임베디드시스템설계및실험

(CB3400001-004)

7주차 실험 결과 보고서

9조

202155536 김주송

201824404 강선호

201924484 박형주

202013119 손봉국

1. 실험 목표

① Interrupt 방식을 활용한 GPIO 제어 및 UART 통신

② 라이브러리 함수 사용법 숙지

2. 실험 내용

- Interrupt

  CPU가 특정 이벤트 발생시 현재 작업을 멈추고 인터럽트 서비스 루틴을 수행 후 다시 이전 작업으로 돌아가는 방식

- EXIT (External Interrupt)

  외부에서 신호가 입력될 경우 Device에 Event나 Interrupt가 발생되는 기능으로 각 Port의 n번 Pin의 EXTIn에 연결한다. 모든 GPIO핀들은 EXTI Line을 통해 연결되어 있다. EXTI 선언했을 경우, 반드시 Handler 구현이 필요하다.

- NVIC (Nested Vectored Interrupt Controller)

  인터럽트 처리 중 또다른 인터럽트 발생시 우선순위를 사용하여 우선순위가 높은 인터럽트부터 처리 후 다른 인터럽트를 처리한다. ARM 보드에서 인터럽트 사용시 NVIC를 통해 우선순위를 결정하고 값이 작을수록 우선순위가 높다.

3. 실험 과정

(1) APB2 peripheral clock 인가

```
void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState)
{
  /* Check the parameters */
  //assert_param(IS_RCC_APB2_PERIPH(RCC_APB2Periph));
  //assert_param(IS_FUNCTIONAL_STATE(NewState));
  if (NewState != DISABLE)
  {
    RCC->APB2ENR |= RCC_APB2Periph;
  }
  else
  {
    RCC->APB2ENR &= ~RCC_APB2Periph;
  }
}

/**
  * @brief  Enables or disables the Low Speed APB (APB1) peripheral clock.
  * @param  RCC_APB1Periph: specifies the APB1 peripheral to gates its clock.
  *   This parameter can be any combination of the following values:
  *     @arg RCC_APB1Periph_TIM2, RCC_APB1Periph_TIM3, RCC_APB1Periph_TIM4,
  *          RCC_APB1Periph_TIM5, RCC_APB1Periph_TIM6, RCC_APB1Periph_TIM7,
  *          RCC_APB1Periph_WWDG, RCC_APB1Periph_SPI2, RCC_APB1Periph_SPI3,
  *          RCC_APB1Periph_USART2, RCC_APB1Periph_USART3, RCC_APB1Periph_USART4,
  *          RCC_APB1Periph_USART5, RCC_APB1Periph_I2C1, RCC_APB1Periph_I2C2,
  *          RCC_APB1Periph_USB, RCC_APB1Periph_CAN1, RCC_APB1Periph_BKP,
  *          RCC_APB1Periph_PWR, RCC_APB1Periph_DAC, RCC_APB1Periph_CEC,
  *          RCC_APB1Periph_TIM12, RCC_APB1Periph_TIM13, RCC_APB1Periph_TIM14
  * @param  NewState: new state of the specified peripheral clock.
  *   This parameter can be: ENABLE or DISABLE.
  * @retval None
  */
```

Stm32f10x_rcc.c 파일 속 APB2PeriphClockCmd를 참고하여 버튼, LED, USART, IO에 대한 clock 을 enable 한다.

```
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 S1, S2, S3 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);
}
```

(2) GPIO 핀 초기화

```
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 */

    /* PB10*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    /*PC4,13*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &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_Mode_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

GPIO 핀을 초기화하는 작업을 수행한다. 이번 실험에서 버튼 KEY1(PC4), KEY2(PB10), KEY3(PC13)과 4개의 LED(PD2, PD3, PD4, PD7), UART TX(PA9)/RX(PA10)를 사용하므로 각 핀을 포트별로 나누어 차례대로 'GPIO_InitStructure.GPIO_Pin'에 각각의 GPIO 핀을 설정한다. Input 또는 Output을 설정해 'GPIO_InitStructure.GPIO_Mode'를 입력한다. 'GPIO_InitStructure.GPIO_Speed'에 출력 speed를 함께 설정한 후 'GPIO_Init()'를 사용하여 해당하는 포트의 핀에 접근해 초기화를 진행한다.

(3) GPIO 핀 지정 및 EXTI Line 설정

```
void EXTI_Configure(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;

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

    /* Button 1(PC4) is pressed */
    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 2 */
    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 3 */
    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
}
```

Interrupt를 발생시키기 위해 입력이 되는 버튼 1, 2, 3 핀이 EXTI Line으로 사용되도록 GPIO_EXITLineConfig()'를 이용하여 설정할 포트와 핀을 지정한다. EXIT Line을 매치하고 Mode에 Interrupt를 설정한다. Trigger에 입력 받을 타입을 설정하고 Line을 Enable한다. 'EXIT_Init()'를 사용해 위의 설정을 적용한다.

(4) USART 초기화

```
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= 28800;
    USART1_InitStructure.USART_Mode= USART_Mode_Rx|USART_Mode_Tx ;
    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;
    USART_Init(USART1,&USART1_InitStructure);


    // 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);
}
```

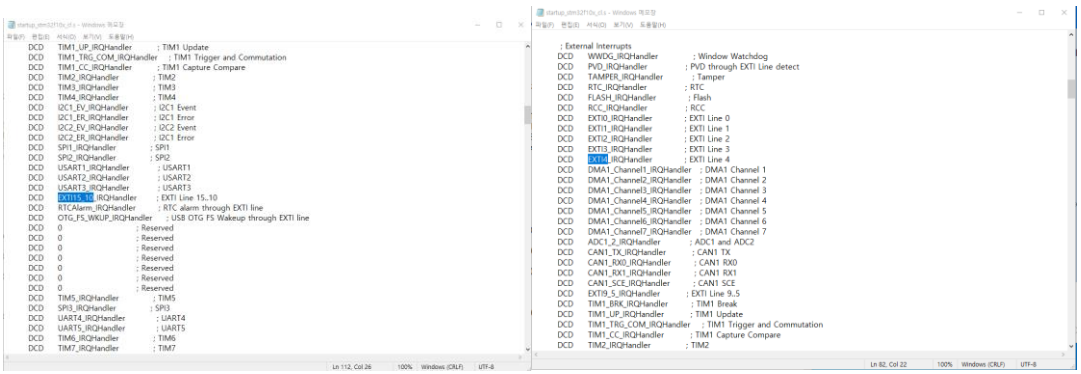
먼저 'USART_Cmd'를 사용해 USART1를 enable한다. Baud rate는 지난 실험에서 사용한 값과 동일한 28800을 대입하고 "TEAM09₩r₩n"을 출력할 수 있도록 Word length는 8bit로 부여한다. USART에서 데이터 흐름을 제어하지 않기 때문에 HardwareFlowControl_None을 부여한다. Parity 비트도 설정하지 않으므로 No로 설정하고, RX/TX를 사용하므로 OR을 이용해 RX/TX mode로 설정해준다. 마지막으로 'USART_Init()'을 이용해 USART를 초기화한다.

'USART_ITConfig()'를 통해 RX 인터럽트로 enable하고 매개변수로 'Receive Data register not empty interrupt'(USART_IT_RXNE)를 사용한다.

(5) NVIC 설정

```
void NVIC_Configure(void) {

    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'

    // Button S1
    NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Button S2, S3
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // 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 = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

Priority Group을 0으로 설정하고 'NVIC_initTypeDef'와 'NVIC_Init()'를 이용하여 NVIC를 초기화하고 'NVIC_EnableIRT()'로 NVIC 인터럽트 컨트롤러의 인터럽트를 활성화하였다. 미리 정의되어 있어 'stm32f10x_cl.s'를 참고하여 작성하였다.



(6) A 동작, B 동작 설정

```
int dx = 1;
```

```
void FunctionA()
{
  dx=1;
}

void FunctionB()
{
  dx=-1;
}
```

A 동작을 설정하기 위한 함수인 FunctionA()와 B 동작을 설정하기 위한 함수 FunctionB()를 설정한다. FuctionA()가 실행되면 1->2->3->4로, FunctionB()가 실행되면 4->3->2->1로 실행한다. Interrupt를 통해 버튼이나 문자를 입력하면 dx 변수 값을 1 또는 -1로 변경하며 그 값을 index 변수에 더하여 방향을 변경하는 방식을 이용한다.

(7) USART Handler 설정

```
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') FunctionA();
        else if (word =='b') FunctionB();

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

입력한 문자에 대해 동작을 정의한다. 'a'를 입력했을 경우, A 동작(버튼 KEY1을 눌렀을 때의 동작 - FunctionA())을 수행하도록 설정하고, 'b'를 입력했을 경우는 B 동작(버튼 KEY2를 눌렀을 때의 동작 – FunctionB())을 수행하도록 설정한다. Interrupt 처리 확인을 위해 'USART_ClearITPendingBit()'를 이용해 해당 interrupt pending bit를 clear 한다.

(8) Button Handler 설정

```
void EXTI4_IRQHandler(void) { // when the button is pressed

    if (EXTI_GetITStatus(EXTI_Line4) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_4) == Bit_RESET) {
            // TODO implement
            FunctionA();
        }
        EXTI_ClearITPendingBit(EXTI_Line4);
    }

}
```

```
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) {
            // TODO implement
            FunctionB();
        }
        EXTI_ClearITPendingBit(EXTI_Line10);
    }
    if (EXTI_GetITStatus(EXTI_Line13) != RESET) {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13) == Bit_RESET) {
            // TODO implement
            char str[] = "TEAM09\r\n";
            for(int i = 0; str[i] != '\0'; i++){
                sendDataUART1(str[i]);
            }
        }
        EXTI_ClearITPendingBit(EXTI_Line13);
    }
}
```

버튼 bit를 초기화한 후, KEY1(PC4) 버튼을 누르면 A가 동작하고 KEY2(PB10) 버튼을 누르면 B가 동작할 수 있도록 FunctionA()와 FunctionB()를 수행한다. KEY3(PC13) 버튼을 누를 때 Putty로 "TEAM09₩r₩n"을 출력하도록 설정한다. Interrupt 처리 확인을 위해 'USART_ClearITPendingBit()'를 이용해 해당 interrupt pending bit를 clear 한다.

(9) While 문 동작 구현

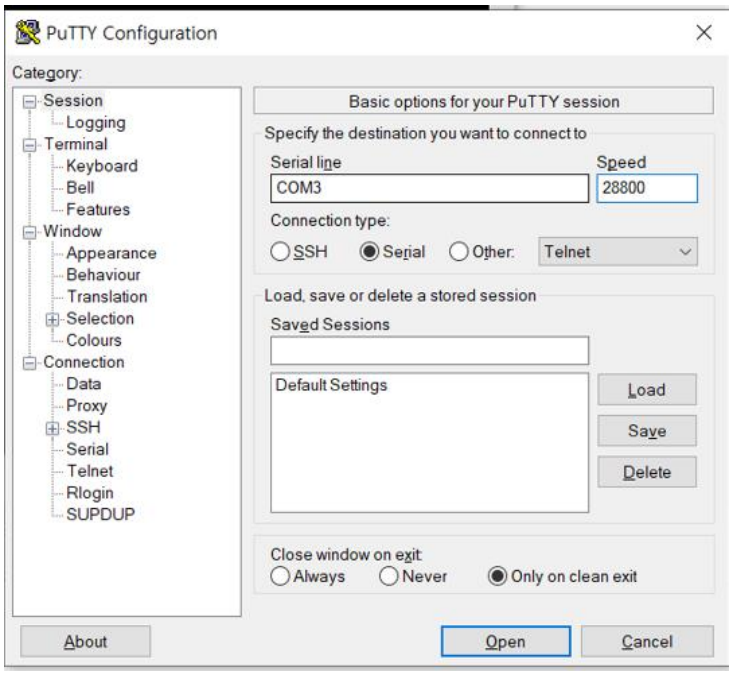
```
uint16_t seq[4] = {GPIO_Pin_2, GPIO_Pin_3, GPIO_Pin_4, GPIO_Pin_7};
```

```
int main(void)
{

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

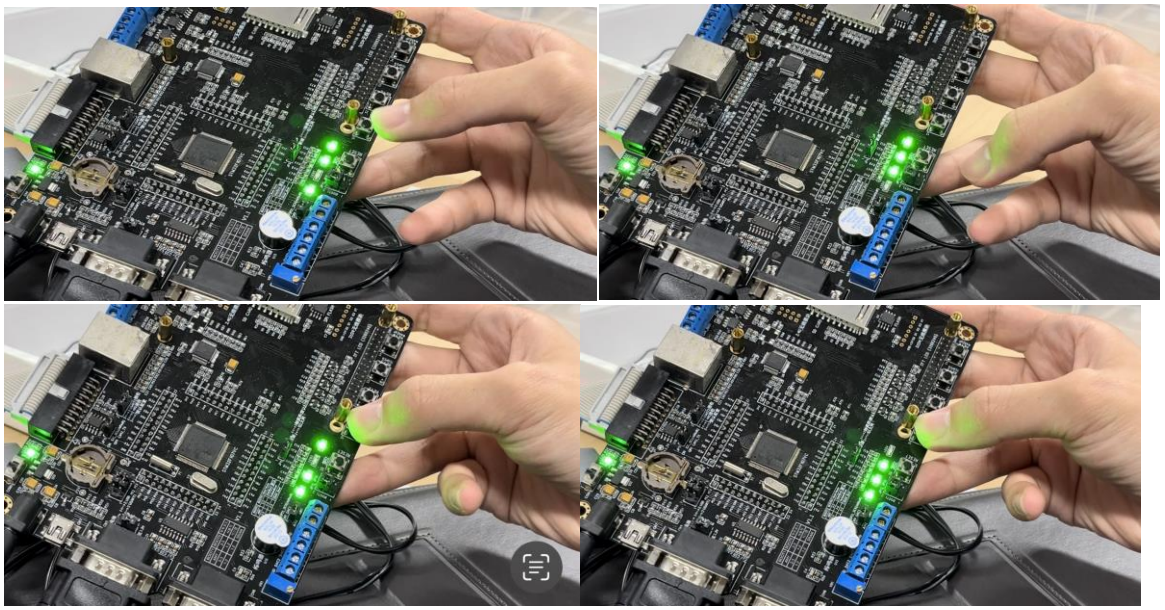
```
    int index = 0;
    while (1) {
        // TODO: implement
        GPIO_WriteBit(GPIOD, seq[index], Bit_SET);
        Delay();
        GPIO_WriteBit(GPIOD, seq[index], Bit_RESET);
        index += dx;
        if (index < 0) index = 3;
        index %= 4;

        // Delay
        Delay();
    }
    return 0;
}
```

동작 반복을 위해 while문을 사용하여 그 안에 동작을 구현한다. 핀 출력 번호를 위한 주솟값 배열 seq를 이용하여 출력을 설정한다. LED 출력 순서를 판단하기 위한 변수 index를 이용하여 다음 LED의 위치를 변경해준다.

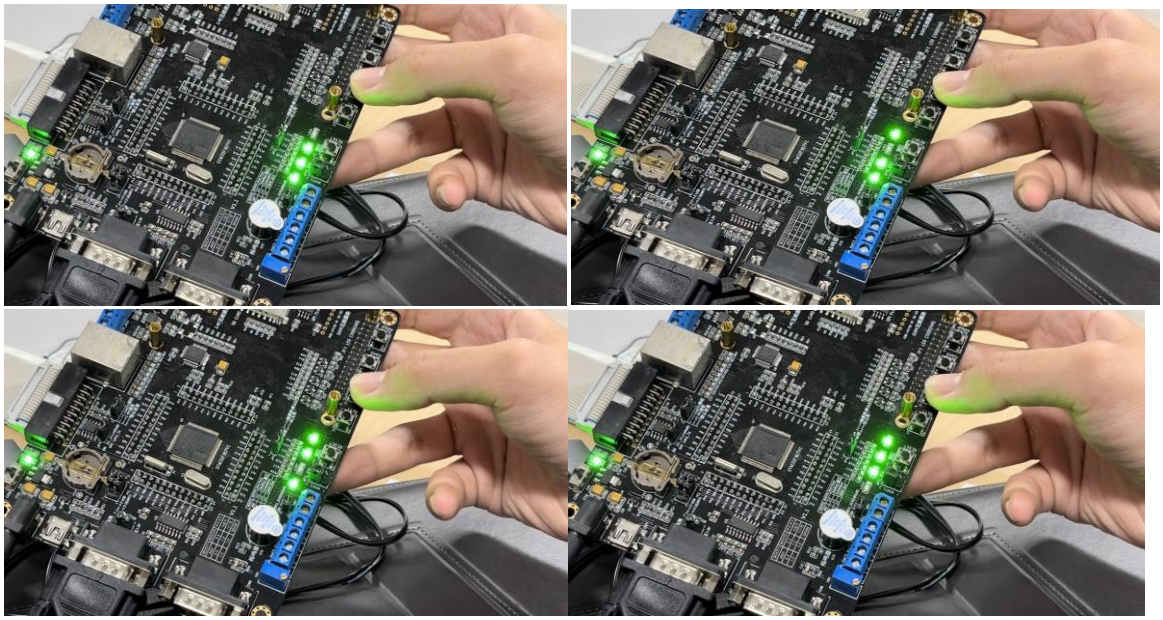
4. 실험 결과



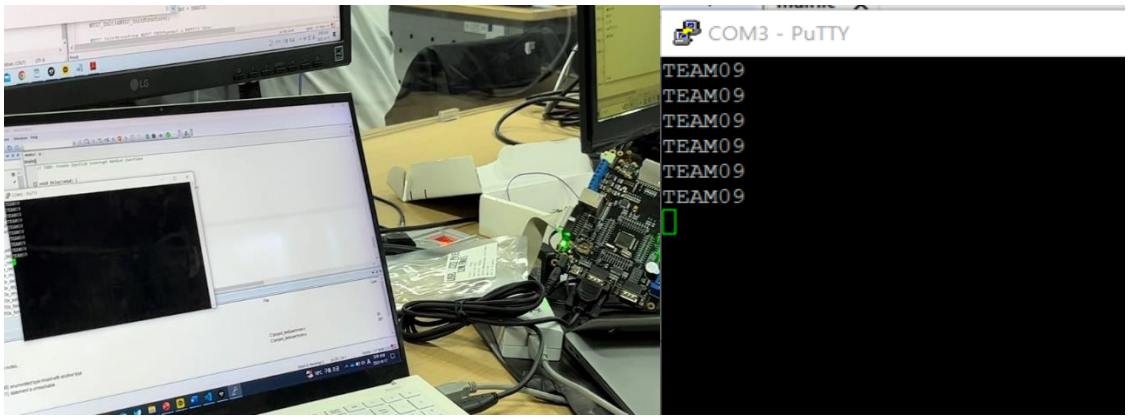
Serial line과 Speed를 알맞게 설정해준 후, 실행한다.



KEY1 버튼을 눌렀을 때, A 동작 (1->2->3->4 순)이 진행되는 것을 확인할 수 있었다.



KEY2 버튼을 눌렀을 때, B 동작 (4->3->2->1 순)이 진행되는 것을 확인할 수 있었다.



PC의 Putty에서 a를 입력했을 때, A 동작 (1->2->3->4 순)이 진행되고 b를 입력했을 경우, B 동작 (4->3->2->1 순)이 진행된다. KEY3 버튼을 누를 경우 Putty에 "TEAM09₩r₩n"이 출력되는 것을 확인할 수 있다.