임베디드시스템

실험 및 설계

6주차 실험 보고서

2조 노윤정, 박건우, 이동근, 조영진

1. 실험목표

   1) Interrupt 방식을 활용한 GPIO 제어 및 UART 통신

   2) 라이브러리의 구조체와 함수 사용법 숙지

2. 실험 미션

   1) Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해

   2) NVIC와 EXTI를 이용하여 GPIO에 인터럽트 핸들링 세팅

   3) 보드를 켜면 LED 물결 기능 구현

   4) 조이스틱을 이용하여 LED 물결 방향 전환

   5) PC의 Putty에서 문자 입력을 통해 LED 물결 방향 전환

   6) S1 버튼 누를 경우 Putty로 "Team02" 출력

3. 실험과정

   1) 헤더파일 및 전역 변수 설정

```
#include "misc.h"
#include "stm32f10x.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"

/* function prototype */
void RCC_Configure(void);
void GPIO_Configure(void);
void EXTI_Configure(void);
void USART1_Init(void);
void NVIC_Configure(void);

void EXTI15_10_IRQHandler(void);

void Delay(void);

void sendDataUART1(uint16_t data);

/* Global Variable */
int dif, index, doUartSend;
```

라이브러리에 있는 다양한 구조체를 이용하기 위해 헤더파일을 추가해 준다. LED의 방향전환을 위한 상태 표시, UART 통신 상태 표시를 위해 전역변수를 이용하기 위해 전역변수를 선언해준다.

2) RCC Configuration

8.3.4 APB2 peripheral reset register (RCC_APB2RSTR)

Address offset: 0x0C

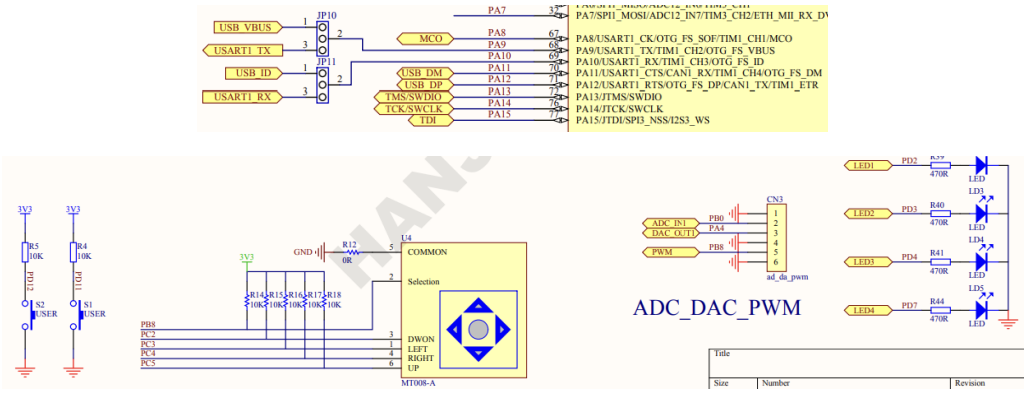
Reset value: 0x00000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1 RST	Res.	SPI1 RST	TIM1 RST	ADC2 RST	ADC1 RST	Reserved		IOPE RST	IOPD RST	IOPC RST	IOPB RST	IOPA RST	Res.	AFIO RST
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

```
27  void RCC_Configure(void) {
28      // TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'
29
30      /* UART TX/RX port clock enable */
31      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
32      /* JoyStick Up/Down port clock enable */
33      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
34      /* JoyStick Selection port clock enable */
35      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
36      /* LED port clock enable */
37      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
38      /* USART1 clock enable */
39      RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
40      /* Alternate Function IO clock enable */
41      RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
42  }
```

라이브러리의 구조체를 이용하여 실험에서 사용할 포트에 clock 신호를 인가해준다.



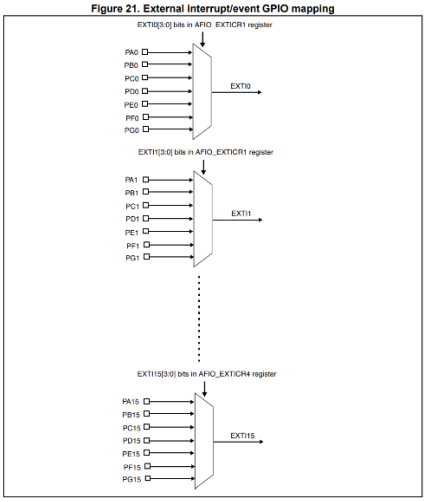
사용할 포트는 GPIOA(USART)와 GPIOC(Joystick Up/Down), GPIOB(Joystick Selection), GPIOD(LED), USART1, AFIO이기 때문에 각각의 포트를 Enable 시켜준다.

3) GPIO Configuration

```c
void GPIO_Configure(void) {
    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_IPU;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /* JoyStick selection pin setting */
    // GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    // GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    // GPIO_Init(GPIOB, &GPIO_InitStructure); // -*- don't use Joystick selection

    /* button pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    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_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

포트에 Clock을 인가한 다음 몇 번째 포트를 사용할 것인지 설정이 필요하다. Joystick에서 사용할 Up/Down 버튼은 Pin 번호가 각각 PC5, PC2이고, Switch의 Pin은 PD11이다. 4개의 LED는 PD2, PD3, PD4, PD7이며 UART pin의 TX는 PA9, PA10이므로 GPIO_InitStructure 구조체를 이용해 Pin번호를 설정하고 각각 알맞은 Mode를 설정해준다.

4) EXTI(External Interrupt) Configuration



모든 GPIO Pin들은 EXTI Line을 통해 연결되어 있다. EXTICR1 Register를 통해 입력 받을 포트를 선

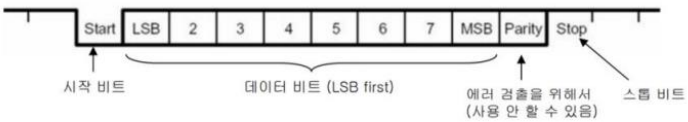
택하며 같은 번호의 핀들은 같은 Line을 공유한다. EXTI를 사용할 때는 Line, Mode, Trigger, Lineconfig 설정이 필요하다. 또한 EXTI를 선언 했을 시 반드시 Handler 구현이 필요하다. Handler 구현은 7번에서 살펴볼 것이다.

```
83  void EXTI_Configure(void) {
84      EXTI_InitTypeDef EXTI_InitStructure;
85
86      // TODO: Select the GPIO pin (Joystick, button) used as EXTI Line using function 'GPIO_EXTILineConfig'
87      // TODO: Initialize the EXTI using the structure 'EXTI_InitTypeDef' and the function 'EXTI_Init'
88
89      /* Joystick Down */
90      GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource2);
91      EXTI_InitStructure.EXTI_Line = EXTI_Line2;
92      EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
93      EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
94      EXTI_InitStructure.EXTI_LineCmd = ENABLE;
95      EXTI_Init(&EXTI_InitStructure);
96
97      /* Joystick Up */
98      GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource5);
99      EXTI_InitStructure.EXTI_Line = EXTI_Line5;
100     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
101     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
102     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
103     EXTI_Init(&EXTI_InitStructure);
104
105     /* Button */
106     GPIO_EXTILineConfig(GPIO_PortSourceGPIOD, GPIO_PinSource11);
107     EXTI_InitStructure.EXTI_Line = EXTI_Line11;
108     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
109     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
110     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
111     EXTI_Init(&EXTI_InitStructure);
112 }
```

EXTI_InitTypeDef를 이용해 EXTILine을 어떤 설정으로 Enable 할 것인지 설정한다. 가이드에서 외부 Interrupt는 EXTI10 ~ EXTI15까지 각 Port의 Pin 번호가 Interrupt Pin과 매치한다고 했으므로 Joystick Down은 GPIOC의 PinSource2, Joystick UP은 GPIOC의 PinSource5, Switch1는 GPIOD의 PinSource11로 설정해준다. 또한 가이드에서 EXTI_Mode는 Mode_Interrupt로 Trigger는 Falling으로 설정하라 하였기에 모두 Interrupt와 Falling으로 설정하였다.

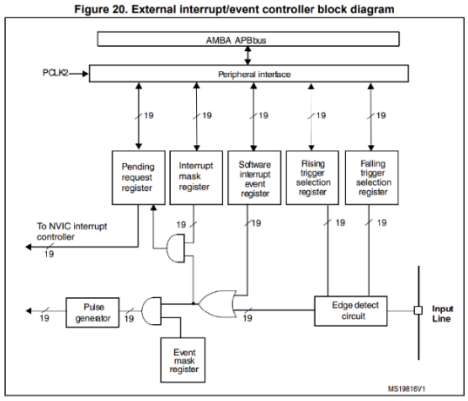
5) USART Configuration

```
114 void USART1_Init(void) {
115     USART_InitTypeDef USART1_InitStructure;
116
117     USART_Cmd(USART1, ENABLE);
118     USART1_InitStructure.USART_BaudRate = 9600;
119     USART1_InitStructure.USART_WordLength = USART_WordLength_8b;
120     USART1_InitStructure.USART_StopBits = USART_StopBits_1;
121     USART1_InitStructure.USART_Parity = USART_Parity_No;
122     USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
123     USART1_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
124     USART_Init(USART1, &USART1_InitStructure);
125     USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
126 }
```



USART 직렬통신 설정을 정의하기 위해 라이브러리의 USASRT_InitTypedef를 이용한다. USART1을 통해 데이터를 주고받는 단위인 Word의 size를 8bit로 설정하였고 USART의 Tx와 Rx를 Enable 시키기 위해 USART_Mode에서 Rx와 Tx를 설정해주었다. 가이드를 참고해 USART의 Stop Bit는 1bit로 설정하였고 USART_ITConfig를 통해 USART1 Rx의 Interrupt를 Enable 시켜 USART Interrupt가 발생할 수 있도록 만들었다.

6) NVIC(Nest Vectored Interrupt Controller) Configuration



Interrupt Request는 Mask Register를 통해 알 수 있다. Processor는 Interrupt를 인지하여 처리하기 전에 Pending Register를 검사하여 발생된 Interrupt 중 우선 순위가 가장 높은 Interrupt부터 처리한다. Interrupt 처리 중 또다른 Interrupt 발생시 우선순위를 사용하는데 우선순위가 높은 Interrupt부터 처리 후 다른 Interrupt를 처리하기 때문에 각 Interrupt의 우선 순위 설정이 필요하다. ARM 보드에서 Interrupt 사용시 NVIC를 통해 우선순위를 결정한다.

```
@code
 The table below gives the allowed values of the pre-emption priority and subpriority according
 to the Priority Grouping configuration performed by NVIC_PriorityGroupConfig function
 ============================================================================================================
   NVIC_PriorityGroup   | NVIC_IRQChannelPreemptionPriority | NVIC_IRQChannelSubPriority  | Description
 ============================================================================================================
   NVIC_PriorityGroup_0 |                0                  |            0-15             |   0 bits for pre-emption priority
                        |                                   |                             |   4 bits for subpriority
 ------------------------------------------------------------------------------------------------------------
   NVIC_PriorityGroup_1 |              0-1                  |             0-7             |   1 bits for pre-emption priority
                        |                                   |                             |   3 bits for subpriority
 ------------------------------------------------------------------------------------------------------------
   NVIC_PriorityGroup_2 |              0-3                  |             0-3             |   2 bits for pre-emption priority
                        |                                   |                             |   2 bits for subpriority
 ------------------------------------------------------------------------------------------------------------
   NVIC_PriorityGroup_3 |              0-7                  |             0-1             |   3 bits for pre-emption priority
                        |                                   |                             |   1 bits for subpriority
 ------------------------------------------------------------------------------------------------------------
   NVIC_PriorityGroup_4 |             0-15                  |              0              |   4 bits for pre-emption priority
                        |                                   |                             |   0 bits for subpriority
 ============================================================================================================
@endcode
```

Pre-emption Priority는 우선순위가 높은 Interrupt가 들어오면, 현재 작업을 멈추고 해당 Interrupt를 진행하게 된다. Pre-emption Priority를 통해 선점 우선순위가 결정된다. 반면, Sub Priority는 Interrupt 수행 중이 아닌, 아직 대기 중에 있어서 우선순위가 높은 것을 먼저 수행한다. 따라서 Pre-emption Priority가 동일한 경우 Sub Priority로 우선순위를 결정한다. 숫자 값이 작을수록 우선순위가 높다.

```c
void NVIC_Configure(void) {
    NVIC_InitTypeDef NVIC_InitStructure;

    // TODO: fill the arg you want
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    // TODO: Initialize the NVIC using the structure 'NVIC_InitTypeDef' and the function 'NVIC_Init'
    // Joystick Down
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;  // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;         // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Joystick Up
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;  // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;         // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // User S1 Button
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;         // TODO
    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;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;  // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;         // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

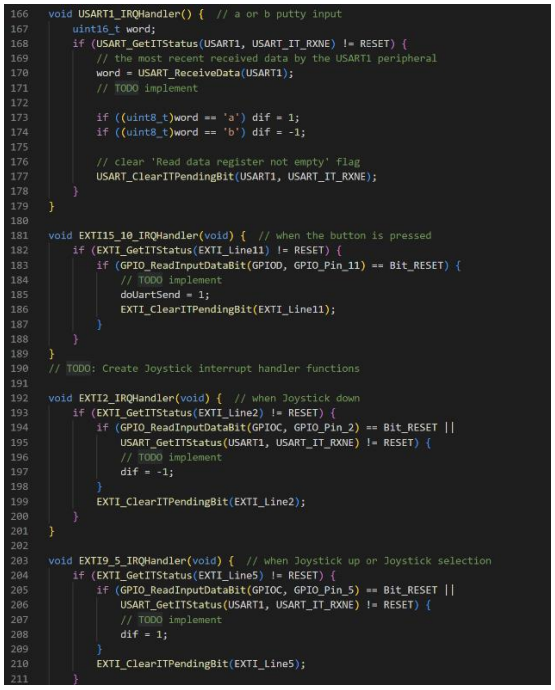
Switch의 경우 어떠한 상황에도 상관없이 항상 Interrupt가 발생해야 하므로 Pre-emption Priority를 0으로 설정했다. Joystick의 경우 Up인 경우 Down인 경우 모두 Pre-emption Priority에서는 동일해도 상관없기 때문에 Switch 다음 우선순위인 1로 설정해주었다. 마지막으로 남은 UART1에 대한 Pre-emption Priority 마지막 우선순위인 2로 설정해주었다.

7) IRQ Handler – 각 Interrupt들이 발생했을 때 처리할 작업 정의

```
        ; 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
        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        ; I2C1 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
```

각 Interrupt가 발생했을 때 처리할 작업을 정의하기위해 각 Interrupt handler에서 호출되는 함수의 프로토타입이 정의된 것을 참고하여 구현한다. Switch1은 PD11이므로 EXTI15_10_IRQHandler를, Joystick Down은 PC2이므로 EXTI2_IRQHandler를, UP은 PC5이므로 EXTI9_5_IRQHandler를, USART1

의 경우 USART1_IRQHandler를 Handler이름으로 선언해준다.



```
void USART1_IRQHandler() {  // a or b putty input
    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

        if ((uint8_t)word == 'a') dif = 1;
        if ((uint8_t)word == 'b') dif = -1;

        // clear 'Read data register not empty' flag
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
}

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
            doUartSend = 1;
            EXTI_ClearITPendingBit(EXTI_Line11);
        }
    }
}
// TODO: Create Joystick interrupt handler functions

void EXTI2_IRQHandler(void) {  // when Joystick down
    if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == Bit_RESET ||
            USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
            // TODO implement
            dif = -1;
        }
        EXTI_ClearITPendingBit(EXTI_Line2);
    }
}

void EXTI9_5_IRQHandler(void) {  // when Joystick up or Joystick selection
    if (EXTI_GetITStatus(EXTI_Line5) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_5) == Bit_RESET ||
            USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
            // TODO implement
            dif = 1;
        }
        EXTI_ClearITPendingBit(EXTI_Line5);
    }
```

USART1의 경우 Putty를 통해 입력되는 문자가 'a' 라면 전역변수인 dif를 1로, 'b' 입력 시 dif를 -1로 바뀌도록 설정해주었다. USART_ClearITPendingBit를 통해 Pending Bit를 clear 해주며 시스템이 Interrupt가 처리되었다는 것을 인지하도록 한다. 만약 이를 수행하지 않으면 시스템은 다시 Interrupt가 발생한 것으로 인지하여 계속해서 Handler를 호출하게 된다.

Switch1의 경우 Interrupt가 발생했을 때 전역변수인 doUartSend를 1로 설정할 수 있도록 해주었다. Joystick의 Down에서 Interrupt가 발생했을 시 전역변수인 dif를 -1로, Up 혹은 Selection에서 Interrupt 발생 시 dif가 1로 바뀔 수 있도록 설정해주었다. Switch와 Joystick의 경우 모두 마찬가지로 ClerITPendingBit를 통해 Pending Bit를 clear 해주었다.

8) Main Function

```c
int main(void) {
    SystemInit();

    RCC_Configure();

    GPIO_Configure();

    EXTI_Configure();

    USART1_Init();

    NVIC_Configure();
    index = 0;
    dif = 1;
    doUartSend = 0;
    char msg[] = "Team02.\r\n";  // -*- length = 9;
    while (1) {
        // TODO: implement
        GPIOD->BRR = GPIO_BRR_BR2 | GPIO_BRR_BR3 | GPIO_BRR_BR4 | GPIO_BRR_BR7;
        switch (index) {
            case 0:
                GPIOD->BSRR |= GPIO_BSRR_BS2;
                break;
            case 1:
                GPIOD->BSRR |= GPIO_BSRR_BS3;
                break;
            case 2:
                GPIOD->BSRR |= GPIO_BSRR_BS4;
                break;
            case 3:
                GPIOD->BSRR |= GPIO_BSRR_BS7;
                break;
        }
        if (doUartSend) {
            for (int i = 0; i < 9; i++) {
                sendDataUART1(msg[i]);
            }
        }
        doUartSend = 0;
        index += dif;
        if (index > 3) index = 0;
        if (index < 0) index = 3;
        // Delay
        Delay();
    }
    return 0;
}
```

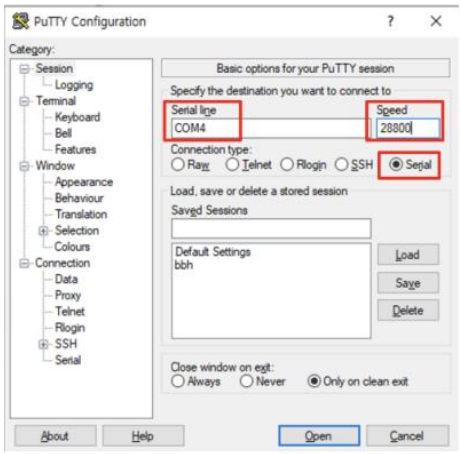
제일 처음 선언했던 전역변수 index, dif, doUartSend를 차례로 0, 1, 0으로 초기화 시켜준다. Switch 작동 시 Putty로 출력될 길이가 9인 문자열 "Team02."도 설정해준다. 먼저, 프로그램이 종료되지 않고 반복적으로 신호를 보내도록 하기 위해 while문 안에서 작동하도록 만들었다. 우선 4개의 LED를 모두 꺼지게 하고 switch문을 통해 특정 상황에 각각의 LED가 켜지도록 만들었다.

아무런 버튼을 누르지 않았을 때 index = 0, dif = 1이므로 case 0에서 LED1이 켜지게 되고 이후 index+=dif 연산을 통해 index=1로 바뀌게 된다. while문에 의해 다시 switch문으로 돌아오고 case1에서 LED2가 켜지게 되며 동일하게 순서대로 LED3, LED4가 켜지게 된다. 이 경우 index 값이 1씩 증가하게 되는데 3을 초과하면 다시 index=0으로 설정해줌으로써 다시 LED1이 켜지게 된다. Joystick의 Down을 누르게 되면 dif가 -1로 바뀌게 되고 index +=dif 연산을 통해 index 값이 -1씩 감소하게된다. 따라서 LED 또한 번호가 큰 LED에서 작은 순으로 켜지게 된다. 이후 index 값이 0보다 작게 되면 다시 index=3으로 설정해줌으로써 다시 LED4부터 번호가 작은 LED로 순서대로 켜지게 된다. Putty를 통해 입력 받는 문자 'a'와 'b'에 의해서도 index값이 변하게 되는데 이 또한 Joystick에 의한 변화와 같은 원리로 LED가 켜지는 방향을 바꾼다.

Switch1을 눌렸을 경우 doUartSend의 값이 1로 변하게 되는데 이는 while문 안에 있는 if(doUartSend)문이 동작하게 만든다. If문 안의 for문이 작동하면서 msg안에 저장한 "Team02."가 순

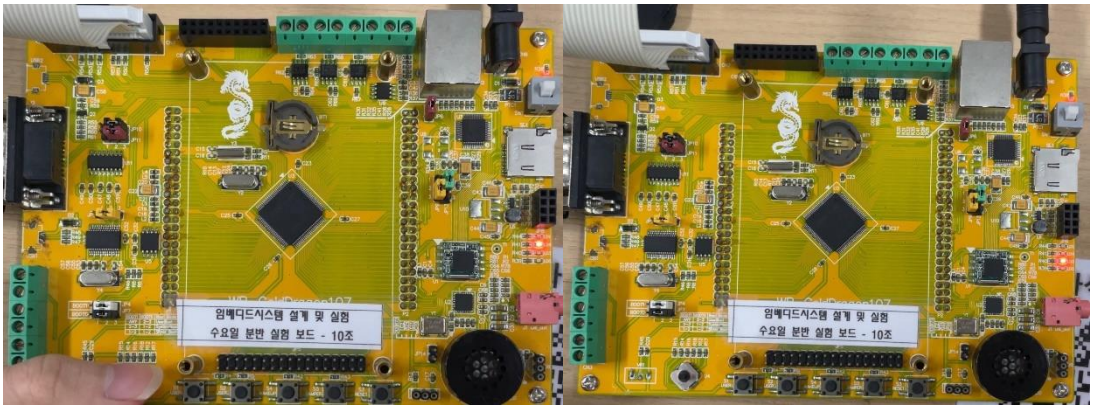
서대로 출력하게 된다. 이후 doUartSend=0으로 되돌려 줌으로써 Switch1에 입력이 들어왔을 때 이를 감지할 수 있게 만들어 주었다.

9) Serial Configuration

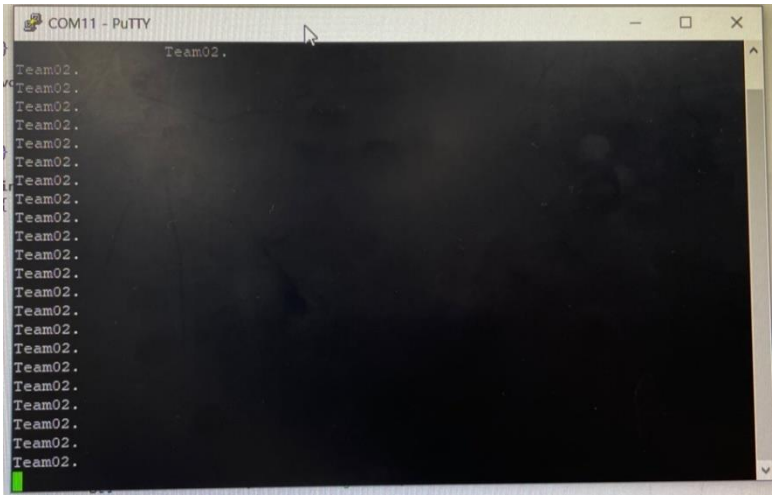


실습 컴퓨터와 실험보드를 연결해 Putty를 실행한 후, Connection Type은 Serial로 맞춘 뒤, 장치관리자의 포트 항목을 참고해 Serial line을 COM11로, Speed 부분은 28.8KHz로 설정하였다.

4. 실험결과



왼쪽 아래에 있는 Joystick을 Up으로 조작 시, LED가 1->2->3->4 방향으로 켜지고, Down으로 조작 시, LED가 4->3->2->1 방향으로 켜진다. 또한 PC의 Putty에서 'a'문자를 입력하면 LED가 1~4 방향으로 켜지고 'b'문자를 입력하면 4~1 방향으로 동작한다.



Switch1을 누르면 putty로 Team02.가 출력된다.

5. 고찰

실험 중, 중간 점검을 위해 보드에 코드를 올려봤으나 보드에 새로운 코드가 제대로 올라가지 않는 문제가 발생했다. 코드에 문제가 있는가 싶어 다시 살펴보기도 하고 프로그램 설정에 문제가 있는가 싶었지만, 오류를 발견하지 못하고 뒤늦게 프로젝트 파일을 새로 만들었다. 그랬더니 보드에 코드가 올라가지 않는 문제점을 해결할 수 있었다. 하지만 이후 LED에 불이 들어오지 않는 문제가 발생했다. 또한 저번 실습 때처럼 Putty에 출력되는 텍스트 일부가 깨지는 현상과 계단식으로 출력되는 현상도 확인할 수 있었다. 저번 실험 때 찾은 해결 방법처럼 Putty에서의 Speed 부분을 여러 가지로 변경해 봤지만 해결되지 않았고, LED 문제 또한 코드에서의 문제를 찾지 못했다. 결국 보드를 바꿔서 다시 진행을 해봤으나 해결이 되지 않았고 코드를 하나씩 확인해보았다. 다시 보니 USART Configuration 코드에서 Rx와 Tx를 제대로 Enable 시켜주지 않았고 이 때문에 Putty에서 오류가 발생하는 것이었다. 또한 정리되지 않았던 코드들을 가지런히 정리하니 원하는 코드대로 진행이 되었다. 이를 통해 복잡한 코드보다 단순하고 사소한 부분일수록 실수하면 알아채기 더 어렵고 작은 문제라도 프로그램 전체가 작동하지 못할 만큼 치명적인 문제로 이어질 수 있으므로 더욱 신경 써서 진행해야 한다는 점을 한 번 더 되새길 수 있었다.

또한 이번 실험에서는 우선순위가 Switch를 제외하면 큰 영향을 끼치지 않기 때문에 순위가 상관이 없었지만 이후 우선순위가 중요한 다른 실험을 하게 된다면 Interrupt의 성격에 맞는 우선순위 설정에 유의하는 것이 도움이 되리라 생각한다.