임베디드 시스템 설계 및 실험
화요일 분반 6주차 실험 결과 보고서

조 : 6조
조원 : 이주승 김선규 이동현 이지현 최세희

실험목표

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

세부 목표

1. Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해
2. NVIC와 EXTI를 이용하여 GPIO에 인터럽트 핸들링 세팅

보드를 켜면 LED 물결 기능 유지 (LED 1->2->3->4->1->2->3->4->1->… 반복)
A LED 물결 방향 변경 - 1->2->3->4
B LED 물결 방향 변경 - 4->3->2->1

3. 조이스틱을 Up : A 동작, 조이스틱 Down : B 동작
4. PC의 Putty에서 a, b 문자 입력하여 보드 제어 (PC -> 보드 명령)
   ('a' : A 동작, 'b' : B 동작)
5. S1 버튼을 누를 경우 Putty로 "TEAM06.₩r₩n" 출력

이론적 배경

1. Polling vs Interrupt
   ① Polling이란 CPU가 특정 이벤트를 처리하기 위해 이벤트가 발생할 때까지 모든 연산을 이벤트가 발생하는지 감시하는 방식을 말한다.

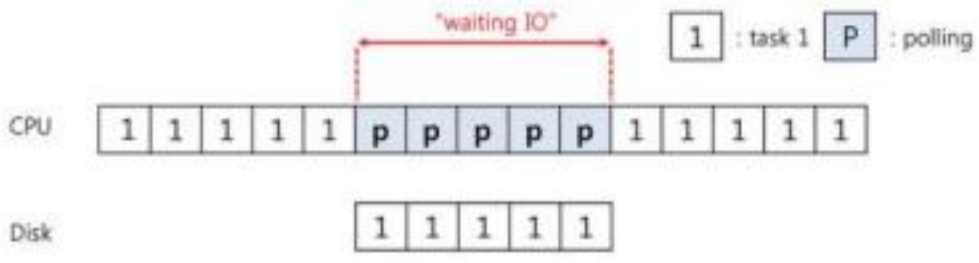


그림 1 : Polling 방식을 시각으로 표현.

   ② Interrupt 란 CPU 가 특정 이벤트 발생시 현재 작업을 멈추고 해당 인터럽트 서비스 루틴을 수행 후 다시 이전 작업으로 돌아가는 방식을 말한다.

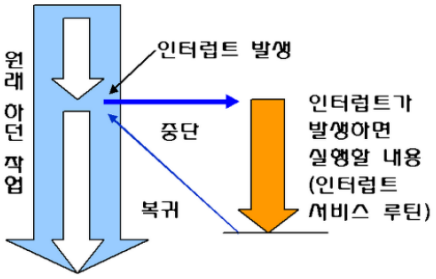


그림 2 : Interrupt 방식을 시각으로 표현.

2. Interrupt

① Hardware Interrupt vs Software Interrupt

Hardware Interrupt 는 비동기식 이벤트 처리로 주변장치의 요청에 의해 발생하는 Interrupt 로 높은 선 순위를 가지며, 하드 디스크 읽기 요청, 디스크 읽기 끝남, 키보드 입력 등에 발생한다.

Software Interrupt 는 동기식 이벤트 처리로 사용자가 프로그램 내에서 Interrupt 가 발생하도록 설정하는 인터럽트이다. 낮은 우선 순위를 가지며 Trap, Exception 이 여기에 해당된다.

② EXTI(External Interrupt)

EXTI 는 외부에서 신호가 입력될 경우 Devide 에 Event 나 Interrupt 가 발생되는 기능을 말한다.

입력 받을 수 있는 신호는 Rising-Edge, Falling-Edge, Rising & Falling-Edge 가 있다.

각 Port 의 n 번 Pin 의 EXTI n 에 연결해야 한다.

EXTI 는 Event Mode 와 Interrupt Mode 를 선택하여 설정 가능하다.

Interrupt Mode 로 설정할 경우 Interrupt 가 발생해 해당 Interrupt Handler 가 동작하며, 20 개의 Edge Detector Line 으로 구성되어 각 Line 이 설정에 따라 Rising/Falling Trigger 를 감지한다.

모든 GPIO 핀들은 EXTI line 을 통해 연결되어 있다

그림 3 의 내용을 참고하여 각 Interrupt Handler 에서 호출되는 함수의 프로토타입이 정의되어 있는 것을 확인할 수 있고, 정의된 이름을 그대로 사용하여 원하는 함수를 구현할 수 있다.

예) 9 번 핀을 사용 시에 Handler 이름은 EXIT9_5_IRQHandler(5~9PIN)로 선언하면 된다.

```
        ; 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
```

그림 3 : Libraries₩CMSIS₩DeviceSupport₩Startup₩startup_stm32f10x_cl.s

③ NVIC(Nested Vectored Interrupt Controller)

NVIC 는 여러 Interrupt 를 관리해주는 컨트롤러이다.

이는 Interrupt 처리 중 또다른 Interrupt 발생시 우선순위를 정의한다. 우선순위가 높은 Interrupt 부터 처리 후 다른 Interrupt 를 순차적으로 처리한다.

ARM 보드에서 Interrupt 사용시 NVIC 통하여 우선순위를 경정하게 되는데 이는 그림 4 를 참고하여 원하는 순위를 지정할 수 있다. 여기서 지정한 값이 낮을수록 우선순위가 높게 지정된다.

```
ode
he table below gives the allowed values of the pre-emption priority and subpriority according
o 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
```

그림 4 : Libraries₩STM32F10x_StdPeriph_Driver_v3.5₩inc₩misc.h

구현 내용

1. Clock Enable

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

        /* UART TX/RX port clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //Fixed

        /* JoyStick Up/Down port clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); //Fixed GPIOC >> PC5-UP, PC2-DOWN

        /* LED port clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); //Fixed GPIOD >> PD2,3,4,7 - L1,L2,L3,L4

        /* USART1 clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE); //Fixed USART

        /* Alternate Function IO clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

실험에 필요한 GPIO A(TX/RX), GPIO C(조이스틱 Up, Down), GPIO D(LED), USART1, AFIO Clock을 Enable한다.

2. GPIO Configuration

```
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_5 | GPIO_Pin_2;      //Fixed 5 : UP, 2 : DOWN
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;               // Fixed IPD : INPUT PULL-DOWN
    GPIO_Init(GPIOC, &GPIO_InitStructure);                      // Fixed > GPIOC SET

    /* button pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;                  //Fixed 11: BUTTON S1
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;               // Fixed IPD : INPUT PULL-DOWN
    GPIO_Init(GPIOD, &GPIO_InitStructure);                      // Fixed > GPIOD SET

    /* 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_InitTypeDef GPIO_InitStructure_TX;                     //Fixed 변수선언
    GPIO_InitStructure_TX.GPIO_Pin = GPIO_Pin_9;                   //Fixed 9 : TX
    GPIO_InitStructure_TX.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure_TX.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure_TX);

    //RX
    GPIO_InitTypeDef GPIO_InitStructure_RX;                     //Fixed 변수선언
    GPIO_InitStructure_RX.GPIO_Pin = GPIO_Pin_10;                  //Fixed 10 : RX
    GPIO_InitStructure_RX.GPIO_Mode = GPIO_Mode_IPU|GPIO_Mode_IPD; //Fixed
    GPIO_Init(GPIOA, &GPIO_InitStructure_RX);
}
```

각 GPIO의 Pin을 InitTypeDef 구조체를 사용하여 설정한다.

GPIO C에서 조이스틱 Up(Pin 5), Down(Pin 2)을 Input Pull-Down mode로 설정, GPIO D에서 버튼 S1(Pin 11)을 Input Pull-Down mode로 설정, LED(Pin 2, 3, 4, 7)를 Output Push-Pull mode로 설정, GPIO A에서 TX(Pin 9)를 Alternative Function Push-Pull mode로 설정, RX(Pin 10)를 Input Pull-Up, Input Pull-Down mode로 설정한다. 마지막으로 GPIO_Init 함수를 통해 initialize한다.

3. EXTI Configuration

```
void EXTI_Configure(void)
{
    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;                    //Fixed C5 : UP
    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;                    //Fixed D11 : BUTTON S1
    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
}
```

조이스틱 Down의 Pin이 2번이므로 EXTI Line 2에 할당한다. 조이스틱 Up과 Button도 같은 방법으로 각각 EXTI Line 5와 EXTI Line 11에 할당한다. Mode는 Interrupt, Trigger Falling으로 설정하고 Enable한다.

4. USART Initialization

```
void USART1_Init(void)
{
        USART_InitTypeDef USART1_InitStructure;

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

        // TODO: Initialize the USART using the structure 'USART_InitTypeDef' and the function 'USART_Init'
        //Week 5 TODO6~10 참고
        USART1_InitStructure.USART_BaudRate = 9600;                                         //Fixed
        USART1_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;    //Fixed Week5 TODO_10
        USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;                    //Fixed
        USART1_InitStructure.USART_Parity = USART_Parity_No;                                //Fixed
        USART1_InitStructure.USART_StopBits = USART_StopBits_1;                             //Fixed
        USART1_InitStructure.USART_WordLength = USART_WordLength_8b;                        //Fixed 8b : 8Bit

        USART_Init(USART1, &USART1_InitStructure);                                           //Fixed stm32f19x_usart.h

        /* TODO: Enable the USART1 RX interrupts using the function 'USART_ITConfig'
           and the argument value 'Receive Data register not empty interrupt' */
        USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);                                      //Fixed stm32f19x_usart.c

}
```

```
/**
  * @brief  Enables or disables the specified USART interrupts.
  * @param  USARTx: Select the USART or the UART peripheral.
  *   This parameter can be one of the following values:
  *   USART1, USART2, USART3, UART4 or UART5.
  * @param  USART_IT: specifies the USART interrupt sources to be enabled or disabled.
  *   This parameter can be one of the following values:
  *     @arg USART_IT_CTS:  CTS change interrupt (not available for UART4 and UART5)
  *     @arg USART_IT_LBD:  LIN Break detection interrupt
  *     @arg USART_IT_TXE:  Transmit Data Register empty interrupt
  *     @arg USART_IT_TC:   Transmission complete interrupt
  *     @arg USART_IT_RXNE: Receive Data register not empty interrupt
  *     @arg USART_IT_IDLE: Idle line detection interrupt
  *     @arg USART_IT_PE:   Parity Error interrupt
  *     @arg USART_IT_ERR:  Error interrupt(Frame error, noise error, overrun error)
  * @param  NewState: new state of the specified USARTx interrupts.
  *   This parameter can be: ENABLE or DISABLE.
  * @retval None
  */
```

5주차의 USART 설정과 라이브러리를 참고하여 USART 설정을 정의한다.

Baud rate는 9600, Hardware flow control은 None, USART mode는 RX/TX, Parity bit는 Disable, Stop bits는 1bit, Wordlength는 8bit로 설정한다.

USART interrupt source를 Receive Data register not empty interrupt로 설정한다. stm32f10x_usart.c 파일을 참고하여 parameter에 USART_IT_RXNE를 적용한다.

5. NVIC Configuration

```
void NVIC_Configure(void) {

    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure_UP;
    NVIC_InitTypeDef NVIC_InitStructure_DOWN;
    NVIC_InitTypeDef NVIC_InitStructure_BUTT;

    // TODO: fill the arg you want
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //Fixed pre-enption 2bit, sub priority 2bit

    // TODO: Initialize the NVIC using the structure 'NVIC_InitTypeDef' and the function 'NVIC_Init'

    //부팅시 업부터 해야 순서대로 동작함.
    //UP
    NVIC_InitStructure_UP.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure_UP.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure_UP.NVIC_IRQChannelPreemptionPriority = 0x01; // TODO
    NVIC_InitStructure_UP.NVIC_IRQChannelSubPriority = 0x00; // TODO
    NVIC_Init(&NVIC_InitStructure_UP);

    //DOWN
    NVIC_InitStructure_DOWN.NVIC_IRQChannel = EXTI2_IRQn;
    NVIC_InitStructure_DOWN.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure_DOWN.NVIC_IRQChannelPreemptionPriority = 0x02; // TODO
    NVIC_InitStructure_DOWN.NVIC_IRQChannelSubPriority = 0x00; // TODO
    NVIC_Init(&NVIC_InitStructure_DOWN);

    //BUTTON
    NVIC_InitStructure_BUTT.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure_BUTT.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure_BUTT.NVIC_IRQChannelPreemptionPriority = 0x03;  // TODO
    NVIC_InitStructure_BUTT.NVIC_IRQChannelSubPriority = 0x00;  // TODO
    NVIC_Init(&NVIC_InitStructure_BUTT);
    // UART1
    //NVIC_InitTypeDef  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 = 0x00 ; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

NVIC에서는 pre-emption priority와 sub priority 설정을 통해 Interrupt의 우선 순위를 정할 수 있다. 먼저 pre-emption priority로 우선 순위를 결정하고, 그 값이 같을 경우 sub priority로 우선 순위를 결정한다. 그 값이 다르면 sub priority는 우선 순위에 영향을 미치지 않는다. 값이 작을수록 우선 순위가 높으므로, 먼저 USART 세팅을 위해 USART1 Interrupt를 enable하고, 나머지 동작들도 순서에 맞게 우선 순위를 부여한다.

구현 기능이 4가지이므로 pre-emption과 sub priority 모두 4가지의 우선순위까지 설정 가능한 Priority group 2를 선택하고 각각의 Pin 번호에 맞게 IRQ channel을 설정한다.

6. IRQHandler 정의

각 IRQHandler에 Interrupt 발생 시 실행할 동작을 정의한다.

① USART

```
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
        if (word == 'a')                //Up
        {
          mode = 0;
        }
        else if (word == 'b')           //Down
        {
          mode = 1;
        }

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

USART Interrupt 발생 시 입력 받은 Data가 'a'이면 전역변수 'mode'의 값을 0으로(동작 A), 'b'이면 전역변수 'mode'의 값을 1로(동작 B)로 설정한다.

② S1 Button

```
void EXTI15_10_IRQHandler(void) { //Fixed button press

        if (EXTI_GetITStatus(EXTI_Line11) != RESET) {
                if (GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_11) == Bit_RESET) {
                        // TODO implement
                        print = 1;
                }
        EXTI_ClearITPendingBit(EXTI_Line11);
        }
}
```

S1 Button Interrupt 발생 시 전역변수 'print'의 값을 1로 설정하여 main 함수에서 UART 통신을 통해 Putty로 출력할 데이터를 보낼 수 있도록 한다.

③ 조이스틱 Up/Down

```
// TODO: Create Joystick interrupt handler functions
/* --- TODO START - Add IRQHandler functions */
void EXTI9_5_IRQHandler(void) { //Fixed JoyStick UP

        if (EXTI_GetITStatus(EXTI_Line5) != RESET) {
                if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_5) == Bit_RESET) {
                        mode = 0; // Change mode A when joystick is UP
                }
        EXTI_ClearITPendingBit(EXTI_Line5);
        }
}

void EXTI2_IRQHandler(void) { //Fixed JoyStick DOWN

        if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
                if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == Bit_RESET) {
                        mode = 1; // Change mode B when joystick is DOWN
                }
        EXTI_ClearITPendingBit(EXTI_Line2);
        }
}
```

조이스틱 Up/Down Interrupt 발생 시 각각 전역변수 'mode'의 값을 0(동작 A), 1(동작 B)로 설정한다.

7. LED 및 Putty 동작 정의

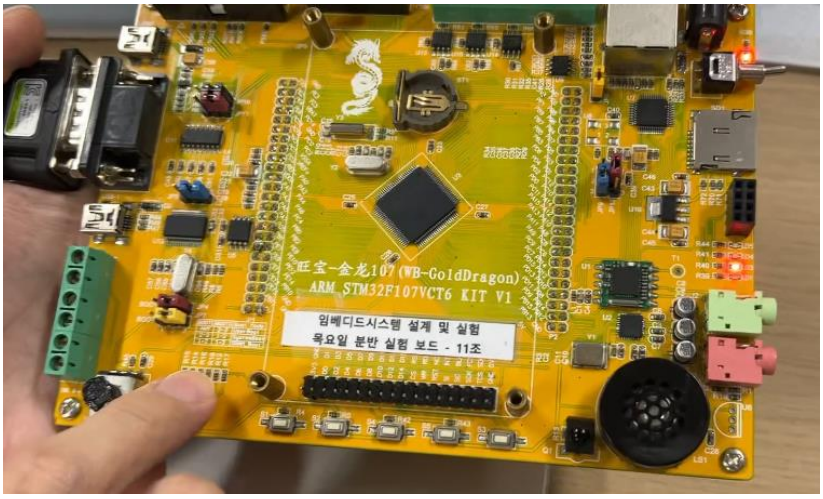
```
    // TODO: implement
    uint16_t led[4] = {
      GPIO_Pin_2,
      GPIO_Pin_3,
      GPIO_Pin_4,
      GPIO_Pin_7
    };

    while (1) {
      GPIO_SetBits(GPIOD, led[led_out]);
      Delay();
      GPIO_ResetBits(GPIOD, led[led_out]);
      if (mode == 0) {
        while (led_out < 0) {
          led_out += 4;
        }
        led_out = (led_out+1)%4;
      }
      else {
        while (led_out < 0) {
          led_out += 4;
        }
        led_out = (led_out-1 +4)%4;
      }
      if (print == 1) {
        char *tmp = &puttyPrint[0];
        while (*tmp != '\0') {
          sendDataUART1(*tmp);
          tmp++;
        }
        print = 0;
      }
    }
    return 0;
```

전역변수 ‘mode’의 값이 0이면 동작 A를 수행하고, 1이면 동작 B를 수행한다. GPIO_SetBits() 함수로 LED를 켜고, GPIO_ResetBits() 함수로 LED를 끄며, 두 함수 사이에 Delay() 함수를 실행시켜 LED가 켜지고 꺼지는 간격을 조절한다.

전역변수 ‘print’의 값이 1이면 S1 Button을 눌러 Interrupt가 발생한 것이므로 UART 통신으로 Putty에 출력할 값을 전송한 후 ‘print’의 값을 다시 0으로 설정하여 중복 출력을 방지한다.

실험 결과

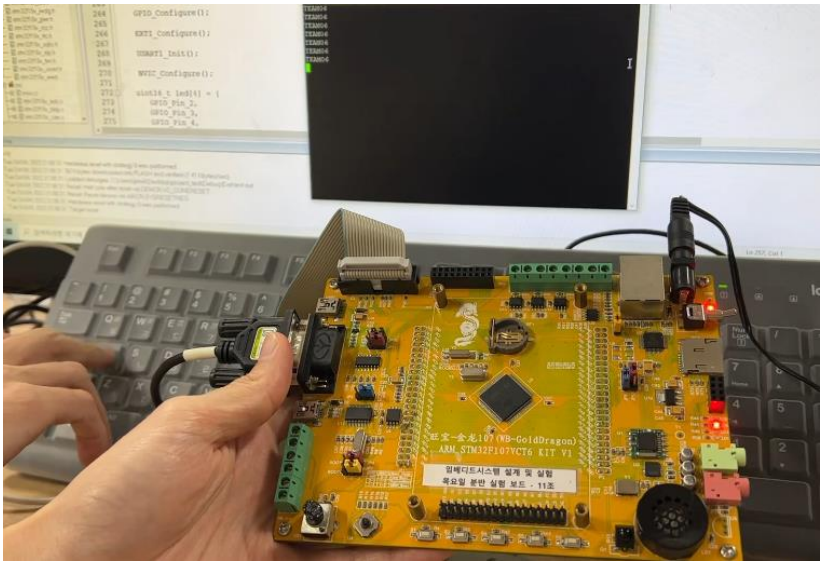
1. 조이스틱, LED 동작 및 Interrupt 구현



보드에 전원이 들어오면 LED가 1>2>3>4>1 순차적으로 점멸하며 반복한다.
조이스틱을 DOWN 시 인터럽트가 발생하여 LED가 1>2>3>4>1을 반대로 4>3>2>1>4 순으로 점멸이 바뀐다. 다시 조이스틱을 UP을 입력하면 또다시 인터럽트가 발생하여 LED가 1>2>3>4>1 순으로 바뀐다.

2. UART 통신으로 Putty 입출력

보드와 Putty에 연결한 후 조이스틱 Select를 입력하면 Putty MSG에 “TEAM06”이 출력된다.
컴퓨터 키보드에 ‘a’를 입력하게 되면 보드의 LED가 1>2>3>4>1 순으로 점멸한다.
컴퓨터 키보드에 ‘b’를 입력하게 되면 보드의 LED가 4>3>2>1>4 순으로 점멸한다.



어려웠던 점

1. USART_ITConfig(USART1, USART_IT_RXNE, ENABLE)를 하는 과정에 있어서 관련된 정보를 계속 stm32f19x_usart.h에서 찾으려고 했다. 여기서 계속 시간을 낭비하게 되었지만 stm32f19x_usart.c에 자세한 정보가 있다는 것을 알게 되었고 실험을 이어 나갈 수 있었다.

2. 보드에 있는 장치를 조작하는 것은 이제 어느 정도 익숙해져서 조작을 수월하게 할 수 있었지만, 지난주 실험에 나왔던 USART에 대한 조작은 아직까지 익숙하지 않았기 때문에 다소 조작하는 데 있어서 시간이 걸렸다. 또한 지난주 수업내용을 다시 열어봐서 조작 방법을 알아보는 번거로움이 있었다.