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 ที่เกิดก่อนหน้า

ARM ได้ออกแบบให้ Cortex M3 มีรีจีสเตอร์เพื่อใช้กำหนดระดับความสำคัญของสัญญาณ interrupt ขนาด 8 บิต ทั้งนี้ผู้ผลิตแต่ละรายสามารถกำหนดให้มีการใช้งานน้อยกว่า 8 บิตได้ เช่น ไอซี STM32F1XX 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

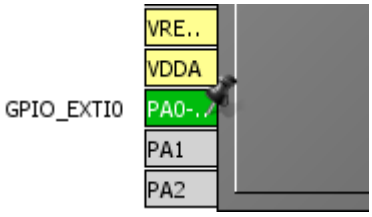
NVIC_PriorityGroup	PreemptionPriority		SubPriority	
	จำนวนบิต	ค่าเป็นไปได้	จำนวนบิต	ค่าเป็นไปได้
NVIC_PriorityGroup_0	0	0	4	0-15
NVIC_PriorityGroup_1	1	0-1	3	0-7
NVIC_PriorityGroup_2	2	0-3	2	0-3
NVIC_PriorityGroup_3	3	0-7	1	0-1
NVIC_PriorityGroup_4	4	0-15	0	0

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

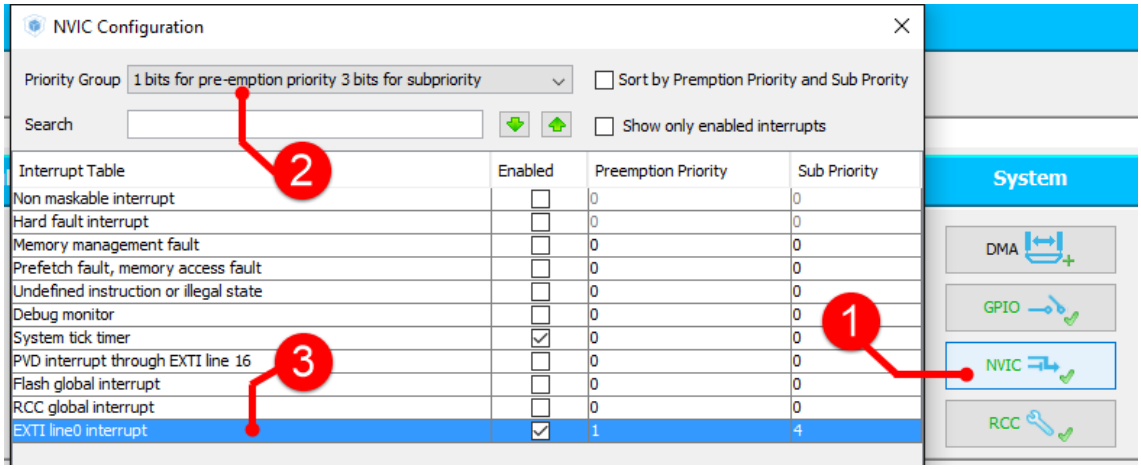
2. การตั้งค่าในโปรแกรม STM32CubeMX

การตั้งค่าสำหรับการทดลองครั้งนี้แบ่งออกเป็น 2 ส่วน ได้แก่ NVIC และ EXTI โดยเริ่มต้นที่แท็บ Pinout ในโปรแกรม STM32CubeMX กำหนดให้ขา PA0 ทำหน้าที่เป็นตัวรับสัญญาณจากภายนอกหมายเลข 0 (EXTI0) ดังรูปที่ 2.1 จากนั้นตั้งค่า RCC และความถี่ของสัญญาณนาฬิกาตามการทดลองก่อนหน้า

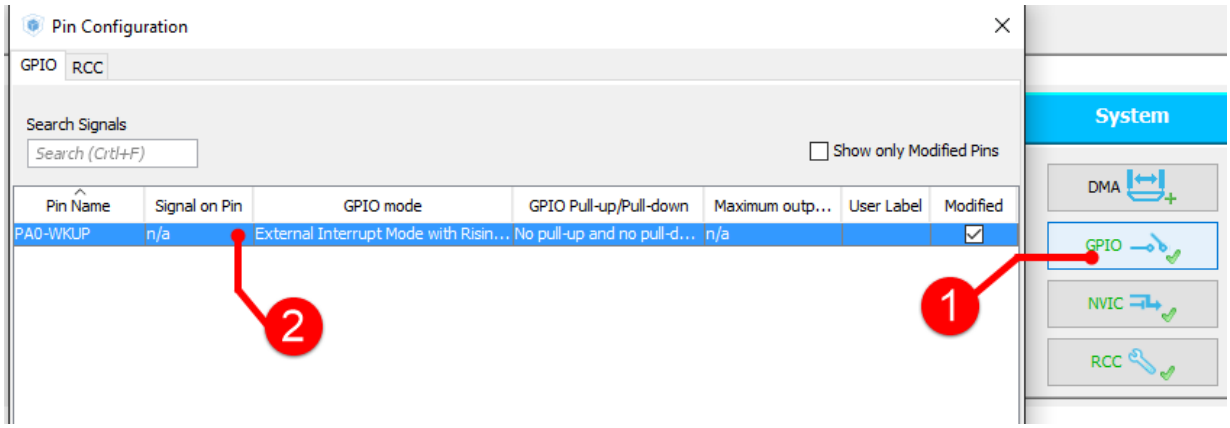


รูปที่ 2.1 แสดงการตั้งค่าให้ PA0 ทำหน้าที่ EXTI0

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 หรือตรวจจับขอบขาขึ้นของสัญญาณที่เข้ามายังขา PA0 ดังรูปที่ 2.3



รูปที่ 2.2 แสดงการตั้งค่า `NVIC_PriorityGroup_1`



รูปที่ 2.3 แสดงการตั้งค่า `PA0` ให้ทำหน้าที่ `EXTI0` โดยตรวจจับขอบขาขึ้นของสัญญาณที่เข้ามา

3. อธิบายการทำงานของ NVIC

การตั้งค่า NVIC เพื่อควบคุมสัญญาณ interrupt ของโค้ดที่สร้างจากโปรแกรม STM32CubeMX จะอยู่ในฟังก์ชัน `HAL_Init()` ในไฟล์ `stm32f1xx_hal.c` ดังรูปที่ 3.1

```
  __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
#endif
#endif /* PREFETCH_ENABLE */

  /* Set Interrupt Group Priority */
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* Use systick as time base source and configure 1ms tic
  HAL_InitTick(TICK_INT_PRIORITY);
```

รูปที่ 3.1 แสดงการตั้งค่า Group Priority ในฟังก์ชัน HAL_Init() ในไฟล์ stm32f1xx_hal.c

```
void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* 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, 4);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

}
```

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

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

ฟังก์ชัน MX_GPIO_init ( )

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่า GPIO บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับที่กำหนดไว้ในโปรแกรม
- เริ่มต้นด้วยการ Enable สัญญาณนาฬิกาให้ GPIOA (สำหรับสวิตช์ Wakeup)

  __HAL_RCC_GPIOA_CLK_ENABLE();
- กำหนดให้ PA0 ทำหน้าที่ EXTI0 โดยกำหนดให้ทำงานเป็นอินพุต floating และจะสร้างสัญญาณ interrupt ไปยัง NVIC เมื่อตรวจพบขอบขาขึ้นของสัญญาณที่รับเข้ามา (มีการกดสวิตช์ wakeup)

  GPIO_InitStruct.Pin = GPIO_PIN_0;

  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;

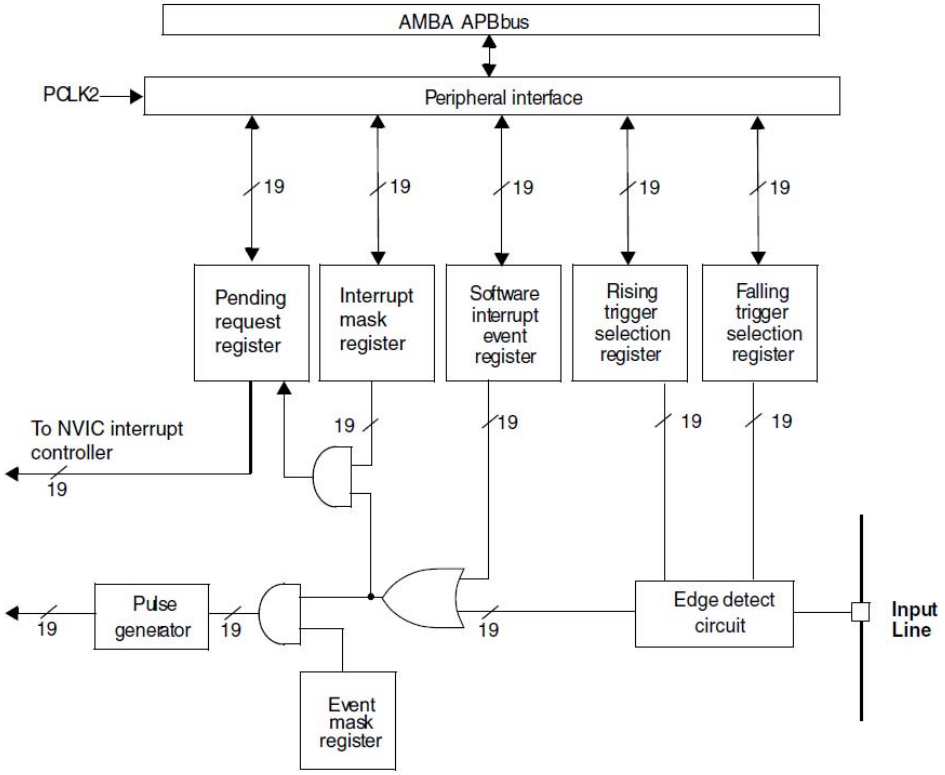
  GPIO_InitStruct.Pull = GPIO_NOPULL;

  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
- กำหนดระดับความสำคัญให้กับ EXTI0 ซึ่งกำหนดให้มี `PreemptionPriority = 1` และ `SubPriority = 4` พร้อมสั่งให้เริ่มต้นการทำงาน

```
HAL_NVIC_SetPriority(EXTI0_IRQn, 1, 4);
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 5.1 ซึ่งขณะใดขณะหนึ่งจะมีเพียงขา GPIO เพียงขาเดียวเท่านั้นที่ทำหน้าที่รับสัญญาณอินพุตแล้วส่งต่อไปยังโมดูล EXTI แต่ละหมายเลข

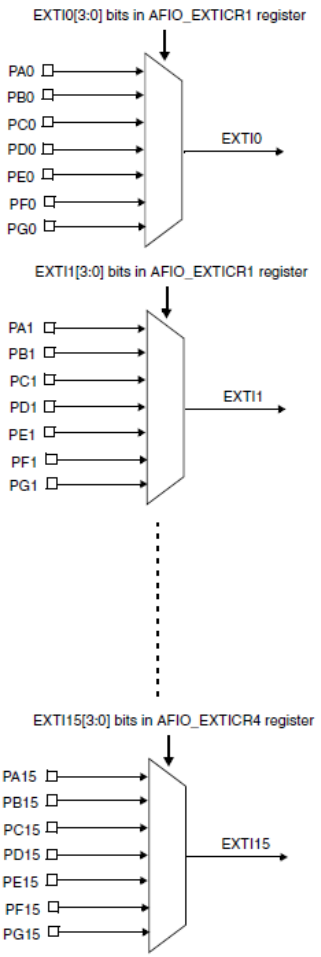


รูปที่ 4.1 แสดงโครงสร้างของโมดูล External Interrupt

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 wakeup (PA0) ถูกกดจะเกิดสัญญาณ interrupt จากโมดูล EXTI0 ไปยังหน่วยประมวลผล หน่วยประมวลผลจะหยุดการทำงานปัจจุบันลงแล้วไปทำงานที่ฟังก์ชัน `EXTI0_IRQHandler()` ซึ่งเป็น ISR ของ EXTI0 interrupt



รูปที่ 5.1 แสดงการเชื่อมต่อ GPIO ไปยังโมดูล EXTI

สำหรับฟังก์ชัน `EXTI0_IRQHandler()` ในไฟล์ `stm32f1xx_it.c` เป็น ISR ของโมดูล EXTI0 ที่ได้กำหนดไว้แล้ว โดยชื่อฟังก์ชันจะสัมพันธ์กับการประกาศ Vector Table ในไฟล์ startup_stm32f107xc.s ด้วยภาษา Assembly ดังรูปที่ 5.2 สำหรับสัญญาณ EXTI หมายเลขอื่นๆ ก็จะมีฟังก์ชัน ISR ดังตารางที่ 5.1

ตารางที่ 5.1 แสดงฟังก์ชัน ISR ของ EXTI แต่ละหมายเลข

หมายเลข EXTI	ชื่อฟังก์ชัน ISR	หมายเหตุ
EXTI0	EXTI0_IRQHandler	-
EXTI1	EXTI1_IRQHandler	-
EXTI2	EXTI2_IRQHandler	-
EXTI3	EXTI3_IRQHandler	-
EXTI4	EXTI4_IRQHandler	-
EXTI5 – EXTI9	EXTI9_5_IRQHandler	EXTI5 ถึง EXTI9 ใช้ ISR ร่วมกัน
EXTI10 – EXTI15	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 5.2 แสดงการกำหนด Vector Table

รูปที่ 5.3 แสดงตัวอย่าง ISR ของ EXTI0 โดยจะทำงานเมื่อสวิตช์ wakeup ถูกกด ซึ่งจะส่งตัวอักษร **'W'** จำนวน 20 ตัวอักษรออกมาทาง UART2 ส่วนฟังก์ชัน `HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0)` ที่ถูกเรียกใช้ในฟังก์ชันนี้เป็นการตรวจสอบและเคลียร์บิต Interrupt Pending เพื่อยกเลิกสัญญาณ Interrupt

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
  int i;
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */
  for (i=0; i<20; i++)
  {
    HAL_UART_Transmit(&huart2, (uint8_t *) "W", 1, 100);
    HAL_Delay(300);
  }
  /* USER CODE END EXTI0_IRQn 1 */
}
```

รูปที่ 5.3 แสดง Interrupt Service Routine ของ EXTI0

6. การทดลอง

1. ใช้โปรแกรม STM32CubeMX สร้างโปรเจ็คขึ้นมา โดยเรียกใช้ PA0, UART2 และ RCC โดยกำหนดให้ PA0 ทำหน้าที่ GPIO_EXTI0 ดังรูปที่ 2.1 ถึง รูปที่ 2.3 จากนั้นเขียน ISR เพื่อตอบสนองการกดสวิตช์ PA0 ดังรูปที่ 5.3 ให้เขียนโปรแกรมเพื่อให้ฟังก์ชัน `main()` ส่งตัวอักษร **'x'** ออกมาเรื่อยๆ ไม่สิ้นสุด โดยหน่วงเวลาระหว่างตัวอักษร 300 ms

ในฟังก์ชัน `EXTI0_IRQHandler()`มีการเรียกใช้ตัวแปร `huart2` เพื่อส่งข้อมูลตัวอักษรทาง `UART2` แต่เนื่องจากตัวแปรดังกล่าวได้ประกาศใช้และเริ่มต้นค่าในไฟล์ `main.c` ทำให้คอมไพเลอร์แจ้งข้อความผิดพลาด แก้ปัญหาดังกล่าวโดยการประกาศตัวแปร `huart2` ซ้ำในไฟล์ `stm32f1xx_it.c` พร้อมใช้คีย์เวิร์ด extern นำหน้า ดังนี้

- extern UART_HandleTypeDef huart2;

จากนั้นทดลองกดสวิตช์ wakeup สังเกตผลแล้วบันทึกผลการทดลอง

---

2. ให้เขียนโปรแกรมตรวจจับสัญญาณขอบขาลงของสวิตช์ Tamper เพื่อสร้างสัญญาณ interrupt ขึ้น แล้วเขียนโปรแกรม ISR ตอบสนองต่อสัญญาณ interrupt นั้น โดยให้ Toggle LED0 แล้วส่งตัวอักษร **'T'** ทางพอร์ต UART2 จำนวน 20 ตัวอักษร

3. ทดสอบการทำงานของ Priority Interrupt โดยใช้ `NVIC_PriorityGroup_1` และตั้งค่า `Preemption` และ `SubPriority` ดังตารางที่ 6.1 สำหรับการทดลองให้กดสวิตช์ wakeup แล้วจึงกดสวิตช์ tamper ขณะที่ ISR ของ EXTI0 ยังทำงานอยู่ และทำกลับกัน สังเกตแล้วบันทึกผล

ตารางที่ 6.1 แสดงการตั้งค่า Interrupt Priority ให้กับสัญญาณ Interrupt

ข้อ	สัญญาณ interrupt	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority
3.1	สวิตช์ Wakeup	1	4
	สวิตช์ Temper	1	0
3.2	สวิตช์ Wakeup	1	1
	สวิตช์ Temper	0	3

ผลการทดลอง 3.1

ผลการทดลอง 3.2

7. การทดลองพิเศษ (เลือก 1 ข้อ)

1. จงสร้างโปรแกรมเพื่อให้ไฟ LED ติดตามรูปแบบที่กำหนดดังตารางที่ 7.1

ตารางที่ 7.1 แสดงรูปแบบ LED

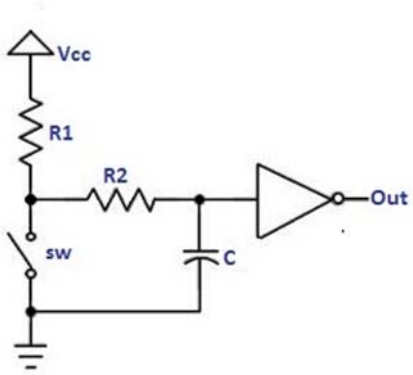
State \ LED	7	6	5	4	3	1	0
0	■	■	■	■	■	■	■
1	■	■	■	■	■	■	
2	■	■	■	■	■		
3	■	■	■	■			■
4	■	■	■			■	■
5	■	■			■	■	■
6	■			■	■	■	■
7			■	■	■	■	■
8		■	■	■	■	■	■
0	■	■	■	■	■	■	■

■ LED ติด

□ LED ดับ

โดยจะมีการเปลี่ยน state ทุกๆ 1 วินาที และวนจาก state 0 จนถึง state 8 แล้วจึงกลับไปเริ่มที่ state 0 ใหม่ โดยให้สร้างสัญญาณ interrupt โดยต่อกับ Debounced Switch ภายนอก 2 สวิตช์ ดังรูปที่ 7.1 โดยให้ TA เป็นผู้กำหนดขา EXTI ที่จะใช้ทำ interrupt และ Priority ในตารางที่ 7.2 โดยมีรายละเอียดการทำงานดังนี้

- เมื่อกดสวิตช์แรกให้เปลี่ยนรูปแบบไฟ LED โดยเริ่มต้นที่ LED ทุกดวงดับหมด จากนั้นให้ LED 0 ติด 1 วินาที ดับ 1 วินาที แล้วให้ LED 1 ติดแล้วจึงสลับไปเรื่อยๆ จนถึง LED 7 โดยเมื่อ LED 7 ดับแล้วให้จบการทำงานของ ISR
- เมื่อกดสวิตช์ที่สองให้ไฟ LED ทุกดวงดับแล้วติดพร้อมกันจำนวน 2 ครั้ง โดยจะต้องมีการ**ตอบสนองต่อสัญญาณ interrupt ทันที** แล้วจึงจบการทำงานของ ISR



- V_{CC} ให้ใช้ขา +3V3 จากบอร์ด
- R_1 10 kΩ
- R_2 1 kΩ
- C 10 μF
- ไม่ต้องต่อ Inverter เนื่องจากใช้ Schmidt Trigger ภายในวงจร GPIO แทน

รูปที่ 7.1 การต่อวงจรเพื่อ debounce switch

ตารางที่ 7.2 แสดงการตั้งค่าสำหรับข้อพิเศษ

สวิตช์	EXTI	Priority Group	PreemptionPriority	SubPriority
1				
2				

2. ให้เปลี่ยนรูปแบบการรับส่งข้อมูลการทดลองข้อ 2 ในการทดลองครั้งที่ 3 เรื่อง UART จาก Polling มาเป็น Interrupt แทน โดยตั้งค่าตามความเหมาะสม

ใบตรวจการทดลองที่ 4

วัน/เดือน/ปี _____ ☐ **Sec 1** ☐ **Sec 2 กลุ่มที่** _____

1. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

2. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 1 ผู้ตรวจ _____ วันที่ตรวจ _____ ☐ W ☐ W+1 ☐ W+2

การทดลองข้อ 3 ผู้ตรวจ _____ วันที่ตรวจ _____ ☐ W ☐ W+1 ☐ W+2

การทดลองพิเศษ ผู้ตรวจ_____ วันที่ตรวจ _____ ☐ W ☐ W+1 ☐ W+2

คำถามท้ายการทดลอง

1. หากใช้ NVIC_PriorityGroup_0 สัญญาณ interrupt จากสวิตช์ wakeup จะสามารถ interrupt ISR ของสวิตช์ Tamper ที่กำลังทำงานอยู่ได้หรือไม่ ถ้าได้ให้ยกตัวอย่างประกอบ ถ้าไม่ได้ให้บอกสาเหตุ
