2021년 2학기
임베디드 시스템설계 및 실험
7주차 실험보고서

201712159 조현우
201724566 전승윤
201724611 최호진
201924650 박지호

목차

1. 실험 목적

1. Interrupt 방식을 활용한 GPIO 제어 및 UART 통신

2. 라이브러리 함수 사용법 숙지

2. 실험 과정

1) Datasheet 및 Reference Manual을 참고해 사용할 Port의 RCC, GPIO 설정

2) NVIC와 EXTI를 이용해 GPIO에 인터럽트 핸들링 세팅

3) 명시된 조건으로 작동하기 위해 동작 코드 작성

3. 실험 세부 내용

1) 사용할 Port의 RCC, GPIO 설정

먼저 사용할 Port의 RCC enable과 GPIO configuration이 필요하다. 사용하는 Port들은 다음 표와 같다.

이름	Input/Output	포트
UART RX	Input	PA10
JoyStick Up/Down		PC5, 2
User S1 button		PC2
LED Port(1,2,3,4)	Output	PD2, 3, 4, 7
UART TX		PA9

표 1 - Ports

또한 USART1과 Alternate Fuctnion IO에도 RCC clock enable을 해야 한다.

이번 실험은 전 주차 실험과 달리 헤더 파일에 정의되어 있는 구조체와 함수를 사용한다. 먼저 RCC enable를 하는 함수는 다음과 같다.

```
/* UART TX/RX port clock enable */
/* PA9, 10 */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
```

그림 1 - RCC enable

마우스 오른쪽 버튼을 눌러 함수와 변수의 정의를 쉽게 찾아 볼 수 있다. RCC_APB2PeriphClockCmd()에 RCC_APB2Perih_GPIOX(X는 Port의 종류이다.)와 ENABLE을 넣어 사용할 Port의 Clock Enable을 부여할 수 있다.

```
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_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

그림 2 - GPIO Configuration by structure

다음은 GPIO를 설정하는 코드이다. 맨 위에 미리 정의된 구조체로 변수를 만든 뒤 구조체 변수에 사용할 Port와 Mode를 부여해 GPIO_Init()의 인자로 넣어 원하는 설정을 하는 코드이다.

2) NVIC와 EXTI를 이용해 GPIO에 인터럽트 핸들링 세팅

Interrupt 사용을 위해서는 EXTI와 NVIC 설정이 필요하다. 그 중 EXTI는 External Interrupt/event controller의 약자로 외부에서 온 신호를 받아 Interrupt 신호로 바꿔준다. EXTI는 같은 Port 번호를 가진 신호만 받으므로 Interrupt가 필요한 Port를 미리 알 필요가 있다. 이번 실험에서 Interrupt가 필요한 장치는 다음과 같다.

이름	Input/Output	포트
JoyStick Up/Down	Input	PC5, 2
User S1 button		PD11

표 2 - Ports which need EXTI Configuration

UART는 EXTI 대신 UART 전용의 Interrupt를 통해 설정한다. 그러므로 EXTI 설정은 위 3개의 동작에서만 하면 된다.

```
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);
```

그림 3 - EXTI Configuraton

다음 코드는 EXTI 설정하는 코드이다. 위의 GPIO 때처럼 미리 정의되어 있는 구조체를 활용해 원하는 값을 주어 EXTI_Init()를 통해 설정하는 모습이다.

```
USART_InitTypeDef USART1_InitStructure;

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

USART1_InitStructure.USART_BaudRate = 9600;
USART1_InitStructure.USART_WordLength = USART_WordLength_8b;
USART1_InitStructure.USART_StopBits = USART_StopBits_1;
USART1_InitStructure.USART_Parity = USART_Parity_No;
USART1_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
USART1_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_Init(USART1, &USART1_InitStructure);


USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
```

그림 4 - USART Configuration

위의 코드는 USART 설정을 하는 코드이다. 전 차시의 실험에서 했던 것처럼 구조체를 통해 USART 설정을 한다. 이번 실험에서는 Baudrate를 9600으로 한다. 전 차시 실험처럼 10진수 값을 16진수 값으로 바꿀 필요는 없으므로 9600 그대로 준다.

다음은 NVIC 설정을 해야한다. NVIC는 Nested Vectored Interrupt Controller의 약자로 Interrupt들의 우선순위를 정할 수 있다. 그러므로 올바른 동작을 위해 Interrupt의 우선순위를 두어 설정해야 한다.

```
@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
  ============================================================================================================
```

그림 5 - Priority Configuration in NVIC

위의 사진은 NVIC의 Priority를 설정하기 위한 설명문이다. Group은 총 5개로 이루어져 있으며 Group별로 지정할 수 있는 bit의 수가 다르다. 예를 들어 1번 Group은 1개의 bit와 3개의 bits를 이용해 각각 Preemption Priority와 Sub Priority를 지정할 수 있다. 우선순위는 작은 수일수록 먼저이며 Preemption Priority를 먼저 비교한다.

```
NVIC_InitTypeDef NVIC_InitStructure;

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

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

// Joystick Down   EXTI2_IRQHandler
NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // TODO
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

그림 6 - NVIC Configuration

위의 코드는 NVIC를 구조체를 이용해 설정한 코드이다. 이번 실험에서 사용될 동작들은 동시에 일어날 시나리오를 가정하기 않았기 때문에 모두 같은 우선순위를 두었다.

또한 IRQ의 EXTI 채널 연결 할 때 port 0~4번은 개별로 따로 되어 있고, 5에서 9와 10에서 15 port는 묶여있다.

3) 동작 코드 작성

이제 동작을 위한 코드를 작성해야 한다. 늘 돌아가야 할 LED 흐름을 위해 전역 변수로 myflag를 설정한다. myflag가 0일 경우 1,2,3,4 순으로 LED가 바뀌며, myflag가 1일 경우 4,3,2,1 순으로 LED가 변경된다. 또한 UART 통신을 위해 print 전역 변수를 만든다.

```
while (1) {

  // TODO: implement
 if(myflag == 0) {
   index++;
   if(index >= 4) index = 0;
 }
 else if(myflag == 1) {
   index--;
   if(index < 0) index = 3;
 }
 if(print == 1){
    for(int i = 0; i < 8; ++i) {
      sendDataUART1((uint16_t)msg[i]);
    }
    print = 0;
 }

 if(index == 0) {
   *((volatile unsigned int *)0x40011410) |= 0x00000004;
 } else if(index == 1) {
   *((volatile unsigned int *)0x40011410) |= 0x00000008;
 } else if(index == 2) {
   *((volatile unsigned int *)0x40011410) |= 0x00000010;
 } else if(index == 3) {
   *((volatile unsigned int *)0x40011410) |= 0x00000080;
 }
 // Delay
 Delay();
 *((volatile unsigned int *)0x40011410) |= 0x009C0000;
}
```

그림 7 - while loop Action

while(true) 반복문 안에는 매 loop 마다해야 할 동작을 적는다.

다음은 Interrupt가 온 후 처리해야 할 동작의 설정이다. Interrupt Vector Table에는 각 Interrupt 별로 함수의 prototype이 정의되어 있다.

```
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
```

그림 8 - EXTI 0~4 Interrupt function prototype

```
DCD      EXTI9_5_IRQHandler          ; EXTI Line 9..5
```

그림 9 - EXTI 5~9 Interrupt function prototype

```
void EXTI2_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2) == Bit_RESET) {
            // TODO implement
            /* send UART 1 */
            myflag = 1;
        }
        EXTI_ClearITPendingBit(EXTI_Line2);
    }

}
```

그림 10 - Interrupt Action

Table을 참조해 함수의 이름을 찾고 다음과 같이 함수를 정의한다. 그림 10은 Joystick의 방향이 down으로 되어 PC2로부터 Interrupt가 걸렸고 이를 처리하기 위해 EXTI2_IRQHandler 함수를 호출한 것이다. joystick이 down이 되면 led는 4,3,2,1 순으로 불이 켜져야 하므로 전역변수 myflag의 값을 1로 하였다. 위로 하면 PC5가 Interrupt를 호출하며 이를 처리하기 위해서는 EXTI9_5_IRQHandler를 이용해야 한다. 여기서는 myflag 값을 반대로 0으로 하면 된다. “Team03\r\n”을 보내기 위한 함수는 User S1 버튼을 통해 작동하므로 EXTI15_10_IRQHandler 함수를 사용한다. 이 함수 내에서는 print 값을 1로 하여 while 문 내에 UART 통신 함수를 호출하도록 한다.

4. 실험 결과

Joystick의 위/아래 방향으로 기울여 Interrupt를 통해 led 발광의 순서를 바꿀 수 있었다. 또한 Putty를 통해 키보드의 u와 d를 눌러 led 발광 순서를 바꿀 수 있었다. 그러나 사진으로는 발광 순서를 담아내지 못하므로 참조된 영상으로 확인하기 바란다.

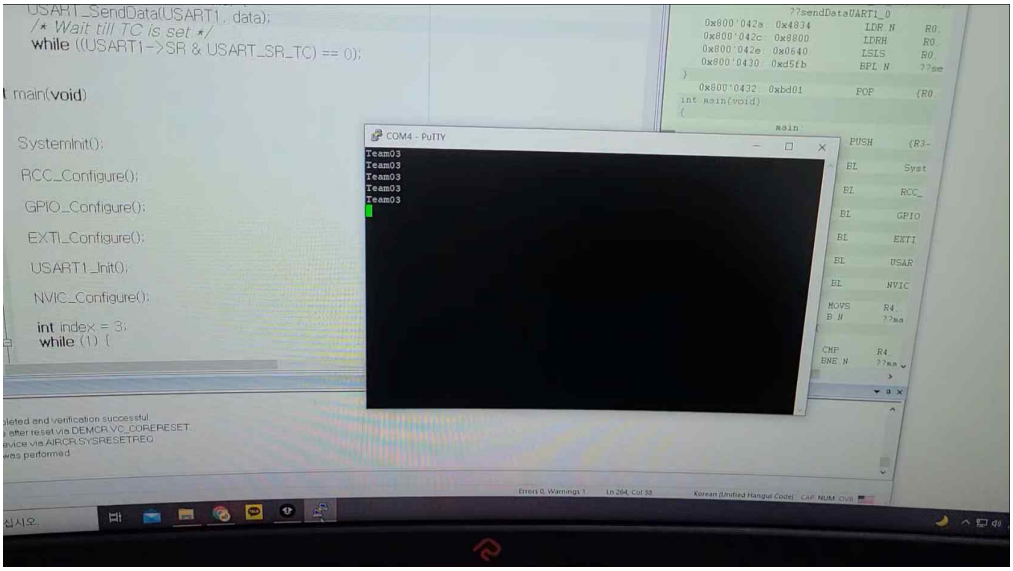

그림 11 - UART Communication

또한 User S1 버튼을 눌러 UART 통신을 통해 PC의 putty로 “Team03\r\n”을 전송할 수 있었다.