

การทดลองที่ 4 การใช้งาน NVIC และ EXTI

วัตถุประสงค์

- 1) เข้าใจการทำงานของ Nested Vectored Interrupt Controller
- 2) สามารถเขียนโปรแกรมควบคุมการทำงานของ External Interrupt

1. Priority Interrupt

Interrupt คือการทำให้ไมโครคอนโทรลเลอร์หยุดทำงานชั่วคราวเพื่อไปตอบสนองต่อสัญญาณ interrupt ที่เกิดขึ้น เช่น สัญญาณ External Interrupt (EXTI) ทางขา GPIO เป็นต้น ภายหลังจากการตอบสนองสัญญาณ interrupt เสร็จสิ้นลง ไมโครคอนโทรลเลอร์จะกลับไปทำงานเดิมต่อ

Nested Vectored Interrupt Controller หรือ NVIC คือโมดูลที่อยู่ภายในไมโครคอนโทรลเลอร์ทำหน้าที่ควบคุมการตั้งค่าและการตอบสนองต่อสัญญาณ interrupt ไมโครคอนโทรลเลอร์สามารถรองรับสัญญาณ interrupt ได้หลายแหล่ง ซึ่งจะต้องมีการกำหนดระดับความสำคัญให้กับสัญญาณ interrupt แต่ละแหล่งด้วย เพื่อการจัดการเวลาที่สัญญาณ interrupt เกิดขึ้นพร้อมกันหลายสัญญาณ หรือกรณีที่เกิดสัญญาณ interrupt แทรกเข้ามาขณะที่ไมโครคอนโทรลเลอร์กำลังตอบสนองต่อสัญญาณ interrupt ที่เกิดก่อนหน้านี้

ARM ได้ออกแบบให้ Cortex M4 มีรีจิสเตอร์เพื่อใช้กำหนดระดับความสำคัญของสัญญาณ interrupt ขนาด 8 บิต ทั้งนี้ผู้ผลิตแต่ละรายสามารถกำหนดให้มีการใช้งานน้อยกว่า 8 บิตได้ เช่น ไอซี STM32F429 ของบริษัท STMicroelectronics นั้น ใช้เพียง 4 บิตของรีจิสเตอร์เพื่อกำหนดระดับความสำคัญของ interrupt จากแต่ละแหล่ง การใช้งานจะแบ่ง 4 บิตของรีจิสเตอร์ออกเป็น 2 ส่วน ได้แก่ PreemptionPriority และ SubPriority ทำให้เกิดการจัดกลุ่มได้ 5 รูปแบบ เรียกว่า NVIC_PriorityGroup_0 ถึง NVIC_PriorityGroup_4 รายละเอียดของแต่ละกลุ่มสรุปได้ดังตารางที่ 1.1

ตารางที่ 1.1 แสดงรายละเอียดของ NVIC_PriorityGroup แต่ละกลุ่ม

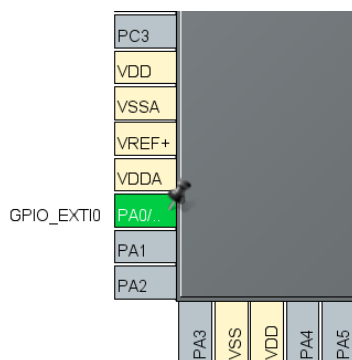
NVIC_PriorityGroup	PreemptionPriority		SubPriority	
	จำนวนบิต	ค่าเป็นไปได้	จำนวนบิต	ค่าเป็นไปได้
NVIC_PriorityGroup_0	0	0	4	0-15
NVIC_PriorityGroup_1	1	0-1	3	0-7
NVIC_PriorityGroup_2	2	0-3	2	0-3
NVIC_PriorityGroup_3	3	0-7	1	0-1
NVIC_PriorityGroup_4	4	0-15	0	0

โดยตัวเลข 0 แสดงถึงระดับความสำคัญมากที่สุด สัญญาณ interrupt ที่มีค่า PreemptionPriority ต่ำกว่า (มีความสำคัญมากกว่า) สามารถ interrupt แทรกสัญญาณ interrupt ที่มีค่า PreemptionPriority มากกว่า (มีความสำคัญน้อยกว่า) ซึ่งกำลังได้รับการตอบสนองจากไมโครคอนโทรลเลอร์อยู่ได้

หากเกิดสัญญาณ interrupt สองสัญญาณพร้อมกัน และทั้งสองสัญญาณนั้นมี PreemptionPriority เท่ากัน สัญญาณที่ถูกกำหนดให้มีค่า SubPriority ต่ำกว่าจะได้รับการตอบสนองก่อน

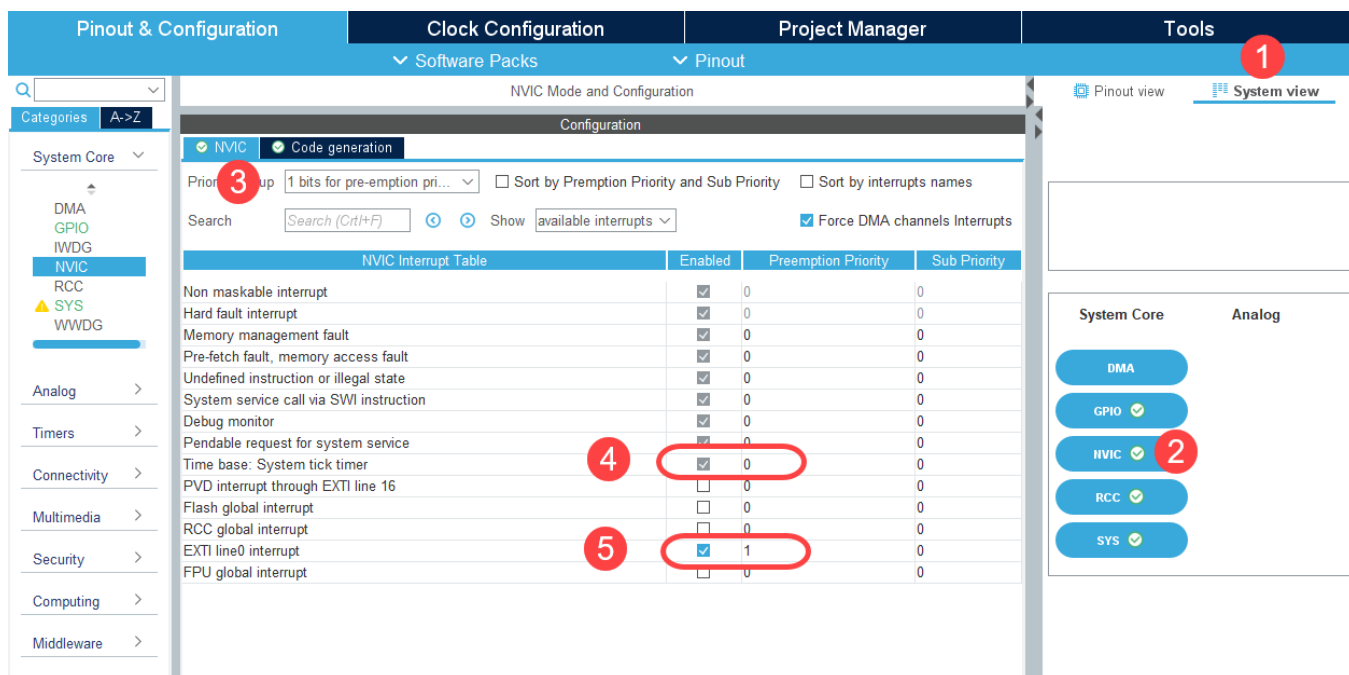
2. การตั้งค่าในโปรแกรม STM32CubeMX

การตั้งค่าสำหรับการทดลองครั้งนี้แบ่งออกเป็น 2 ส่วน ได้แก่ NVIC และ EXTI โดยเริ่มต้นที่แท็บ Pinout ในโปรแกรม STM32CubeMX กำหนดให้ขา PA0 ซึ่งเชื่อมต่อกับสวิตช์ B1 บนบอร์ด ทำหน้าที่เป็นตัวรับสัญญาณจากภายนอกหมายเลข 0 (EXTI0) ดังรูปที่ 2.1 จากนั้นตั้งค่าความถี่ของสัญญาณนาฬิกาตามการทดลองก่อนหน้า

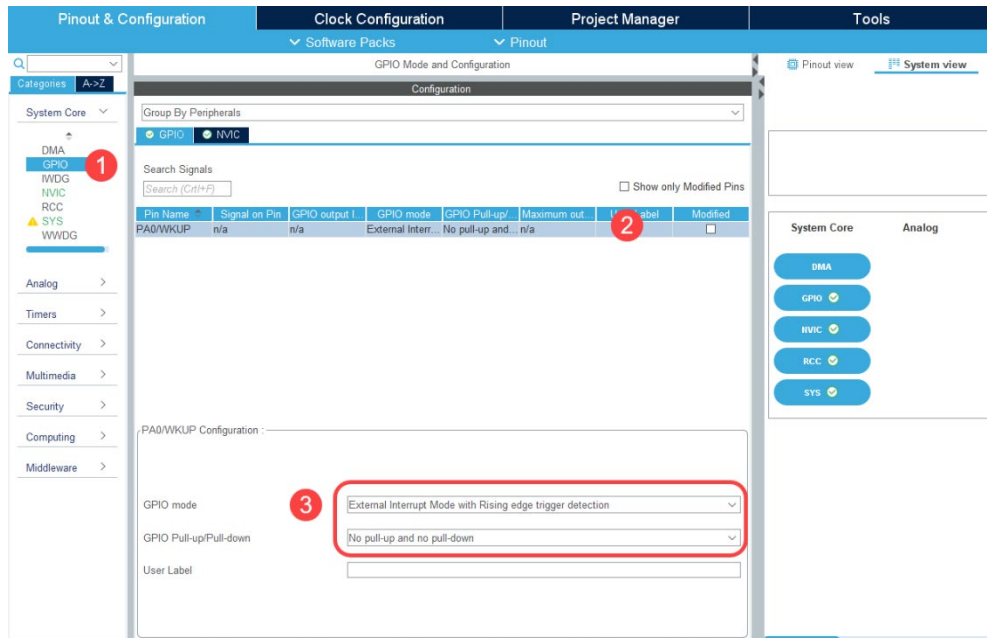


รูปที่ 2.1 แสดงการตั้งค่าให้ PA0 ทำหน้าที่ EXTI0

จากนั้นตั้งค่า NVIC กลุ่ม 1 คือมี PreemptionPriority 1 บิต และ SubPriority 3 บิต ดังรูปที่ 2.2 แล้วตั้งค่าให้ PA0 ทำหน้าที่ตรวจจับสัญญาณที่เข้ามาเพื่อสร้างสัญญาณ interrupt ไปยัง NVIC โดยกำหนดให้เป็นขาอินพุตแบบ floating และตรวจจับหากสัญญาณเปลี่ยนจากลอจิก 0 เป็นลอจิก 1 หรือตรวจจับขอบขาขึ้น (Rising Edge) ของสัญญาณที่เข้ามายังขา PA0 ดังรูปที่ 2.3



รูปที่ 2.2 แสดงการตั้งค่า NVIC_PriorityGroup_1



รูปที่ 2.3 แสดงการตั้งค่า PA0 ให้ทำหน้าที่ EXTI0 โดยตรวจสอบข้อขาขึ้นของสัญญาณที่เข้ามา

3. อธิบายการทำงานของ NVIC

โค้ดการตั้งค่า NVIC เพื่อควบคุมสัญญาณ interrupt ที่สร้างจากโปรแกรม STM32CubeMX จะอยู่ในฟังก์ชัน HAL_MspInit() ในไฟล์ stm32f4xx_hal_msp.c ดังรูปที่ 3.1

```
void HAL_MspInit(void)
{
    /* USER CODE BEGIN MspInit 0 */

    /* USER CODE END MspInit 0 */

    __HAL_RCC_SYSCFG_CLK_ENABLE();
    __HAL_RCC_PWR_CLK_ENABLE();

    HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_1);

    /* System interrupt init*/

    /* USER CODE BEGIN MspInit 1 */

    /* USER CODE END MspInit 1 */
}
```

รูปที่ 3.1 แสดงการตั้งค่า Group Priority ในฟังก์ชัน HAL_MspInit() ในไฟล์ stm32f4xx_hal_msp.c

```
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOA_CLK_ENABLE();

    /*Configure GPIO pin : PA0 */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* EXTI interrupt init*/
    HAL_NVIC_SetPriority(EXTI0_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}
```

รูปที่ 3.2 แสดงการตั้งค่าให้ PA0 ทำหน้าที่ EXTI0 ในฟังก์ชัน MX_GPIO_Init() ในไฟล์ gpio.c

ส่วนการตั้งค่า PreemptionPriority และ SubPriority จะอยู่ในฟังก์ชัน MX_GPIO_Init() ในไฟล์ gpio.c ดังรูปที่ 3.2 มีรายละเอียดดังนี้

ฟังก์ชัน MX_GPIO_init ()

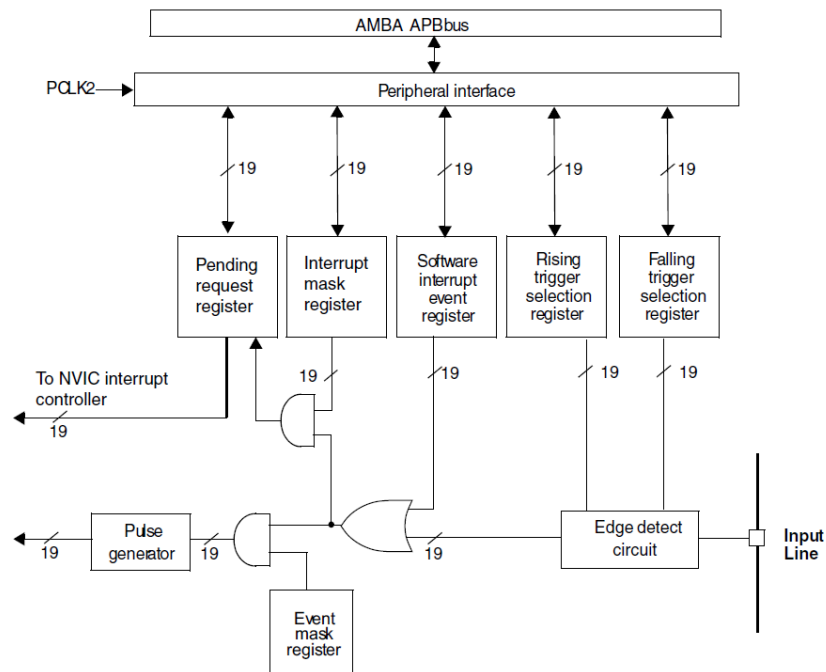
- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อดำเนินการตั้งค่า GPIO บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับที่กำหนดไว้ในโปรแกรม
- เริ่มต้นด้วยการ Enable สัญญาณนาฬิกาให้ GPIOA (สำหรับสวิตช์ B1)
`__GPIOA_CLK_ENABLE();`
- กำหนดให้ PA0 ทำหน้าที่ EXTI0 โดยกำหนดให้ทำงานเป็นอินพุต floating และจะสร้างสัญญาณ interrupt ไปยัง NVIC เมื่อตรวจพบขอบขึ้นลงของสัญญาณที่รับเข้ามา (มีการกดสวิตช์ B1)

```
GPIO_InitStruct.Pin    = GPIO_PIN_0;  
GPIO_InitStruct.Mode    = GPIO_MODE_IT_RISING;  
GPIO_InitStruct.Pull    = GPIO_NOPULL;  
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

- กำหนดระดับความสำคัญให้กับ EXTI0 ซึ่งกำหนดให้มี PreemptionPriority = 1 และ SubPriority = 0 พร้อมสั่งให้เริ่มต้นการทำงาน
`HAL_NVIC_SetPriority(EXTI0_IRQn, 1, 0);`
`HAL_NVIC_EnableIRQ(EXTI0_IRQn);`

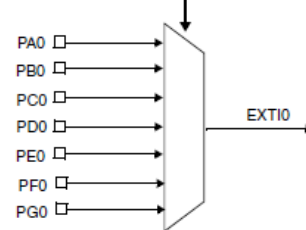
4. EXTI

External Interrupt หรือ EXTI คือโมดูลภายในไมโครคอนโทรลเลอร์ที่ทำหน้าที่ตรวจจับสัญญาณอินพุตที่เข้ามาที่ขา GPIO จากนั้นจะสร้างสัญญาณ interrupt ไปยัง NVIC เมื่อสัญญาณที่เข้ามาตรงตามเงื่อนไขที่ตั้งไว้ ได้แก่ เมื่อสัญญาณเกิดขอบขาขึ้น ขอบขาลง หรือทั้งขอบขาขึ้นและขอบขาลง โครงสร้างของ EXTI แสดงได้ดังรูปที่ 4.1 และแสดงการเชื่อมต่อ GPIO กับโมดูล EXTI ได้ดังรูปที่ 4.2 ซึ่งขณะใดขณะหนึ่งจะมีเพียงขา GPIO เพียงขาเดียวเท่านั้นที่ทำหน้าที่รับสัญญาณอินพุตแล้วส่งต่อไปยังโมดูล EXTI แต่ละหมายเลข

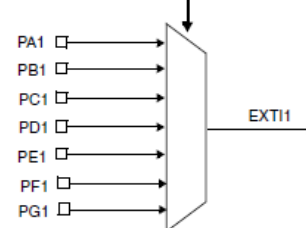


รูปที่ 4.1 แสดงโครงสร้างของโมดูล External Interrupt

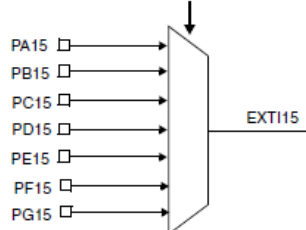
EXTI0[3:0] bits in AFIO_EXTICR1 register



EXTI1[3:0] bits in AFIO_EXTICR1 register



EXTI15[3:0] bits in AFIO_EXTICR4 register



รูปที่ 4.2 แสดงการเชื่อมต่อ GPIO ไปยังโมดูล EXTI

5. Interrupt Service Routine

Interrupt Service Routine (ISR) หรือ Interrupt Handler คือ โปรแกรมที่ทำหน้าที่ตอบสนองต่อสัญญาณ interrupt ที่เข้ามา เมื่อหน่วยประมวลผลได้รับสัญญาณ interrupt จาก NVIC หน่วยประมวลผลจะหยุดการทำงานของโปรแกรมปัจจุบันลงชั่วคราว แล้วเปลี่ยนไปทำงานยัง ISR ที่เกี่ยวข้องกับสัญญาณ interrupt ที่เข้ามา โดยหาตำแหน่งของ ISR ในหน่วยความจำจาก Vector Table เมื่อทำงาน ISR เสร็จแล้วหน่วยประมวลผลก็จะกลับมาทำงานที่ทำค้างอยู่ก่อนที่จะเกิดสัญญาณ interrupt

ตัวอย่างเช่น หากกำหนดการตั้งค่า NVIC และ EXTI ดังรูปที่ 2.1, รูปที่ 2.2 และรูปที่ 2.3 เมื่อสวิตช์ B1 (PA0) ถูกกด จะเกิดสัญญาณ interrupt จากโมดูล EXTI0 ไปยังหน่วยประมวลผล หน่วยประมวลผลจะหยุดการทำงานปัจจุบันลงแล้วไปทำงานที่ฟังก์ชัน `EXTI0_IRQHandler()` ซึ่งเป็น ISR ของ EXTI0 interrupt

สำหรับฟังก์ชัน `EXTI0_IRQHandler()` ในไฟล์ `stm32f4xx_it.c` เป็น ISR ที่ได้กำหนดไว้แล้วล่วงหน้าของสัญญาณ Interrupt `EXTI0_IRQn` ซึ่งเชื่อมต่อกับ EXTI0 โดยชื่อฟังก์ชันจะสัมพันธ์กับการประกาศ Vector Table ในไฟล์ `startup_stm32f429zitx.s` ด้วยภาษา Assembly ดังรูปที่ 5.1 สำหรับสัญญาณ EXTI หมายเลขอื่นๆ ก็จะมีฟังก์ชัน ISR ดังตารางที่ 5.1

ตารางที่ 5.1 แสดงฟังก์ชัน ISR ของ EXTI แต่ละหมายเลข

หมายเลข EXTI	ชื่อสัญญาณ Interrupt	ชื่อฟังก์ชัน ISR	หมายเหตุ
EXTI0	EXTI0_IRQn	EXTI0_IRQHandler	-
EXTI1	EXTI1_IRQn	EXTI1_IRQHandler	-
EXTI2	EXTI2_IRQn	EXTI2_IRQHandler	-
EXTI3	EXTI3_IRQn	EXTI3_IRQHandler	-
EXTI4	EXTI4_IRQn	EXTI4_IRQHandler	-
EXTI5 - EXTI9	EXTI9_5_IRQn	EXTI9_5_IRQHandler	EXTI5 ถึง EXTI9 ใช้ ISR ร่วมกัน
EXTI10 - EXTI15	EXTI15_10_IRQn	EXTI15_10_IRQHandler	EXTI10 ถึง EXTI15 ใช้ ISR ร่วมกัน

```
/* External Interrupts */
.word WWDG_IRQHandler          /* Window WatchDog */
.word PVD_IRQHandler           /* PVD through EXTI Line detection */
.word TAMPER_IRQHandler        /* Tamper and TimeStamps through the EXTI line */
.word RTC_WKUP_IRQHandler      /* RTC Wakeup through the EXTI line */
.word FLASH_IRQHandler         /* FLASH */
.word RCC_IRQHandler           /* RCC */
.word EXTI0_IRQHandler         /* EXTI Line0 */
.word EXTI1_IRQHandler         /* EXTI Line1 */
.word EXTI2_IRQHandler         /* EXTI Line2 */
.word EXTI3_IRQHandler         /* EXTI Line3 */
.word EXTI4_IRQHandler         /* EXTI Line4 */
.word DMA1_Stream0_IRQHandler  /* DMA1 Stream 0 */
.word DMA1_Stream1_IRQHandler  /* DMA1 Stream 1
```

รูปที่ 5.1 แสดงการกำหนด Vector Table

รูปที่ 5.2 แสดงตัวอย่าง ISR ของ `EXTI0_IRQn` ซึ่งรวม EXTI0 อยู่ด้วย โดยจะทำงานเมื่อสวิตช์ B1 ถูกกดซึ่งจะ toggle LED LD1 ที่ขา PG13 ส่วนฟังก์ชัน `HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0)` ที่ถูกเรียกใช้ในฟังก์ชันนี้เป็นการตรวจสอบ การเคลียร์บิต Interrupt Pending เพื่อยกเลิกสัญญาณ Interrupt และเรียกฟังก์ชัน Callback ตามผลการทำงานของ ISR ดังรูปที่ 5.3

```

void EXTI0_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI0_IRQn 0 */

    /* USER CODE END EXTI0_IRQn 0 */
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
    /* USER CODE BEGIN EXTI0_IRQn 1 */
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_13);
    /* USER CODE END EXTI0_IRQn 1 */
}

```

รูปที่ 5.2 แสดง Interrupt Service Routine ของ EXTI0_IRQn

```

void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
    /* EXTI line interrupt detected */
    if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
    {
        __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
        HAL_GPIO_EXTI_Callback(GPIO_Pin);
    }
}

```

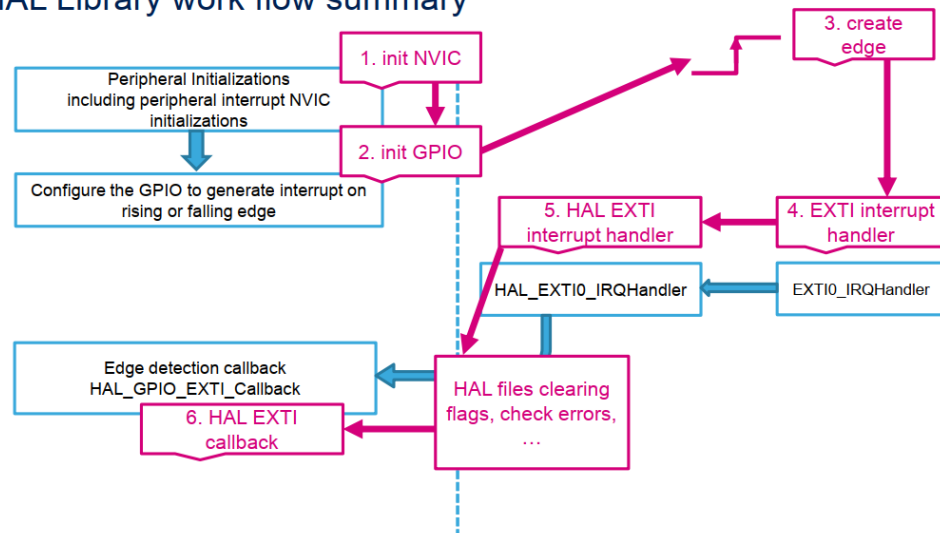
รูปที่ 5.3 แสดงรายละเอียดภายในฟังก์ชัน HAL_GPIO_EXTI_IRQHandler()

6. ฟังก์ชัน Callback

ฟังก์ชัน Callback เป็นฟังก์ชันที่ถูกเรียกจากภายใน ISR เพื่อให้การตอบสนองต่อ interrupt ดำเนินไปอย่างสมบูรณ์ตามสถานะการทำงานของ interrupt เช่น การตอบสนองต่อ interrupt ที่กำลังเกิดขึ้นนั้นว่าทำสำเร็จหรือมีข้อผิดพลาด โดยแสดงลำดับกระบวนการตอบสนองต่อ interrupt ของ EXTI ได้ดังรูปที่ 6.1 ซึ่งมีฟังก์ชัน Callback ฟังก์ชันเดียว ได้แก่ `HAL_GPIO_EXTI_Callback()`

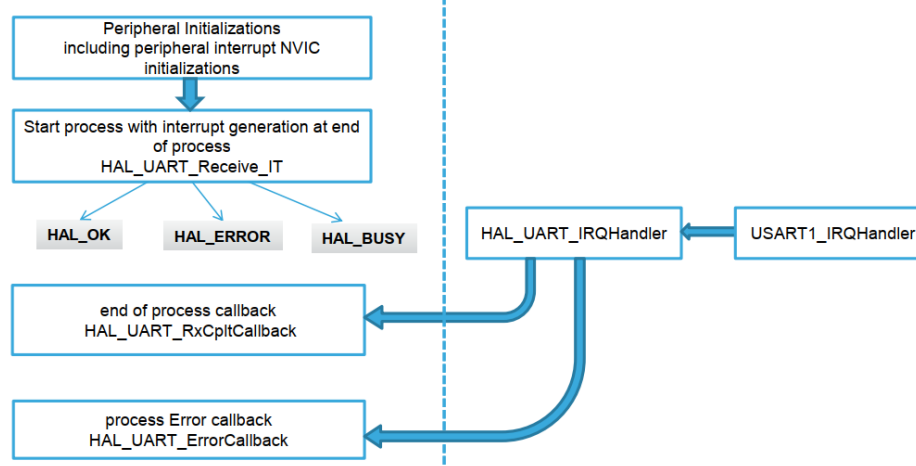
ส่วนรูปที่ 6.2 แสดงลำดับกระบวนการตอบสนองต่อ interrupt ของการรับข้อมูลผ่าน UART ซึ่งมีฟังก์ชัน Callback ฟังก์ชันจำนวน 2 ฟังก์ชัน ได้แก่ ฟังก์ชัน `HAL_UART_RxCpltCallback()` ที่จะถูกเรียกเมื่อการรับข้อมูลทำงานได้เสร็จสมบูรณ์ แต่ถ้าหากการรับข้อมูลพบข้อผิดพลาดฟังก์ชัน Callback ที่ถูกเรียกจะเป็นฟังก์ชัน `HAL_UART_ErrorCallback()` แทน

HAL Library work flow summary



รูปที่ 6.1 แสดงลำดับกระบวนการตอบสนองต่อ interrupt ของโมดูล EXTI0

HAL Library UART with IT receive flow



รูปที่ 6.2 แสดงลำดับกระบวนการตอบสนองต่อ interrupt ของโมดูล USART1

ตามปกติ STM32CubeMX จะสร้างฟังก์ชัน Callback เป็นฟังก์ชันแบบ weak type ดังรูปที่ 6.3 โดยใช้ weak symbol นำหน้าชื่อฟังก์ชัน หากต้องการเปลี่ยนแปลงโค้ดภายในฟังก์ชันแบบ weak type ทำได้โดยสร้างฟังก์ชันที่ใช้ชื่อเดียวกันในไฟล์อื่น เช่น main.c ดังรูปที่ 6.4 ซึ่งเป็นการ implement ฟังก์ชัน Callback ของ EXTI ในไฟล์ main.c โดยที่ไม่ได้แก้ไขเปลี่ยนแปลงฟังก์ชัน Callback ในไฟล์ stm32f4xx_hal_gpio.c ดังรูปที่ 6.3

```
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(GPIO_Pin);

    /* NOTE: This function Should not be modified, when the callback is needed,
       the HAL_GPIO_EXTI_Callback could be implemented in the user file
    */
}
```

รูปที่ 6.3 แสดงฟังก์ชัน Callback ที่ประกาศแบบ weak type ในไฟล์ stm32f4xx_hal_gpio.c

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == GPIO_PIN_0)
    {
        HAL_UART_Transmit(&huart1, (uint8_t *) "---", 3, 100);
        HAL_Delay(200);

        for(int i=0; i<20; i++)
        {
            HAL_UART_Transmit(&huart1, (uint8_t *) "B", 1, 100);
            HAL_Delay(200);
        }
    }
}
/* USER CODE END 4 */
```

รูปที่ 6.4 แสดงการ implement ฟังก์ชัน Callback ของ EXTI ในไฟล์ main.c

ฟังก์ชัน HAL_GPIO_EXTI_Callback () ในไฟล์ main.c

- เป็นการ implement ฟังก์ชัน Callback ของ EXTI ในไฟล์ main.c ฟังก์ชันนี้จะถูกเรียกโดยอัตโนมัติภายใน ISR ของ EXTI โดยรับพารามิเตอร์ 1 พารามิเตอร์ คือ หมายเลข EXTI ที่เกิดสัญญาณ interrupt
- ตรวจสอบว่าสัญญาณ interrupt ที่เกิดขึ้นนั้นมาจากการกดสวิตช์ B1 ที่เชื่อมต่อกับขา PA0 หรือไม่
- หากตรวจสอบพบว่าเป็น interrupt ที่มาจากขา PA0 รวมถึง GPIO ขา 0 จากพอร์ตอื่นๆ เช่น PB0 หรือ PC0 จะพิมพ์ข้อความ "---" ออกทาง UART1 แล้วต่อด้วยพิมพ์ตัวอักษร 'B' จำนวน 20 ตัวอักษร

```
HAL_UART_Transmit(&huart1, (uint8_t *) "---", 3, 100);
HAL_Delay(200);

for(int i=0; i<20; i++)
{
    HAL_UART_Transmit(&huart1, (uint8_t *) "B", 1, 100);
    HAL_Delay(200);
}
```

7. การทดลอง

1. ใช้โปรแกรม STM32CubeMX สร้างโปรเจกต์ขึ้นมา จากนั้นกำหนดขาต่างๆ ดังนี้

- สวิตช์ B1 ที่ขา PA0 ให้ทำหน้าที่ GPIO_EXTI0 ดังรูปที่ 2.1 ถึง รูปที่ 2.3
- LED ทั้งสองดวงที่ขา PG13 และ PG14 ให้ทำหน้าที่ GPIO_Output
- UART1 ที่ขา PA9 และ PA10 ให้ทำหน้าที่ UART

จากนั้นเขียน ISR เพื่อตอบสนองการกดสวิตช์ PA0 ดังรูปที่ 5.2 แล้ว implement ฟังก์ชัน Callback ของ EXTI ในไฟล์ main.c ดังรูปที่ 6.4

สำหรับฟังก์ชัน main() ให้เขียนโปรแกรมเพื่อส่งตัวอักษร Period ‘.’ ออกมาเรื่อยๆ ไม่สิ้นสุด โดยช่วงเวลาระหว่างตัวอักษร 400 ms

จากนั้นทดลองกดสวิตช์ B1 สังเกตแล้วบันทึกผลที่เกิดขึ้นในโปรแกรม Tera Term



2. ให้ต่อสวิตช์ภายนอกเข้ากับขา PC13 แล้วตั้งค่าให้สวิตช์ภายนอกนี้ตรวจจับสัญญาณขอบขาสูงหรือขอบขาขึ้นเพื่อสร้างสัญญาณ interrupt ขึ้น แล้วเขียนโปรแกรม ISR ของ PC13 เพื่อตอบสนองต่อสัญญาณ interrupt จากการกดสวิตช์ภายนอก โดยให้ Toggle LED LD2 (PG14) บนบอร์ด

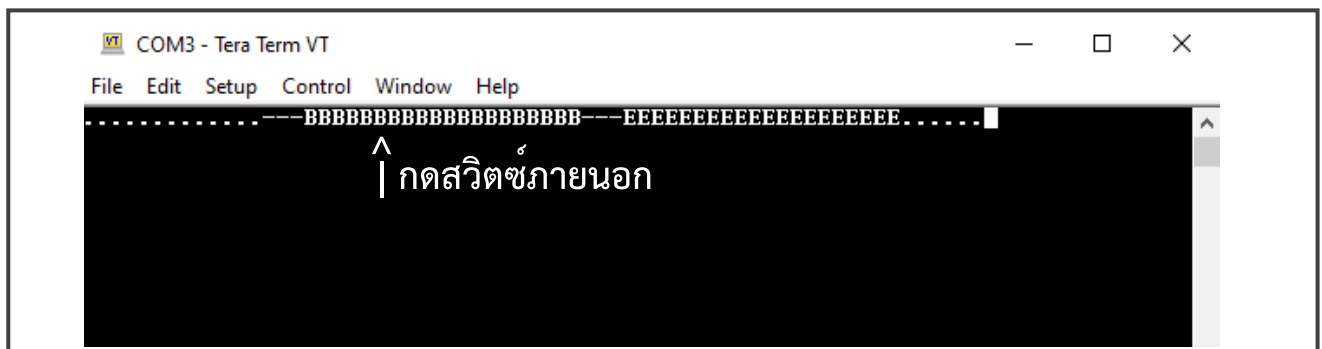
จากนั้น implement ฟังก์ชัน Callback ดังรูปที่ 6.4 เพิ่มเติม โดยให้ตรวจสอบว่าหากเป็น interrupt ที่เกิดจากขา GPIO_PIN_13 ให้ส่งข้อความ “---” ทางพอร์ต UART1 แล้วตามด้วยตัวอักษร ‘E’ จำนวน 20 ตัวอักษร

3. ทดสอบการทำงานของ Priority Interrupt โดยใช้ **NVIC_PriorityGroup_2** และตั้งค่า Preemption และ SubPriority ดังตารางที่ 7.1 สำหรับการทดลองนั้น ให้กดสวิตช์ B1 ก่อนแล้วจึงกดสวิตช์ภายนอกขณะที่กำลังพิมพ์ตัวอักษร ‘B’ อยู่ (ISR ของ EXTI0_IRQn ยังทำงานอยู่) แล้วให้ลองสลับลำดับการกดสวิตช์ สังเกตแล้วบันทึกผล

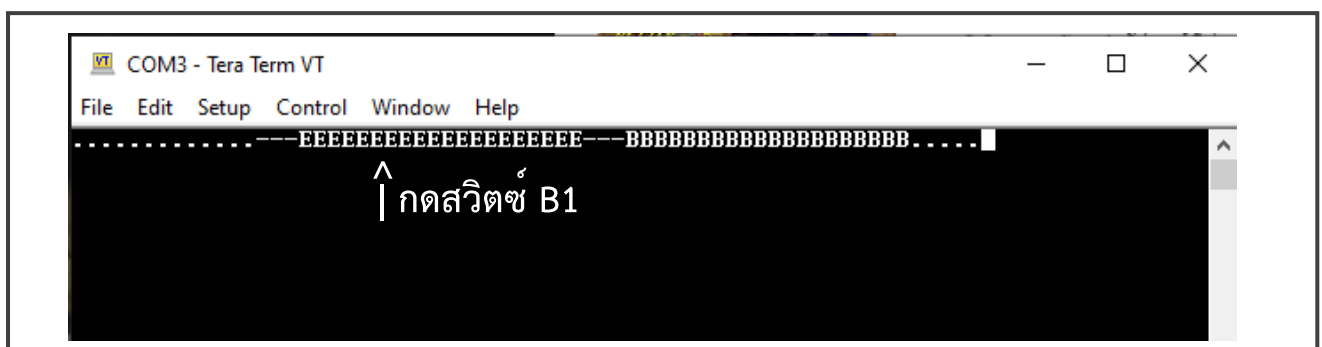
ตารางที่ 7.1 แสดงการตั้งค่า Interrupt Priority ให้กับสัญญาณ Interrupt


ข้อ	สัญญาณ interrupt	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority
3.1	สวิตช์ B1	2	2
	สวิตช์ภายนอก	2	0
3.2	สวิตช์ B1	3	1
	สวิตช์ภายนอก	2	3

ผลที่เกิดขึ้นในโปรแกรม Tera Term จากการทดลอง 3.1 (กดสวิตช์ B1 ก่อน) โดยให้ระบุช่วงเวลาการกดปุ่มโดยประมาณ



ผลที่เกิดขึ้นในโปรแกรม Tera Term จากการทดลอง 3.1 (กดสวิตช์ภายนอกก่อน) โดยให้ระบุช่วงเวลาการกดปุ่มโดยประมาณ






COM3 - Tera Term VT

File Edit Setup Control Window Help

.....---BBBBBB---EEEEEEEEEEEEEEEEEEEEBBBBBBBBBBBBBB.....

^
| กตสวิตซ์ภายนอก



COM3 - Tera Term VT

File Edit Setup Control Window Help

-----EEEEEEEEEEEEEEEEEEEE-----BBBBBBBBBBBBBBBBBBBB-----

-----EEEEEEEEEEEEEEEEEEEE-----BBBBBBBBBBBBBBBBBBBB-----

กดสวิตช์ B1

ใบตรวจการทดลองที่ 4

Microcontroller Application and Development 2564

วัน/เดือน/ปี 19/9/2564 กลุ่มที่ _____

1. รหัสนักศึกษา 62010694 ชื่อ-นามสกุล นายภากรณ์ ธนประชนนท์
2. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____
3. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 3.2 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. จากการทดลองข้อ 3 หากเปลี่ยนไปใช้ **NVIC_PriorityGroup_1** สัญญาณ interrupt จากสวิตช์ B1 หรือสวิตช์ภายนอกจะสามารถ interrupt ISR ของอีกฝ่ายที่กำลังทำงานอยู่ได้หรือไม่ ถ้าได้ให้ยกตัวอย่างประกอบ ถ้าไม่ได้ให้บอกสาเหตุ (สงวน Preemption 0 ให้กับ Timer เท่านั้น)

.....
ไม่ได้เพราะ NVIC_PriorityGroup_1 มี PreemptionPriority ได้แค่ 0 และ 1 หาก
.....
สงวน 0 ให้ Timer ไปแล้ว จะเหลือแค่ PreemptionPriority ค่า 1 ไว้ใช้ ซึ่งมีจำนวนไม่พอ
.....
กับงานของเราที่ต้องการ 2 Interrupt แต่หากใช้ค่าเดียวกันสองตัวก็จะได้ผลอีก
.....
เพราะสัญญาณ Interrupt ที่มีค่า PreemptionPriority เท่ากัน
.....
จะไม่สามารถแทรกการทำงานของ Interrupt อีกตัวได้
.....