As you can see in the video, the EXTI handler is triggered every time the button is pressed, and you can observe the numbers displayed on the 7-segment display.

Click Below



Problem 2: Counting numbers on 7-Segment using SysTick

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, the number should be reset ‘0’ and start counting again.

Procedure

- Complete the SysTick code.
- Use the decoder chip (**74LS47**). Connect it to the bread board and 7-segment display.
- Check if every number, 0 to 9, can be displayed properly on the 7-segment.
- Create a code to display the number counting from 0 to 9 and repeats at the rate of 1 second.
- Add code that resets to 0 when the button is pressed.

Configuration

Digital In for Button (B1)	**Digital Out for 7-Segment decoder**
Digital In	Digital Out
PC13	PA7, PB6, PC7, PA9
PULL-UP	Push-Pull, No Pull-up-Pull-down, Medium Speed

Circuit Diagram

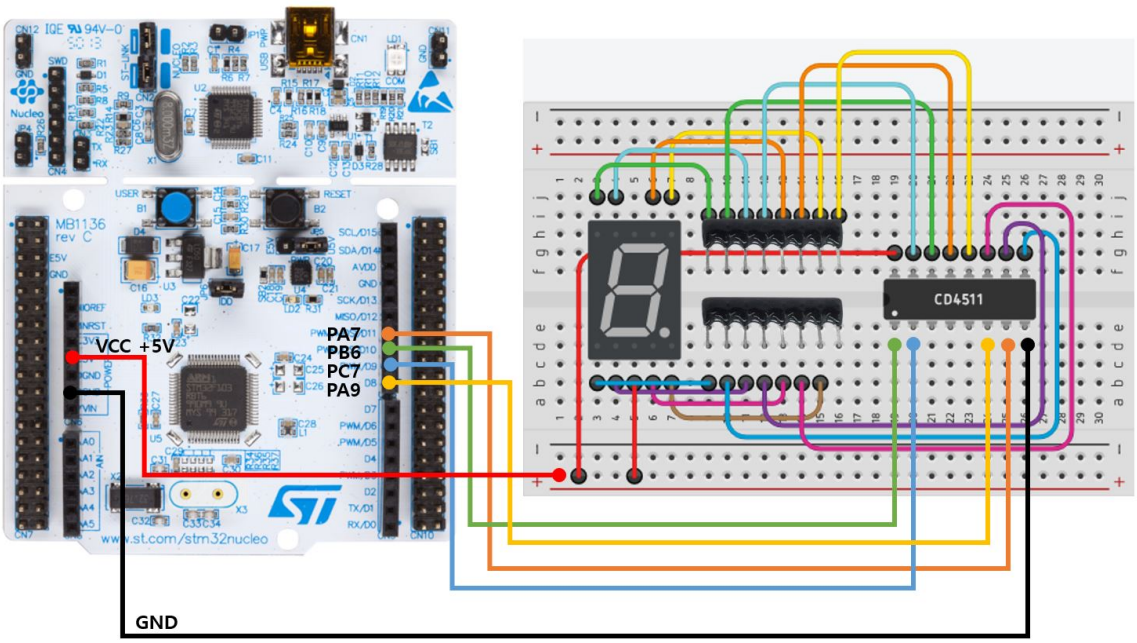


Figure 2. 7-Segment Circuit with Decoder

Code

LAB_EXTI_SysTick.c

In a `while` loop, we utilized the `delay_ms(1000)` function to increase the count every second, resulting in an increment in the 7-segment display value. Furthermore, with the `EXTI15_10_IRQHandler(void)` handler, pressing the button resets the count to 0, initiating the counting process.

```
int cnt = 0;

void setup(void);
void EXTI15_10_IRQHandler(void);

int main(void) {
    // Initialiization --------------------------------------------------------
        setup();

    // Inifinite Loop ---------------------------------------------------------
    while(1){
        sevensegment_display(cnt%10);
        delay_ms(1000);
        cnt ++;

        if (cnt >9){
            cnt =0;
        }
        SysTick_reset();
    }
}

void setup(void){
```

```
    RCC_PLL_init();
    SysTick_init();
    sevensegment_init();
    EXTI_init(GPIOC,BUTTON_PIN,FALL,0);             // EXTI initialization ,
button pin, priority : 0
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);            // BUTTON_PIN : INPUT
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);            // BUTTON_PIN : PULL_UP

}

void EXTI15_10_IRQHandler(void) {

    if (is_pending_EXTI(BUTTON_PIN)) {

        cnt = 0;
        sevensegment_display(cnt%10);
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }
}
```

ecSysTick.c

SysTick functions have been completed by consulting the programming manual.

First, the SysTick IRQ and SysTick counter are disabled using the `SysTick->CTRL` register. The processor clock is selected by setting the `SysTick_CTRL_CLKSOURCE_Msk` bit in the `SysTick->CTRL` register. The `SysTick->LOAD` register is then used to set the reload value for the SysTick timer, which is calculated to generate an interrupt every 1 millisecond (1ms). The `SysTick->VAL` register is set to 0 to initialize the current SysTick value. Using the `SysTick->CTRL` register, SysTick interrupts are enabled, and the SysTick timer is activated. Finally, the priority of the SysTick interrupt is set in the NVIC, and the interrupt is enabled in the NVIC.

This configured SysTick timer generates interrupts every 1 millisecond based on the processor clock and is commonly used for time measurement and delay functions.

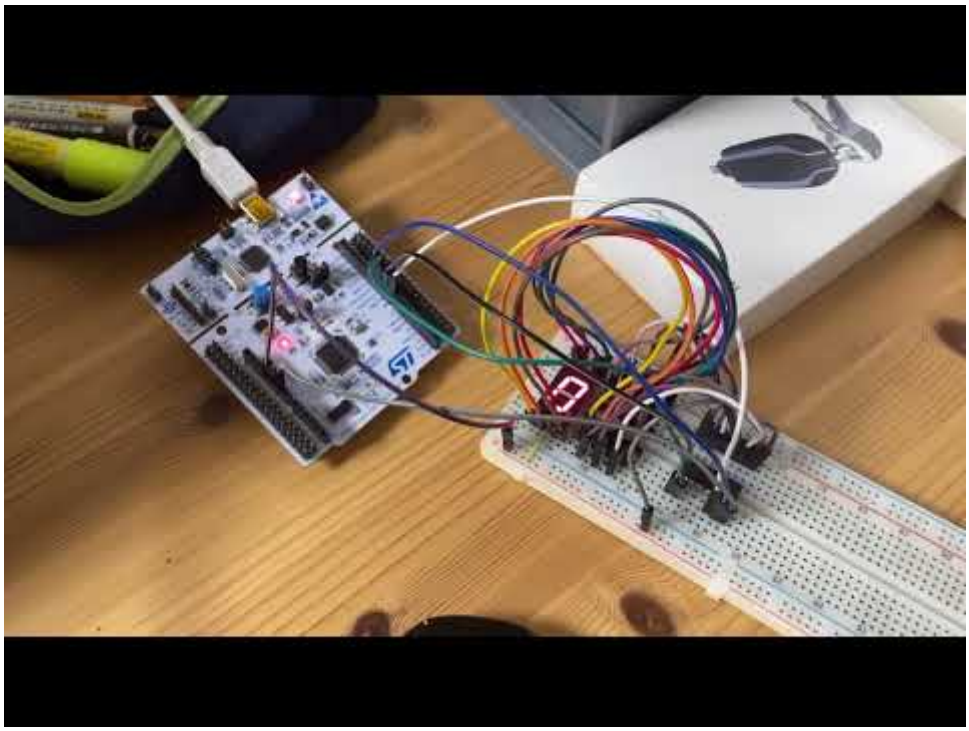
```
#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000

volatile uint32_t msTicks=0;

//EC_SYSTEM_CLK

void SysTick_init(void){
    //  SysTick Control and Status Register
    SysTick->CTRL = 0;                                          // Disable
SysTick IRQ and SysTick Counter

    // Select processor clock
    // 1 = processor clock;  0 = external clock
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

    // uint32_t MCU_CLK=EC_SYSTEM_CLK
    // SysTick Reload Value Register
```

```
    SysTick->LOAD = MCU_CLK_PLL / 1000 - 1;                    // 1ms, for HSI
PLL = 84MHz.

    // SysTick Current Value Register
    SysTick->VAL = 0;

    // Enables SysTick exception request
    // 1 = counting down to zero asserts the SysTick exception request
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    // Enable SysTick IRQ and SysTick Timer
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

    NVIC_SetPriority(SysTick_IRQn, 16);     // Set Priority to 1
    NVIC_EnableIRQ(SysTick_IRQn);           // Enable interrupt in NVIC
}

void SysTick_Handler(void){
    SysTick_counter();
}

void SysTick_counter(){
    msTicks++;
}


void delay_ms (uint32_t mesc){
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < mesc);

    msTicks = 0;
}

void SysTick_reset(void)
{
    // SysTick Current Value Register
    SysTick->VAL = 0;
}

uint32_t SysTick_val(void) {
    return SysTick->VAL;
}

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

void SysTick_disable (void){
    SysTick->CTRL |= 0UL << 0U;
}
```

Results

As you can see in the video, the count increases every 1 second, and when the button is pressed, it resets to 0, starting again.

Click Below



Reference

- STM32 Cortex®-M4 MCUs and MPUs programming manual Download Link