<u>การทดลองที่ 4</u> 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 ที่เกิดก่อนหน้า

ARM ได้ออกแบบให้ Cortex M4 มีรีจีสเตอร์เพื่อใช้กำหนดระดับความสำคัญของสัญญาณ interrupt ขนาด 8 บิต ทั้งนี้ผู้ผลิตแต่ละรายสามารถกำหนดให้มีการใช้งานน้อยกว่า 8 บิตได้ เช่น ไอซี STM32F411 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

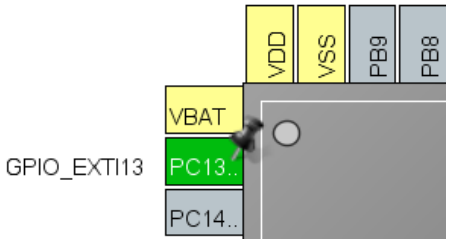
NVIC_PriorityGroup	PreemptionPriority		SubPriority	
	จำนวนบิต	ค่าเป็นไปได้	จำนวนบิต	ค่าเป็นไปได้
NVIC_PriorityGroup_0	0	0	4	0-15
NVIC_PriorityGroup_1	1	0-1	3	0-7
NVIC_PriorityGroup_2	2	0-3	2	0-3
NVIC_PriorityGroup_3	3	0-7	1	0-1
NVIC_PriorityGroup_4	4	0-15	0	0

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

หากเกิดสัญญาณ interrupt สองสัญญาณพร้อมกัน และทั้งสองสัญญาณนั้นมี `PreemptionPriority` เท่ากัน สัญญาณที่ถูกกำหนดให้มีค่า `SubPriority` ต่ำกว่าจะได้รับการตอบสนองก่อน

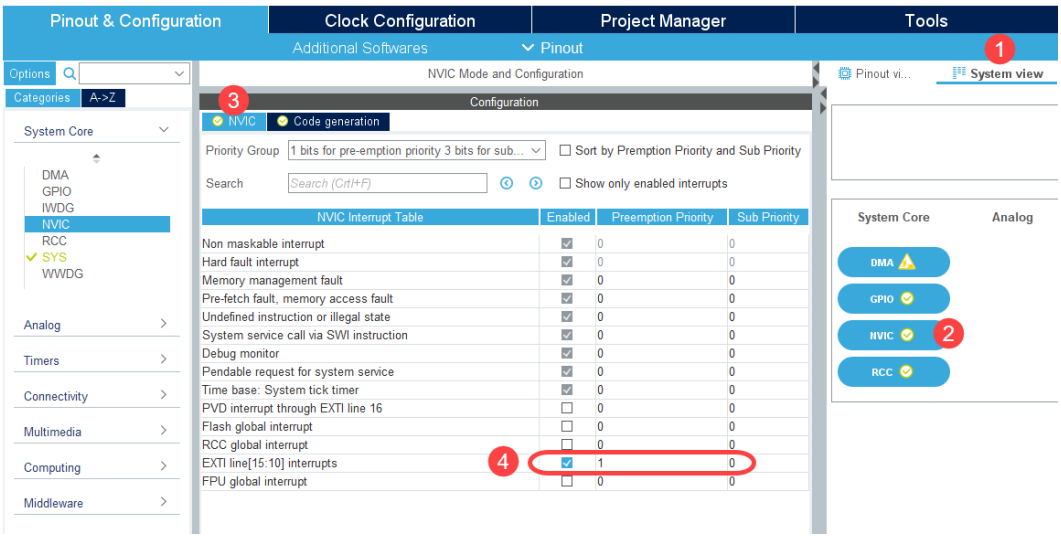
2. การตั้งค่าในโปรแกรม STM32CubeMX

การตั้งค่าสำหรับการทดลองครั้งนี้แบ่งออกเป็น 2 ส่วน ได้แก่ NVIC และ EXTI โดยเริ่มต้นที่แท็บ Pinout ในโปรแกรม STM32CubeMX กำหนดให้ขา PC13 ซึ่งเชื่อมต่อกับสวิตช์ B1 บนบอร์ด ทำหน้าที่เป็นตัวรับสัญญาณจากภายนอกหมายเลข 13 (`EXTI13`) ดังรูปที่ 2.1 จากนั้นตั้งค่า RCC และความถี่ของสัญญาณนาฬิกาตามการทดลองก่อนหน้า

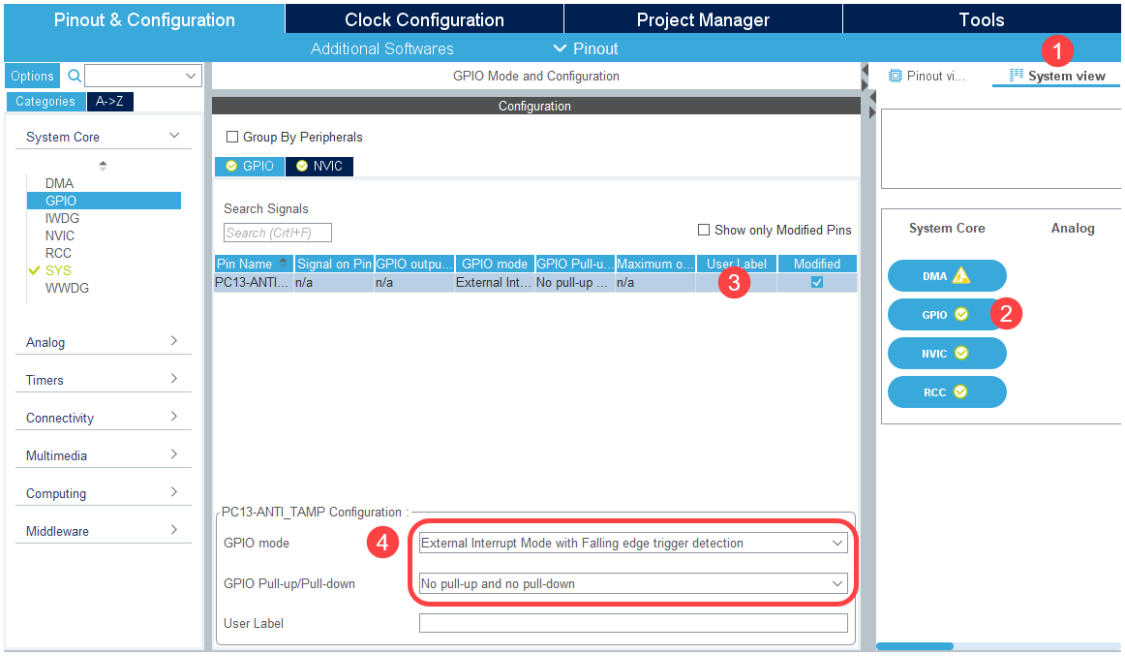


รูปที่ 2.1 แสดงการตั้งค่าให้ PC13 ทำหน้าที่ EXTI13

จากนั้นตั้งค่า NVIC กลุ่ม 1 คือมี `PreemptionPriority` 1 บิต และ `SubPriority` 3 บิต ดังรูปที่ 2.2 แล้วตั้งค่าให้ PC13 ทำหน้าที่ตรวจจับสัญญาณที่เข้ามาเพื่อสร้างสัญญาณ interrupt ไปยัง NVIC โดยกำหนดให้เป็นขาอินพุตแบบ floating และตรวจจับหากสัญญาณเปลี่ยนจากลอจิก 1 เป็นลอจิก 0 หรือตรวจจับขอบขาลง (Falling Edge) ของสัญญาณที่เข้ามายังขา PC13 ดังรูปที่ 2.3



รูปที่ 2.2 แสดงการตั้งค่า `NVIC_PriorityGroup_1`



รูปที่ 2.3 แสดงการตั้งค่า `PC13` ให้ทำหน้าที่ `EXTI13` โดยตรวจจับขอบขาลงของสัญญาณที่เข้ามา

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_GPIOC_CLK_ENABLE();

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}
```

รูปที่ 3.2 แสดงการตั้งค่าให้ PC13 ทำหน้าที่ EXTI13 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 เพื่อตั้งค่า GPIO บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับที่กำหนดไว้ในโปรแกรม
- เริ่มต้นด้วยการ Enable สัญญาณนาฬิกาให้ GPIOC (สำหรับสวิตช์ B1)

```
__GPIOC_CLK_ENABLE();
```

- กำหนดให้ PC13 ทำหน้าที่ EXTI13 โดยกำหนดให้ทำงานเป็นอินพุต floating และจะสร้างสัญญาณ interrupt ไปยัง NVIC เมื่อตรวจพบขอบขาลงของสัญญาณที่รับเข้ามา (มีการกดสวิตช์ B1)

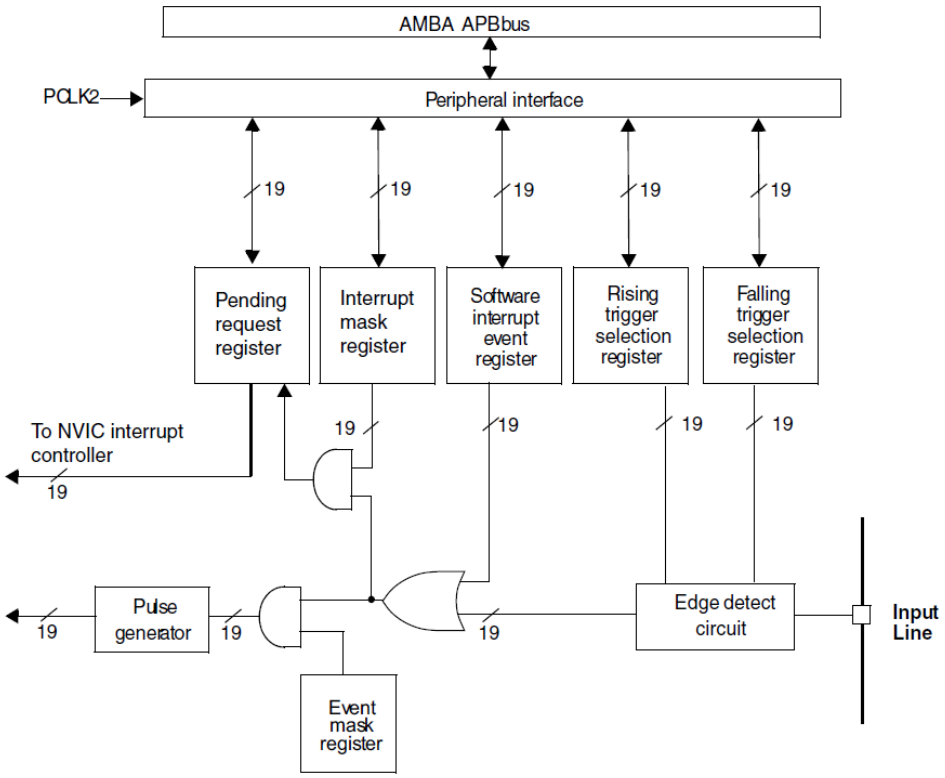
```
GPIO_InitStruct.Pin   = GPIO_PIN_13;
GPIO_InitStruct.Mode  = GPIO_MODE_IT_FALLING;
GPIO_InitStruct.Pull  = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

- กำหนดระดับความสำคัญให้กับ EXTI13 ซึ่งกำหนดให้มี PreemptionPriority = 1 และ SubPriority = 0 พร้อมสั่งให้เริ่มต้นการทำงาน

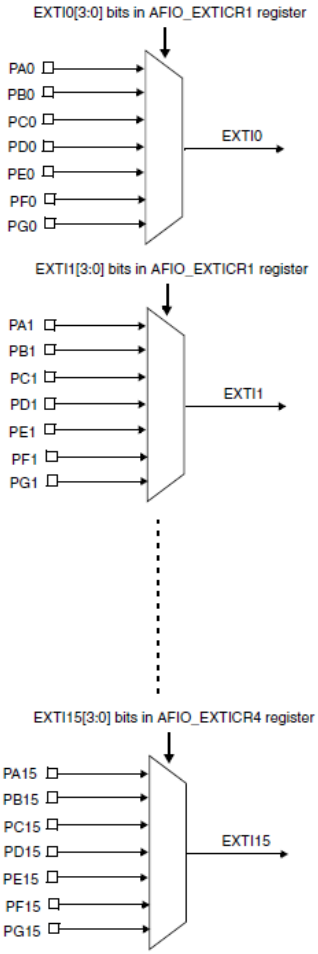
```
HAL_NVIC_SetPriority(EXTI15_10_IRQn, 1, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

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



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 (PC13) ถูกกดจะเกิดสัญญาณ interrupt จากโมดูล EXTI13 ไปยังหน่วยประมวลผล หน่วยประมวลผลจะหยุดการทำงานปัจจุบันลงแล้วไปทำงานที่ฟังก์ชัน `EXTI15_10_IRQHandler()` ซึ่งเป็น ISR ของ EXTI13 interrupt

สำหรับฟังก์ชัน `EXTI15_10_IRQHandler()` ในไฟล์ `stm32f4xx_it.c` เป็น ISR ที่ได้กำหนดไว้แล้วล่วงหน้าของสัญญาณ Interrrupt `EXTI15_10_IRQn` ซึ่งเชื่อมต่อกับ `EXTI13` โดยชื่อฟังก์ชันจะสัมพันธ์กับการประกาศ Vector Table ในไฟล์ `startup_stm32f411xe.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
DCD     WWDG_IRQHandler           ; Window Watchdog
DCD     PVD_IRQHandler            ; PVD through EXTI Line detect
DCD     TAMPER_IRQHandler         ; Tamper
DCD     RTC_IRQHandler            ; RTC
DCD     FLASH_IRQHandler          ; Flash
DCD     RCC_IRQHandler            ; RCC
DCD     EXTI0_IRQHandler          ; EXTI Line 0
DCD     EXTI1_IRQHandler          ; EXTI Line 1
DCD     EXTI2_IRQHandler          ; EXTI Line 2
DCD     EXTI3_IRQHandler          ; EXTI Line 3
DCD     EXTI4_IRQHandler          ; EXTI Line 4
```

รูปที่ 5.1 แสดงการกำหนด Vector Table

รูปที่ 5.2 แสดงตัวอย่าง ISR ของ `EXTI15_10_IRQn` ซึ่งรวม `EXTI13` อยู่ด้วย โดยจะทำงานเมื่อสวิตช์ B1 ถูกกด ซึ่งจะส่งตัวอักษร **'B'** จำนวน 20 ตัวอักษรออกมาทาง UART2 ส่วนฟังก์ชัน `HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13)` ที่ถูกเรียกใช้ในฟังก์ชันนี้เป็นการตรวจสอบและเคลียร์บิต Interrupt Pending เพื่อยกเลิกสัญญาณ Interrupt

```c
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */
  int i;

  for (i=0; i<20; i++)
  {
    HAL_UART_Transmit(&huart2, (uint8_t *) "B", 1, 10);
    HAL_Delay(200);
  }
  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

รูปที่ 5.2 แสดง Interrupt Service Routine ของ `EXTI15_10_IRQn`

6. การทดลอง

1. ใช้โปรแกรม STM32CubeMX สร้างโปรเจ็คขึ้นมา โดยเรียกใช้ PC13, UART2 และ RCC โดยกำหนดให้ PC13 ทำหน้าที่ `GPIO_EXTI13` ดังรูปที่ 2.1 ถึง รูปที่ 2.3 จากนั้นเขียน ISR เพื่อตอบสนองการกดสวิตช์ PC13 ดังรูปที่ 5.2 แล้วเขียนโปรแกรมเพื่อให้ฟังก์ชัน `main()` **ส่งตัวอักษร 'x' ออกมาเรื่อยๆ ไม่สิ้นสุด** โดยหน่วงเวลาระหว่างตัวอักษร 300 ms

ในฟังก์ชัน `EXTI15_10_IRQHandler()` มีการเรียกใช้ตัวแปร `huart2` เพื่อส่งข้อมูลตัวอักษรทาง UART2 แต่เนื่องจากตัวแปรดังกล่าวได้ประกาศใช้และเริ่มต้นค่าในไฟล์ `main.c` ทำให้คอมไพเลอร์แจ้งข้อความผิดพลาด แก้ปัญหาดังกล่าว**โดยการประกาศตัวแปร `huart2` ซ้ำ**ในไฟล์ `stm32f4xx_it.c` พร้อมใช้คีย์เวิร์ด `extern` นำหน้า ดังรูปที่ 6.1

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/

/* USER CODE BEGIN EV */
extern UART_HandleTypeDef huart2;
/* USER CODE END EV */
```

รูปที่ 6.1 แสดงการเรียกใช้ตัวแปรที่ประกาศจากไฟล์อื่น

จากนั้นทดลองกดสวิตช์ B1 สังเกตแล้วบันทึกผลที่เกิดขึ้น

.....

.....

.....

.....

.....

.....

.....

.....

2. ให้ต่อสวิตช์ภายนอก**แบบ Pull up** เข้ากับขา PB2 แล้วตั้งค่าให้สวิตช์ภายนอกนี้ตรวจจับสัญญาณขอบขาลงเพื่อสร้างสัญญาณ interrupt ขึ้น แล้วเขียนโปรแกรม ISR ของ PB2 เพื่อตอบสนองต่อสัญญาณ interrupt จากการกดสวิตช์ภายนอก โดยให้ Toggle LED LD2 บนบอร์ด แล้วส่งตัวอักษร **'E'** ทางพอร์ต UART2 จำนวน 20 ตัวอักษร

3. ทดสอบการทำงานของ Priority Interrupt โดยใช้ **`NVIC_PriorityGroup_2`** และตั้งค่า `Preemption` และ `SubPriority` ดังตารางที่ 6.1 สำหรับการทดลองนั้นให้กดสวิตช์ B1 แล้วจึงกดสวิตช์ภายนอกขณะที่กำลังพิมพ์ตัวอักษร 'B' อยู่ (ISR ของ `EXTI15_10_IRQn` ยังทำงานอยู่) แล้วให้ลองสลับลำดับการกดสวิตช์ สังเกตแล้วบันทึกผล

ตารางที่ 6.1 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

ผลการทดลอง 3.1

ผลการทดลอง 3.2

ใบตรวจการทดลองที่ 4

KU CSC Embedded System

วัน/เดือน/ปี _____ **กลุ่มที่** _____

1. รหัสนิสิต _____ ชื่อ-นามสกุล _____

2. รหัสนิสิต _____ ชื่อ-นามสกุล _____

3. รหัสนิสิต _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 3 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. จากการทดลอง 3 หากเปลี่ยนไปใช้ **`NVIC_PriorityGroup_0`** สัญญาณ interrupt จากสวิตช์ B1 จะสามารถ interrupt ISR ของสวิตช์ภายนอกที่กำลังทำงานอยู่ได้หรือไม่ ถ้าได้ให้ยกตัวอย่างประกอบ ถ้าไม่ได้ให้บอกสาเหตุ

.....

.....

.....

.....

.....

.....

.....