

REPORT



7 주차 결과 보고서

수강과목: 임베디드 시스템 설계 및 실험

담당교수: 백윤주 교수님

학과: 전기컴퓨터공학부 정보컴퓨터공학전공

조원: 201724651 김장환 201724648 김경호

201624481 박윤형 201524588 조창흠

제출일자: 2021.10.28

1. 실험 목적

1. Interrupt 원리의 이해
2. Interrupt 방식을 활용한 GPIO 제어 및 UART 통신
3. 라이브러리 함수 사용법 숙지

2. 배경지식

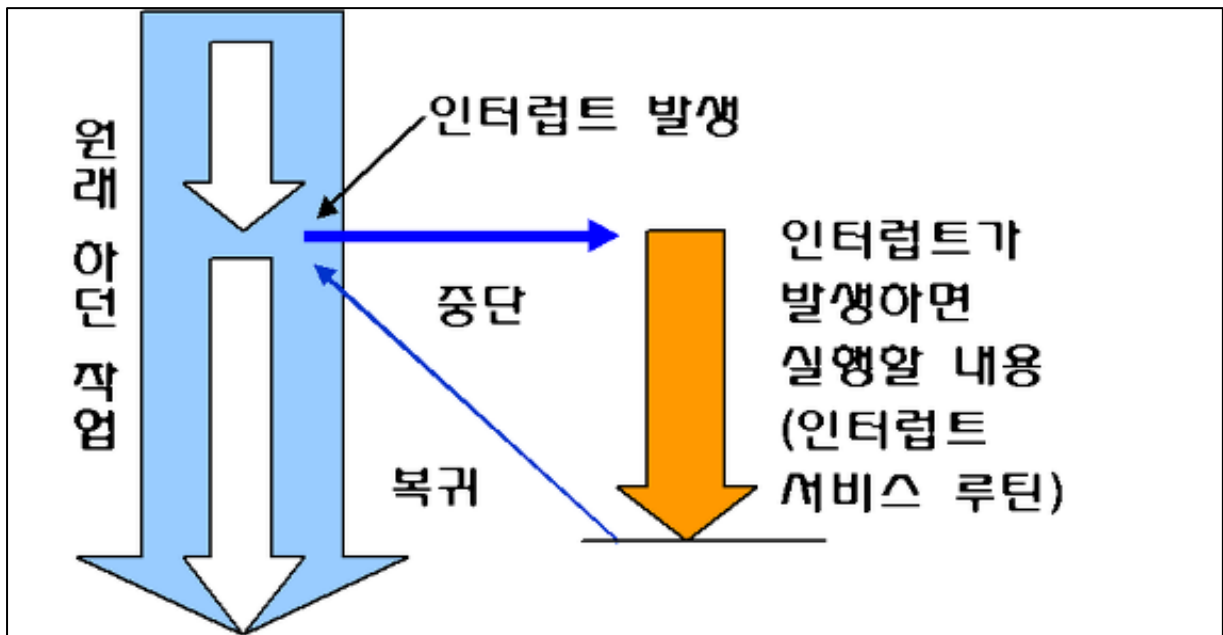
1. POLLING 과 INTERRUPT

< Polling >

- CPU 가 이벤트가 발생할 때 까지 모든 연산에서 감시하는 방식

< Interrupt >

- 이벤트 발생시 CPU 가 작업을 멈추고 해당 이벤트를 위한 인터럽트 서비스 루틴을 수행하고 중단했던 이전 작업으로 복귀하는 방식



2. Interrupt 종류

< Hardware Interrupt >

- 비동기식 이벤트 처리 수행
- 높은 우선 순위를 가짐
- ex) 스위치, 디스크 읽기 끝남, 키보드 입력 등

< Software Interrupt >

- 동기식 이벤트 처리 수행
- 낮은 우선 순위를 가짐

< EXTI(External Interrupt) >

- 외부에서 신호가 입력되면 Device 에 Event 혹은 Interrupt 발생시킴
- 입력되는 신호의 종류는 Rising-Edge, Falling-Edge, Rising & Falling Edge
- 각 Port 의 n 번 핀의 EXTI_n 에 연결
- Event 모드와 Interrupt 모드 중 선택
 - > Interrupt 모드는 Interrupt 가 발생하여 해당 핸들러가 동작하고 edge detector line 으로 구성되어 각 line 의 설정에 따라 rising/falling trigger 를 감지
- EXTI line 으로 입력 받은 신호와 레지스터 설정들을 비교하고 NVIC 컨트롤러로 전송

< NVIC(Nested Vectored Interrupt Controller)

- 인터럽트 처리 중 또 다른 인터럽트 발생시 우선순위를 사용
- 값이 작을 수록 우선순위가 높음

3. 실험 내용

- 라이브러리 구조체 및 함수 사용

직접 주소로 접근	<pre>(*volatile unsigned int *) 0x40021018) &= ~0x20; (*volatile unsigned int *) 0x40021018) = 0x20; (*volatile unsigned int *) 0x40011000) &= ~0x00000F00; (*volatile unsigned int *) 0x40011000) = 0x00000400;</pre>
정의된 주소 값 사용	<pre>RCC->APB2ENR &= ~(RCC_APB2ENR); RCC->APB2ENR = RCC_APB2ENR_IOPDEN; GPIO->CRL &= ~(GPIO_CRL_CNF2 GPIO_CRL_MODE2); GPIO->CRL = GPIO_CRL_MODE2_0;</pre>
구조체 및 함수 사용	<pre>GPIO_InitTypeDef GPIO_InitStructure; RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIO, ENABLE); GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2; GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; GPIO_Init(GPIO, &GPIO_InitStructure);</pre>

직접 주소로 접근	<pre>(*(volatile unsigned int *) 0x40011410) = 0x04; *(volatile unsigned int *) 0x40011414) = 0x04;</pre>
정의된 주소 값 사용	<pre>GPIOD->BSRR = GPIO_BSRR_BS2; GPIOD->BRR = GPIO_BRR_BR2;</pre>
구조체 및 함수 사용	<pre>GPIO_SetBits(GPIOD, GPIO_Pin_2); GPIO_ResetBits(GPIOD, GPIO_Pin_2);</pre>

- 직접 주소로 접근하거나 정의된 주소 값을 사용하는 대신 라이브러리 파일에 정의된 함수 및 구조체를 이용할 수 있다.

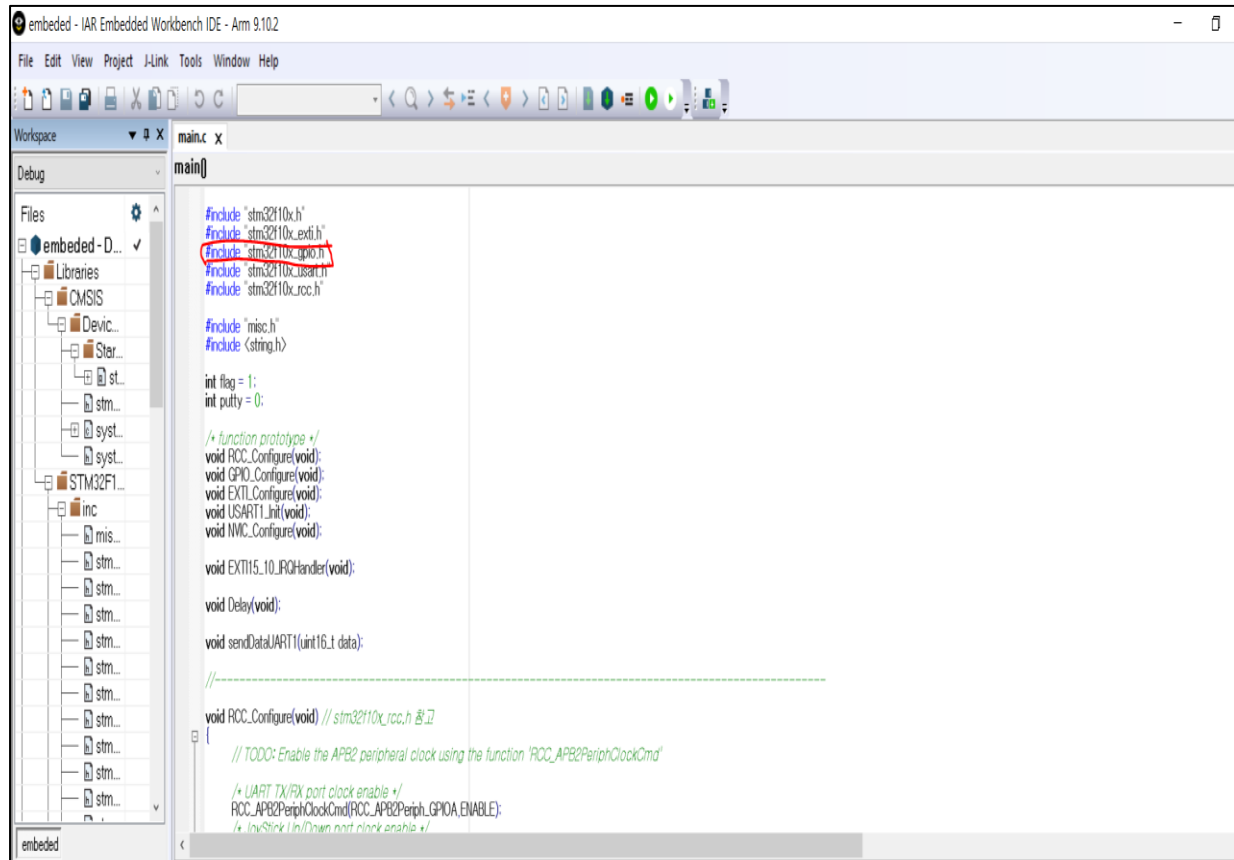
예시: GPIO_SetBits(GPIOD, GPIO_Pin_2);

Libraries\STM32F10x_StdPeriph_Driver\v3.5\inc\stm32f10x_gpio.h

```

stm32f10x_gpio.h
343  */
344
345  /** @defgroup GPIO_Exported_Functions
346   * @{
347   */
348
349  void GPIO_DeInit(GPIO_TypeDef* GPIOx);
350  void GPIO_AFIODeInit(void);
351  void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
352  void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct);
353  uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
354  uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
355  uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
356  uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
357  void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
358  void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
359  void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
360  void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
361  void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
362  void GPIO_EventOutputConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
363  void GPIO_EventOutputCmd(FunctionalState NewState);
364  void GPIO_PinRemapConfig(uint32_t GPIO_Remap, FunctionalState NewState);
365  void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
366  void GPIO_ETH_MediaInterfaceConfig(uint32_t GPIO_ETH_MediaInterface);
367
368  #ifdef __cplusplus
369  }
370  #endif
371
372  #endif /* __STM32F10x_GPIO_H */

```



4. 실험 과정

1. 각각의 레지스터에 RCC 를 이용해 Clock 을 인가해준다.
2. 각각의 포트에 맞는 모드를 configuration 해준다.
3. 인터럽트를 사용하기 위해 EXTI configuration 을 해준다.
4. USART 통신을 위해 USART 설정을 해주고
5. 마지막으로 NVIC 를 설정해주고 각 인터럽트에 맞는 Handler 함수를 만든다.

```
void RCC_Configure(void) // stm32f10x_rcc.h 참고
{
    // TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'

    /* UART TX/RX port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    /* JoyStick Up/Down port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    /* LED port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    /* USART1 clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    /* Alternate Function IO clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

- 각각의 레지스터에 clock을 인가해주는 코드이다. RCC_APB2PeriphClockCmd 함수를 이용해 사용할 레지스터에 clock을 인가해줬다.

```

void GPIO_Configure(void) // stm32f10x_gpio.h 참고
{
    GPIO_InitTypeDef GPIO_InitStructure;

    // TODO: Initialize the GPIO pins using the structure 'GPIO_InitTypeDef' and the function 'GPIO_Init'

    /* JoyStick up, down pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    /* button pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    /* LED pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    /* UART pin setting */
    //TX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    //RX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

```

- 각각의 GPIO register의 모드를 설정해주는 코드이다. 이번에 사용할 GPIO register의 구조체 변수를 사용하고 각각의 레지스터에 알맞는 모드와 핀 번호를 설정해주었다.
- 우리가 이번에 사용하는 LED, Joystick, S1 버튼, USART의 포트는 각각 D, C, A이기 때문에 세 포트에 대해서 구조체 변수를 선언하고, 구조체 내용을 알맞게 설정해주었다.

```

void EXTI_Configure(void) // stm32f10x_gpio.h 참고
{
    EXTI_InitTypeDef EXTI_InitStructure;

    // TODO: Select the GPIO pin (Joystick, button) used as EXTI Line using function 'GPIO_EXTILineConfig'
    // TODO: Initialize the EXTI using the structure 'EXTI_InitTypeDef' and the function 'EXTI_Init'

    /* Joystick Down */
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource2);
    EXTI_InitStructure.EXTI_Line = EXTI_Line2;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

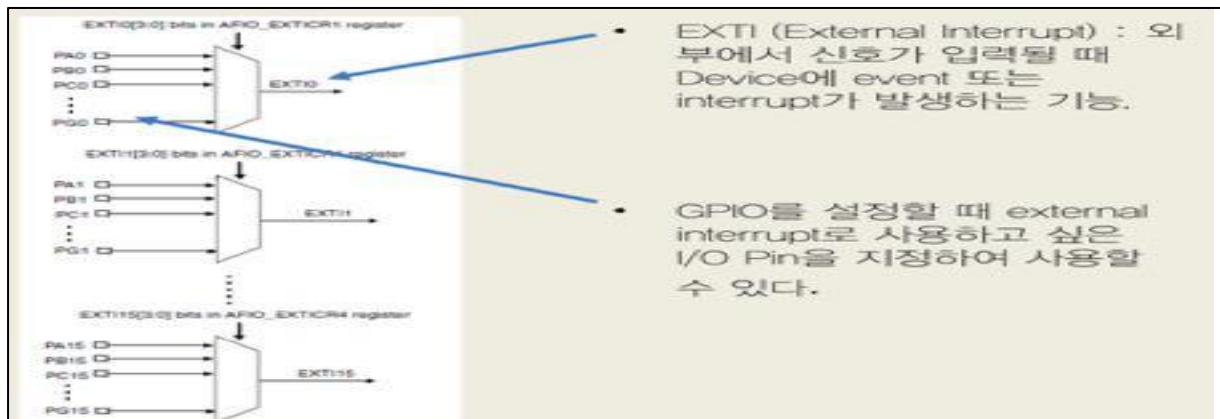
    /* Joystick Up */
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource5);
    EXTI_InitStructure.EXTI_Line = EXTI_Line5;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Button */
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOD, GPIO_PinSource11);
    EXTI_InitStructure.EXTI_Line = EXTI_Line11;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    // NOTE: do not select the UART GPIO pin used as EXTI Line here

```

- EXTI를 사용하기 위해 EXTI관련 설정을 해준다.



- 위의 그림과 같이 외부 신호가 입력되는 포트 번호에 해당하는 라인으로 EXTI를 설정하면 되고, 모든 GPIO 핀들은 EXTI line을 통해 연결한다. 같은 번호의 핀들은 같은 라인을 공유하고 이 라인을 통해 신호를 전달한다.


```

void USART1_Init(void) // stm32f10x_usart.h 참고
{
    USART_InitTypeDef USART1_InitStructure;

    // Enable the USART1 peripheral
    USART_Cmd(USART1, ENABLE);

    // TODO: Initialize the USART using the structure 'USART_InitTypeDef' and the funct

    /*
    BaudRate: 9600
    WordLength: 8bits
    Parity: None
    StopBits: 1bit
    Hardware Flow Control: None
    */
    USART1_InitStructure.USART_BaudRate = 9600;
    USART1_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART1_InitStructure.USART_Parity = USART_Parity_No;
    USART1_InitStructure.USART_StopBits = USART_StopBits_1;
    USART1_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

    USART_Init(USART1, &USART1_InitStructure);
    // TODO: Enable the USART1 RX interrupts using the function 'USART_ITConfig' and th
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
}

```

- USART관련 설정을 해주는 부분이다. Baudrate를 설정해주고, 워드의 길이, stop bit 등을 설정해주는 코드인데, 우리가 사용하는 워드의 길이는 8bit, stop bit는 1bit가 된다. 다음으로 clock을 인가해주고, 시리얼을 통해 입력을 받는 인터럽트가 발생하기 때문에 RX의 IT를 ENABLE해준다.

```

void NVIC_Configure(void) { // misc.h 참고

    NVIC_InitTypeDef NVIC_InitStructure;

    // TODO: fill the arg you want
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    // TODO: Initialize the NVIC using the structure 'NVIC_Init'

    // Joystick Down
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Joystick Up
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // User S1 Button
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // UART1
    // 'NVIC_EnableIRQ' is only required for USART setting
    NVIC_EnableIRQ(USART1_IRQn);
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;

```

- EXTI 를 NVIC 에 등록을 해야 인터럽트를 발생시킬 수 있기 때문에 설정해주는 코드이다.

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
DCD	DMA1_Channel1_IRQHandler	; DMA1 Channel 1
DCD	DMA1_Channel2_IRQHandler	; DMA1 Channel 2
DCD	DMA1_Channel3_IRQHandler	; DMA1 Channel 3
DCD	DMA1_Channel4_IRQHandler	; DMA1 Channel 4
DCD	DMA1_Channel5_IRQHandler	; DMA1 Channel 5
DCD	DMA1_Channel6_IRQHandler	; DMA1 Channel 6
DCD	DMA1_Channel7_IRQHandler	; DMA1 Channel 7
DCD	ADC1_2_IRQHandler	; ADC1 and ADC2
DCD	CAN1_TX_IRQHandler	; CAN1 TX
DCD	CAN1_RX0_IRQHandler	; CAN1 RX0
DCD	CAN1_RX1_IRQHandler	; CAN1 RX1
DCD	CAN1_SCE_IRQHandler	; CAN1 SCE
DCD	EXTI9_5_IRQHandler	; EXTI Line 9..5
DCD	TIM1_BRK_IRQHandler	; TIM1 Break
DCD	TIM1_UP_IRQHandler	; TIM1 Update
DCD	TIM1_TRG_COM_IRQHandler	; TIM1 Trigger and Commutation
DCD	TIM1_CC_IRQHandler	; TIM1 Capture Compare
DCD	TIM2_IRQHandler	; TIM2
DCD	TIM3_IRQHandler	; TIM3
DCD	TIM4_IRQHandler	; TIM4
DCD	I2C1_EV_IRQHandler	; I2C1 Event
DCD	I2C1_ER_IRQHandler	; I2C1 Error
DCD	I2C2_EV_IRQHandler	; I2C2 Event
DCD	I2C2_ER_IRQHandler	; I2C2 Error
DCD	SPI1_IRQHandler	; SPI1
DCD	SPI2_IRQHandler	; SPI2
DCD	USART1_IRQHandler	; USART1
DCD	USART2_IRQHandler	; USART2
DCD	USART3_IRQHandler	; USART3
DCD	EXTI15_10_IRQHandler	; EXTI Line 15..10
DCD	RTCAlarm_IRQHandler	; RTC alarm through EXTI line

- 위의 그림은 Interrupt Vector Table인데, 이 테이블을 보고 어떤 인터럽트를 추가할지 정하면 된다. USART를 사용하기 때문에 우선 USART1_IRQn, 조이스틱을 사용하기 때문에 각 포트번호에 해당하는 EXTI2_IRQn, EXTI9_5_IRQn S1 버튼은 PD11이므로 EXTI15_10_IRQn을 사용했다.

- Hanlder 함수를 구현할 때 Handler 함수 안에 ISR을 처리시키면 다른 인터럽트 발생 시에 지연이 발생할 수 있으므로 main task에서 ISR들을 처리한다. 전역 변수 flag를 사용하여 main task에서 ISR의 동작을 관리한다. Handler 함수에서는 flag 값을 초기화함으로 ISR을 시행시키는 트리거가 된다.

```

void EXTI2_IRQHandler(void){
    if(EXTI_GetITStatus(EXTI_Line2) != RESET){
        if(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == Bit_RESET){
            flag = 0;
        }
        EXTI_ClearITPendingBit(EXTI_Line2);
    }
}

```

```

void EXTI9_5_IRQHandler(void){
    if(EXTI_GetITStatus(EXTI_Line5) != RESET){
        if(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_5) == Bit_RESET){
            flag = 1;
        }
        EXTI_ClearITPendingBit(EXTI_Line5);
    }
}

```

```

void EXTI15_10_IRQHandler(void) { // when the button is pressed

    if (EXTI_GetITStatus(EXTI_Line11) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_11) == Bit_RESET) {
            // TODO implement
            /* send UART 1 */
            putty = 1;
        }
        EXTI_ClearITPendingBit(EXTI_Line11);
    }
}

```

```

void USART1_IRQHandler() {
    uint16_t word;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET){
        // the most recent received data by the USART1 peripheral
        word = USART_ReceiveData(USART1);

        // TODO implement, 키보드 입력 'd' 또는 'u'에 따라 동작
        if(word == 'd'){
            flag = 0;
        }
        if(word == 'u'){
            flag = 1;
        }
        // clear 'Read data register not empty' flag
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
}

```

- Handler함수 끝에는 항상 Pendingbit를 비워줘야 한다. 그렇지 않으면 ISR이 절대로 일시 정지하지 않고 무한반복 된다.
- 위에서 언급했듯이 Handler함수의 ISR을 main task에서 처리한다는 것을 위 사진으로 알 수 있다. Handler함수는 flag값을 바꿔주는 역할을 한다.
- 보드를 키면(프로그램이 실행하면) LED 1,2,3,4의 순서대로 순차점등이 되고 인터럽트가 발생하여 flag가 바뀌면 순차점등의 방향이 반대가 됨을 알 수 있다.

main()

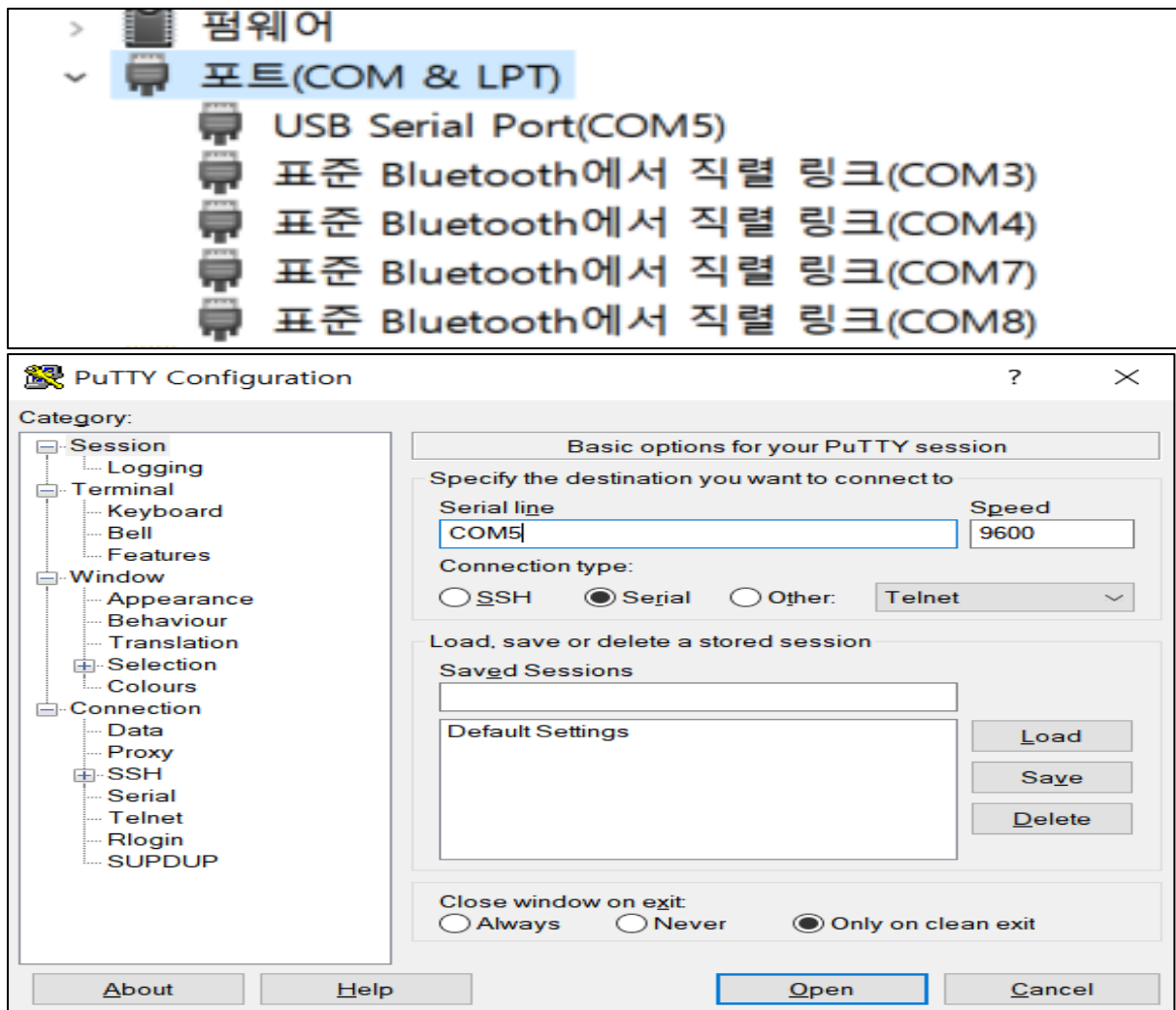
```
int main(void) {
    int i = 0;
    char msg[] = "Hello Team08\r\n";

    SystemInit();
    RCC_Configure();
    GPIO_Configure();
    EXTI_Configure();
    USART1_Init();
    NVIC_Configure();

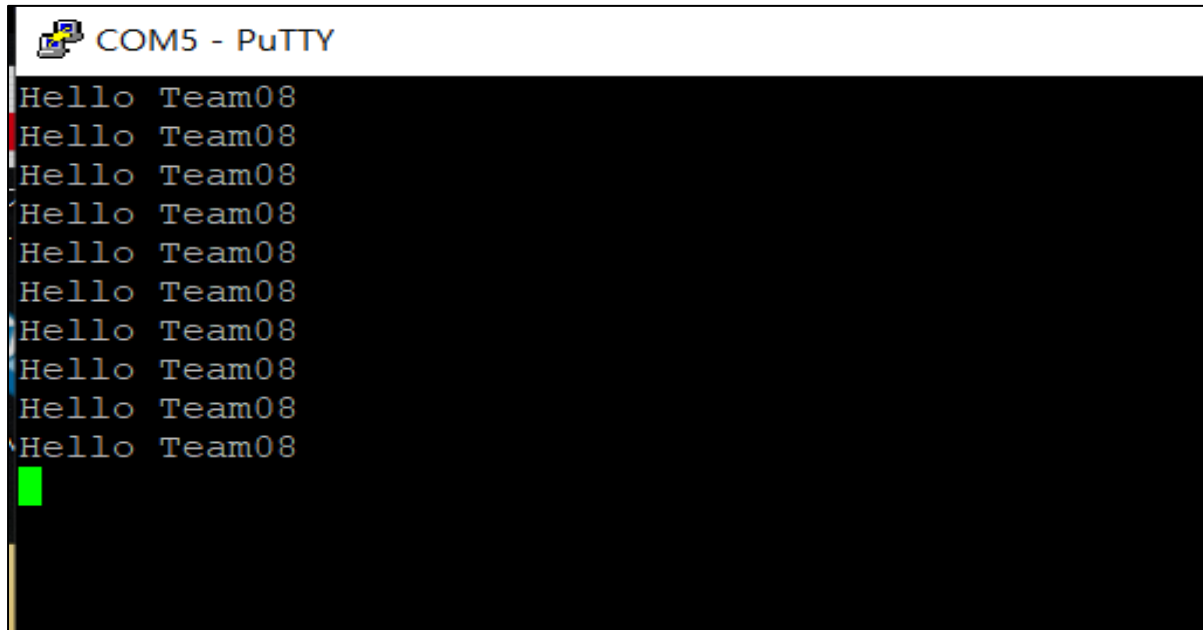
    while (1) {
        // TODO: implement
        if(i == 0){ GPIO_SetBits(GPIOD,GPIO_Pin_2); }
        else if(i == 1){ GPIO_SetBits(GPIOD,GPIO_Pin_3); }
        else if(i == 2){ GPIO_SetBits(GPIOD,GPIO_Pin_4); }
        else if(i == 3){ GPIO_SetBits(GPIOD,GPIO_Pin_7); }

        if(putty == 1){
            for(int j = 0; j<strlen(msg); j++){
                sendDataUART1(msg[j]);
            }
            putty = 0;
        }
        if(flag == 1){
            i++;
            if(i==4) i = 0;
        }
        else{
            i--;
            if(i==-1) i = 3;
        }
        // Delay
        Delay();
        GPIO_ResetBits(GPIOD,GPIO_Pin_2);
        GPIO_ResetBits(GPIOD,GPIO_Pin_3);
        GPIO_ResetBits(GPIOD,GPIO_Pin_4);
        GPIO_ResetBits(GPIOD,GPIO_Pin_7);
    }
    return 0;
}
```

- 이번 과제에서도 이전 과제와 동일하게 Putty 연결 시 장치 관리자를 확인하여 연결된 USB 포트를 확인하여 Serial line을 넣어주면 된다.



- 보드를 켜서 putty를 통한 USART 통신을 위해 putty 설정을 해준다. Serial line을 통해 연결되는 포트는 장치 관리자에서 확인 가능하다.
- S1 버튼을 누를 때마다 Hello Temp08이라는 글자가 반복적으로 출력이 되는 것을 확인할 수 있다.



```
COM5 - PuTTY
Hello Team08
Hello Team08
Hello Team08
Hello Team08
Hello Team08
Hello Team08
Hello Team08
Hello Team08
Hello Team08
Hello Team08
█
```

- 추가적으로, 키보드의 'd' 버튼과 'u' 버튼을 누를 때마다 LED 점등 순서가 뒤바뀌고, 조이스틱을 위아래로 작동했을 시에도 LED 점등 순서가 바뀌는 것도 구현을 했는데, 그것은 동영상으로 설명을 대신하겠다.

5.결론

Interrupt와 polling의 차이점에 대해 알 수 있었고, 라이브러리 함수 및 구조체를 숙지하였다. 이제까지 실험에서는 polling 방식을 통해 CPU가 event를 처리했지만, 이번 실험에서는 interrupt 방식을 통하여 GPIO를 제어하고 UART 통신을 했다. 여러 event(조이스틱 조작, 버튼입력 등)에 대해 interrupt handling을 설정하고 그 과정에서 NVIC와 EXTI의 개념에 대해 이해하고 학습하는 시간을 가질 수 있었다.