임베디드 시스템 설계 및 실험

7주차 결과 보고서

2023.10.18

3분반

8조

201924437 김윤하

202055604 조찬우

202155565 성가빈

202012142 이은진

● 목 표

1. Interrupt 방식을 활용한 GPIO 제어 및 UART 통신

2. 라이브러리 함수 사용법 숙지

세부 실험 내용

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

2. NVIC와 EXTI를 이용하여 GPIO에 인터럽트 핸들링 세팅

(ISR 동작은 최대한 빨리 끝나야 함)

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

(물결 속도는 delay를 이용하여 천천히 동작, ISR에서는 delay가 없어야 합니다)

3. KEY1 버튼 : A 동작, KEY2 버튼 : B 동작

(각 버튼을 눌렀을 때 위 동작이 지연시간 없이 바로 이루어져야 합니다)

4. PC의 Putty에서 a, b 문자 입력하여 보드 제어 (PC -> 보드 명령)

('a' : A 동작, 'b' : B 동작)

5. KEY3 버튼을 누를 경우 Putty로 "TEAMXX.₩r₩n" 출력

```
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);

    /* Button KEY1 KEY2 KEY3 port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, 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);

}
```

그림1. 포트 ENABLE 설정을 위한 RCC 함수

RCC_Configure 함수를 정의해서 실험을 위한 포트의 clock을 전부 ENABLE로 한다. 사용된 포트는 다음과 같다.

1. GPIOA, USART1: TX, RX, PC-보드 간 통신
2. GPIOB, GPIOC: 3개의 제어용 버튼
3. GPIOD: LED

```c
void GPIO_Configure(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

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

    /* Button KEY1 KEY2 KEY3 pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_13; //KEY1(PC4), KEY3(PC13)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10; //KEY2(PB10)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOB, &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_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

그림2. GPIO 설정

GPIO_InitTypeDef 구조체는 stm32f10x_gpio.h에 정의되어있다. 이를 이용해서 GPIO_InitStructure 구조체를 생성한다. 그 후 다음과 같은 작업을 실행한다.

1. 버튼에 사용할 GPIOC를 GPIO핀을 설정 해놓은 GPIOInitStructure로 세팅
2. 버튼에 사용할 GPIOB를 GPIO핀을 설정 해놓은 GPIOInitStructure로 세팅
3. LED에 사용할 GPIOD를 GPIO핀을 설정 해놓은 GPIOInitStructure로 세팅하되, Speed를 50MHz로 설정
4. TX, RX로 사용할 핀들도 다음과 같은 작업을 함.

```
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'

  /* Button KEY1 */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource4);
  EXTI_InitStructure.EXTI_Line = EXTI_Line4;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Button KEY2 */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource10);
  EXTI_InitStructure.EXTI_Line = EXTI_Line10;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Button KEY3 */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource13);
  EXTI_InitStructure.EXTI_Line = EXTI_Line13;
  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
}
```

그림3. EXTI 설정

이번 실험에서 외부(PC)입력으로부터 인터럽트를 발생시키고 받기 위해서 GPIO_EXTILineConfig함수를 사용해서 다음과 같은 방법을 3개의 버튼에 실행한다.

1. GPIO_EXTILineConfig함수를 버튼에 설정하여 버튼이 EXTI로 사용되게 한다.
2. 사전에 정의해놓은 EXTI_InitStructure의 configuration(Line, Mode, Trigger, LineCmd) Line으로 해당하는 EXTI Line에 매치, Mode로 Interrupt를 정의하고, Trigger를 이용해서 Falling 타입의 신호를 받게 하고, LineCmd로 해당 Line을 ENABLE 한다.
3. EXTI_InitStructure를 EXTI_Init를 사용해서 초기화한다.

```
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'
      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_Rx | USART_Mode_Tx;
      USART1_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
      USART_Init(USART1, &USART1_InitStructure);

   // TODO: Enable the USART1 RX interrupts using the function 'USART_ITConfig' and the argument value 'Receive Data regist
      USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

}
```

그림4. USART1 설정

USART에 대해서 설정을 하는데 방법은 그림 3의 방법을 USART에 적용시키되, 몇가지 configuration이 다른데 다음과 같다.

1. BaudRate는 보오율로써 Putty의 BaudRate와 같게 맞추면 된다.
2. WordLength는 nb(n비트)만큼의 공간을 부여한다.
3. Paritiy는 사용하지 않기 때문에 No로 설정
4. Mode는 우리가 사용한는 RX, TX를 부여한다.
5. HardwareFlowControl은 사용하지 않기 때문에 None을 부여한다.

마지막으로 ITConfig를 RX에 대해서 ENABLE 시켜놓아서 인터럽트를 받아들일 수 있게 한다.

```
void NVIC_Configure(void) {

  NVIC_InitTypeDef NVIC_InitStructure;

  // TODO: fill the arg you want
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

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

  // Button KEY1
  NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0xF;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  // Button KEY2
  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0xF;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  // Button KEY3
  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0xF;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
  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 = 0xF;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

그림5. NVIC 설정

NVIC_PriorityGroup은 아무거나 해도 상관없기 때문에 임의로 4번 그룹을 사용하였다. 버튼 1에는 EXTI4_IRQn; 버튼2,3 에는 EXTI15_10_IRQn; UART에는 USART1_IRQn을 설정해주고 전부 EnableIRQ을 설정해준다. 각각의 핸들러가

버튼에 대해서 매핑이 되는 구조이다.

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority최우선 순위

NVIC_InitStructure.NVIC_IRQChannelSubPriority 차선 순위를 정의한다.

NVIC_InitStructure.NVIC_IRQChannelCmd를 ENABLE로 설정해서 NVIC Channel의 사용여부를 입력 할 수 있게한다.

```c
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') {
      dir = 0;
    }
    else if (word == 'b') {
      dir = 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_Line10) != RESET) {
    if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_10) == Bit_RESET) { //KEY 2
          dir = 1;
    }
    EXTI_ClearITPendingBit(EXTI_Line10);
  }
  if (EXTI_GetITStatus(EXTI_Line13) != RESET) {
    if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13) == Bit_RESET) { //KEY 3
          char teamName[] = "Team08₩r₩n";
          for (int i=0; i<8; i++) {
            sendDataUART1(teamName[i]);
          }
    }
    EXTI_ClearITPendingBit(EXTI_Line13);
  }
}

 // TODO: Create Joystick interrupt handler functions
 // TODO: KEY 1 interrupt handler functions
 void EXTI4_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line4) != RESET) {
      if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_4) == Bit_RESET) { //KEY 1
            dir = 0;
      }
      EXTI_ClearITPendingBit(EXTI_Line4);
    }
 }
```

그림6. USART1_IRQHandler, EXTI15_10_IRQHandler, EXTI4_IRQHandler

각각의 Handler를 정의하여 문자를 입력받거나, 버튼을 눌렀을 때 어떤 반응을 할지 설정한다.

USART1_IRQ의 경우 word라는 전역 변수에 'a'가 입력되면 0, 'b'가 입력되면 1이 설정되게 한다.

EXTI15_10_IRQHandler 의 경우 버튼2를 눌렀을 때 dir라는 전역변수를 1로 바꾸어서 LED방향을 1->2->3->4로 하고, 버튼3을 눌렀을 때는 Team08₩r₩n을 출력하게 한다. 어차피 두 버튼은 서로 간섭하지 않기 때문에 같은 Handler에 넣어도 문제가 없다.

EXTI4_IRQHandler는 dir를 0으로 바꾸어서 4->3->2->1 순으로 LED방향을 바꾼다.

```
void Delay(void) {
    int i;

    for (i = 0; i < 2000000; i++) {}
}

void sendDataUART1(uint16_t data) {
    /* Wait till TC is set */
    while ((USART1->SR & USART_SR_TC) == 0);
    USART_SendData(USART1, data);
}

void LED_On() {
  GPIO_SetBits(GPIOD, GPIO_Pin_2);
  GPIO_SetBits(GPIOD, GPIO_Pin_3);
  GPIO_SetBits(GPIOD, GPIO_Pin_4);
  GPIO_SetBits(GPIOD, GPIO_Pin_7);

  switch(ledNumber) {
  case 0:
    GPIO_ResetBits(GPIOD, GPIO_Pin_2);
    break;
  case 1:
    GPIO_ResetBits(GPIOD, GPIO_Pin_3);
    break;
  case 2:
    GPIO_ResetBits(GPIOD, GPIO_Pin_4);
    break;
  case 3:
    GPIO_ResetBits(GPIOD, GPIO_Pin_7);
    break;
  }
}
```

그림7. LED 제어함수

Delay는 LED 점등의 간격을 정해주고, sendDataUART1은 인자로 받는 uint16_t 형식의 데이터를 EXTI15_10_IRQHandler에게 전해주는 역할을 한다.

LED_On을 통해서 GPIO_SetBits를 4개 만들어 놓고 ledNumber라는 전역변수를 받아서 점등할 LED의 GPIO를 Reset해준다.

```c
int main(void)
{

   SystemInit();

   RCC_Configure();

   GPIO_Configure();

   EXTI_Configure();

   USART1_Init();

   NVIC_Configure();

   while (1) {
      // TODO: implement
    if (dir == 0) {
      ledNumber = (ledNumber + 1) % 4;
      LED_On();
    }
    else {
      ledNumber = (ledNumber + 3) % 4;
      LED_On();
    }
    // Delay
    Delay();
   }
   return 0;
}
```

그림10. 메인 함수

SystemInit()부터
NVIC_Configure까지 모든 핀들의 설정을 끝마치고

while문으로
dir이 0일 때 ledNumber에 1씩 더해줘서 오른쪽으로
dir이 1일 때는 ledNumber에 -1을 빼면 음수값을 나누개 해버리므로 +3을 하였다. 그렇게 하면 3 2 1 4 3 2 1 4 순으로 진행하게 된다.

결론 및 느낀 점

실험을 하면서 여러가지 문제가 있었는데 첫번째로 EXTI를 EXIT로 헷갈려서 프로그램의 실행이 안되었던 점이 있다. 함수의 이름을 잘못 적는 것은 프로그램이 컴파일 하는 것 자체에 에러를 띄우지 않기 때문에 잡아내기 매우 힘들었다. 그리고 delay()함수의 위치를 처음에는 if문보다 선행해서 실행시켰는데, 이는 버튼을 눌렀을 때 LED 방향의 전환 속도가 1박자 느려졌던 상황을 초래했다. 또한 그 전에는 프로그램 자체적으로 방대한 양의 #define을 통한 복잡한 숫자 연산을 해서 매우 힘들었지만, 이번에는 구조체를 이용한 프로그래밍을 통해서 훨씬 더 사용자 친화적인 설계를 할 수 있었다. 또한 NVIC를 이용해서 인터럽트의 우선순위를 조정하고, 서로 간섭을 일으키지 않는 버튼 2개를 하나의 Handler로 넣는 등 좋은 경험이 되었다.

결과사진