임베디드시스템 설계 및 실험

7주차 시험결과보고서

2023.10.17

2조

202155592 이지수

202155553 박은재

201924515 유승훈

201524410 고상현

201924402 강민수

1. 이론적 배경

가. Interrupt 방식을 활용한 GPIO 제어

인터럽트(Interrupt)는 CPU에게 특정 이벤트나 조건이 발생했음을 알리는 신호로 인터럽트가 발생시 CPU는 현재 실행 중인 작업을 중단하고 인터럽트 서비스 루틴(ISR)을 실행하여 해당 이벤트를 처리한다. 입력 모드에서 GPIO핀은 외부 신호의 변화를 감지하여 인터럽트를 발생시킬 수 있다. 이를 통해 MCU는 실시간으로 외부 신호의 변화를 감지하고 적절한 동작을 수행할 수 있다. 특정 조건만 체크하는 폴링(Polling) 방식과 비교하여 이벤트가 발생할 때만 동작하는 인터럽트 방식은 MCU의 효율성을 크게 향상시키고 리소스를 절약할 수 있다.

나. UART 통신

UART는 마이크로컨트롤러와 다른 장치 간의 비동기 시리얼 통신을 위한 장치이다. 데이터를 시리얼로 전송하며 시작과 스톱 비트를 포함한 프레임을 형성하고 수신 시 시작 비트를 감지하고 설정된 비트 수를 기반으로 데이터를 수집한다. 또한 데이터 전송이나 수신 시 인터럽트를 발생시키면 데이터 도착을 실시간으로 감지할 수 있고 이를 통해 데이터 처리가 가능하다.

2. 실험목표

가. Interrupt 방식을 활용한 GPIO 제어 및 UART 통신

나. 라이브러리 함수 사용법 숙지

3. 실험 세부 목표

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

나. 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

(1->2->3->4로 LED가 켜진 뒤, 1->2->3->4 순서로 LED 소등)

B: LED 물결 방향 변경 - 4->3->2->1

(4->3->2->1로 LED가 켜진 뒤, 4->3->2->1 순서로 LED 소등)

주의사항: 물결 속도는 delay를 이용하여 천천히 동작, ISR에서는 delay가 없어야 함.

(PD2 = 1, PD3 = 2, PD4 =3, PD7 = 4)

다. KEY1(PC4) 버튼: A 동작, KEY2(PB10) 버튼: B 동작 (지연시간 없이 동작

라. PC의 Putty에서 a, b 문자 입력하여 보드 제어 (PC -> 보드 명령, 'a': A 동작, 'b': B 동작)

마. KEY3(PC13) 버튼을 누를 경우 Putty로 "TEAM02.₩r₩n"를 출력

4. 실험 과정

가. RCC_Configure 함수

RCC_Configure에서 'RCC_APB2PeriphClockCmd' 함수를 활용하여 실험에 사용되는 UART TX/RX port clock, Button port clock, LED port clock, USART1 clock, Alternate Function IO clock를 모두 활성화시켜준다.

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

    /* UART TX/RX port clock enable */ //완료
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    /* Button S1, S2, S3 port clock enable */
    //GPIO C,B에 버튼 Key1, Key2, Key3 존재 //완료
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    /* LED port clock enable */ //완료
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    /* USART1 clock enable */ //완료
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    /* Alternate Function IO clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

그림 1. RCC_Configure 함수

나. GPIO_Configure 함수

GPIO_Configure에서는 GPIO_InitTypeDef 구조체를 사용하여 GPIO핀 각 핀의 설정 값을 지정해준다. GPIOB와 GPIOC에 있는 S1, S2, S3 버튼(Key1, Key2, Key3)은 내부 풀 다운 저항을 사용하는 입력 모드로 설정하고 GPIOD에 있는 PD2, PD3, PD4, PD7 핀으로 연결된 LED는 50MHz의 속도로 푸시-풀 출력 모드로 설정한다. 또한 USART를 활용하기 때문에 TX 핀은 대체 함수 출력 모드로 설정하고 RX 핀은 내부 풀 다운 저항을 사용하는 입력 모드로 구성한다.

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

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

    //Key1
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;// INPUT PULL-DOWN
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Key2
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;// INPUT PULL-DOWN
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    //Key3
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;// INPUT PULL-DOWN
    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 */ //TODO //완료
    //TX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // USART TX - alternate function
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    //RX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; // USART RX - PULL-DOWN
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

그림 2. GPIO_Configure 함수

다. EXTI_Configure 함수

EXTI_Configure에서 S1, S2, S3 버튼을 통한 외부 입력이 인터럽트를 발생시키도록 설정한다. 이를 위해 STM32에서 제공하는 GPIO_EXTILineConfig 함수를 사용하여 해당 핀들을 EXTI Line으로 지정한다. 또한 EXTI_InitTypeDef 구조체와 EXTI_Init 함수를 활용하여 인터럽트 관련 설정을 초기화한다. 먼저 GPIO_EXTILineConfig 함수를 사용하여 각 버튼에 대응하는 GPIO 포트와 핀을 EXTI Line으로 지정한다. 이후 EXTI_InitStructure 구조체를 통해 해당 EXTI Line, 인터럽트 모드, 트리거 타입 등을 설정한다. 특히 EXTI_InitStructure.EXTI_Mode에는 인터럽트 모드를 설정해주고 EXTI_InitStructure.EXTI_Trigger에는 EXTI_Trigger_Falling를 감지하는 설정을 지정한다. 설정이 완료되면 EXTI_InitStructure.EXTI_LineCmd를 사용하여 해당 EXTI Line을 활성화한다. 최종적으로 EXTI_Init 함수를 호출하여 지정된 설정을 적용시킨다.

```
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 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 */ //TODO //완료
    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 */ //TODO //완료
    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_Configure 함수

라. USART1_Init 함수

USART1_Init는 USART1을 초기화해주는 역할을 한다. 먼저 USART_Cmd 함수를 활용하여USART1을 활성화해준다. 그 다음 USART_InitTypeDef 구조체를 활용하여 USART 설정을 진행한다. 'TEAM02₩r₩n' 문자열의 정확한 전송을 보장하기 위해 28800 BaudRate의 통신 속도와 8비트의 데이터 길이를 설정해준다. Parity 비트는 사용하지 않기 때문에 USART_Parity_No로 데이터 흐름 제어도 사용하지 않기 때문에 USART_HardwareFlowControl_None으로 지정된다. USART의 수신(Rx) 및 송신(Tx) 기능을 활성화하고 모든 설정이 끝난 후에 USART_Init 함수를 호출하여 이를 USART1에 적용한다. 마지막으로 USART_ITConfig 함수로 수신 인터럽트를 활성화시킨다.

```
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_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART1_InitStructure.USART_Parity = USART_Parity_No;
    USART1_InitStructure.USART_StopBits = USART_StopBits_1;
    USART1_InitStructure.USART_WordLength = USART_WordLength_8b;
    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); // USART RX   Interrupt
}
```

그림4. USART1_Init 함수

마. NVIC_Configure 함수

NVIC_Configure는 STM32의 Nested Vectored Interrupt Controller (NVIC) 설정을 수행한다. NVIC_InitTypeDef 구조체를 사용하여 초기에 인터럽트 설정을 준비하며 NVIC_PriorityGroupConfig 함수를 통해 인터럽트 우선 순위 그룹을 정의한다. S1 버튼은 EXTI4 인터럽트 라인 S2와 S3 버튼은 EXTI15_10 인터럽트 라인에 연결되어 각각의 우선 순위와 하위 우선 순위 설정 후 인터럽트 활성화한다. 또한 UART1 통신은 데이터 전송 및 수신 시 인터럽트가 발생하므로 이에 대한 우선 순위와 하위 우선 순위도 설정하고 활성화한다. 이를 통해 인터럽트 소스의 우선 순위 설정과 해당 인터럽트의 활성화를 진행할 수 있다.

```
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'

    // Button S1: PC4
    NVIC_EnableIRQ(EXTI4_IRQn); //key1
    NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
    // Button S2, S3 : B10,C13
    NVIC_EnableIRQ(EXTI15_10_IRQn); //key2,Key3
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01; // 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 = 0x01; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

그림5. NVIC_Configure 함수

바. USART1_IRQHandler 함수

USART1_IRQHandler는 USART1에서 발생하는 인터럽트를 처리한다. 주로 데이터 수신 시 해당 인터럽트가 발생하며 이를 처리하기 위해 USART_GetITStatus 함수를 사용하여 USART_IT_RXNE 플래그가 설정되었는지 확인한다. 설정된 경우 USART_ReceiveData 함수를 사용하여 수신된 데이터를 읽는다. 이번 실험에서는 'a'와 'b' 문자를 통해 특정 동작을 제어한다. 'a' 문자를 수신하면 A동작인 1->2->3->4 순으로 wave가 설정되도록 up_to_down 변수를 0으로 'b' 문자를 수신하면 B동작인 4->3->2->1 순으로 wave가 설정되도록 up_to_down 변수를 1로 설정한다. 처리가 완료된 후 USART_ClearITPendingBit 함수를 사용하여 USART_IT_RXNE 플래그를 클리어해준다.

```
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_to_down = 0; //1->2->3->4
        }else if(word == 'b'){
            up_to_down = 1;
        }

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

그림6. USART1_IRQHandler 함수

사. EXTI4_IRQHandler 함수, EXTI15_10_IRQHandler 함수

EXTI4_IRQHandler는 GPIOC의 4번 핀인 key1 버튼과 연결된 EXTI_Line4 인터럽트 발생 시 호출된다. 함수 실행 시 먼저 EXTI_GetITStatus를 통해 해당 인터럽트의 발생 여부를 확인하며 GPIO_ReadInputDataBit를 이용해 버튼 상태를 파악한다. 버튼이 눌렸을 때 up_to_down 변수는 0으로 설정되어 A동작인 1->2->3->4 순으로 wave가 설정되도록 한다. 함수의 마지막 부분에서는 EXTI_ClearITPendingBit를 사용하여 인터럽트 플래그를 초기화해 다음 인터럽트 발생을 대비한다.

EXTI15_10_IRQHandler는 GPIOB의 10번 핀과 GPIOC의 13번 핀 즉 key2와 key3 버튼과 연결된 인터럽트 발생 시 호출된다. 함수 내에서는 두 가지 주요 조건을 검사한다. 첫 번째 조건은 key2 버튼의 상태를 확인하고 눌렸을 경우 up_to_down 변수를 1로 설정하여 B동작인 4->3->2->1 순으로 wave를 표현한다. 두 번째 조건은 key3 버튼 상태를 확인하며 눌렸을 경우 sendDataUART1 함수를 통해 "TEAM02₩r₩n" 문자열을 UART1로 전송한다. 마찬가지로 각 조건문 후에는 인터럽트 플래그를 초기화하여 다음 인터럽트 발생을 대비한다.

```
void EXTI4_IRQHandler(void) { // when the button is pressed
    if (EXTI_GetITStatus(EXTI_Line4) != RESET) { //key1 눌렸을때 A동작 수행
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_4) == Bit_RESET) {
            // TODO implement
            up_to_down = 0; //1->2->3>4
        }
        EXTI_ClearITPendingBit(EXTI_Line4);
    }
}

void EXTI15_10_IRQHandler(void) { // when the button is pressed
    char team[] = "TEAM02\r\n";
    if (EXTI_GetITStatus(EXTI_Line10) != RESET) { //key2 눌렸을때 B동작 수행
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_10) == Bit_RESET) {
            // TODO implement
            up_to_down = 1; //4->3->2->1
        }
        EXTI_ClearITPendingBit(EXTI_Line10);
    }
    if (EXTI_GetITStatus(EXTI_Line13) != RESET) { //key3 눌렸을때
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13) == Bit_RESET) {
            // TODO implement
            for(int i=0; i<8; ++i){
                sendDataUART1(team[i]);
            }
        }
        EXTI_ClearITPendingBit(EXTI_Line13);
    }
}
```

그림7. EXTI4_IRQHandler 함수, EXTI15_10_IRQHandler, 함수

아. Delay 함수, sendDataUART1 함수, turnOnLed 함수, main 함수

main에서는 시스템 초기화, RCC, GPIO, EXTI, USART1 및 NVIC의 설정 함수들을 순차적으로 호출하여 마이크로컨트롤러의 환경을 준비한다. 메인 루프에서는 먼저 모든 LED를 꺼주고 turnOnLed 함수를 통해 현재 상태에 따라 특정 LED를 켜고 끈다. up_to_down 변수에 따라 LED의 점등 순서가 변경된다. up_to_down이 0이면 A동작을 하고 1이면 B동작을 한다. 각 LED 점등 사이에는 Delay 함수를 통해 일정 시간 동안 대기하여 LED가 각 동작에 맞게 wave한다.

```
// TODO: Create Joystick interrupt handler functions
void Delay(void) {
    for (int i = 0; i < 2000000; i++) {}
}

void sendDataUART1(uint16_t data) {
    while ((USART1->SR & USART_SR_TC) == 0);
    USART_SendData(USART1, data);
}

void turnOnLed(int index){
    uint16_t pinList[] = {GPIO_Pin_2, GPIO_Pin_3, GPIO_Pin_4, GPIO_Pin_7};
    GPIO_SetBits(GPIOD, pinList[index]);
    Delay();
    GPIO_ResetBits(GPIOD, pinList[index]);
}

int main(void)
{
    SystemInit();
    RCC_Configure();
    GPIO_Configure();
    EXTI_Configure();
    USART1_Init();
    NVIC_Configure();

    int state = 0;

    while (1) {
        // TODO: implement
        GPIO_ResetBits(GPIOD, GPIO_Pin_2);
        GPIO_ResetBits(GPIOD, GPIO_Pin_3);
        GPIO_ResetBits(GPIOD, GPIO_Pin_4);
        GPIO_ResetBits(GPIOD, GPIO_Pin_7);
        turnOnLed(state);

        if(up_to_down == 0) state = (state + 1) % 4;
        else if(state == 0 && up_to_down) state = 3;
        else state = (state - 1) % 4;
        Delay();// Delay
    }
    return 0;
}
```

그림8. Delay 함수, sendDataUART1 함수, turnOnLed 함수, main 함수

5. 실험 결과

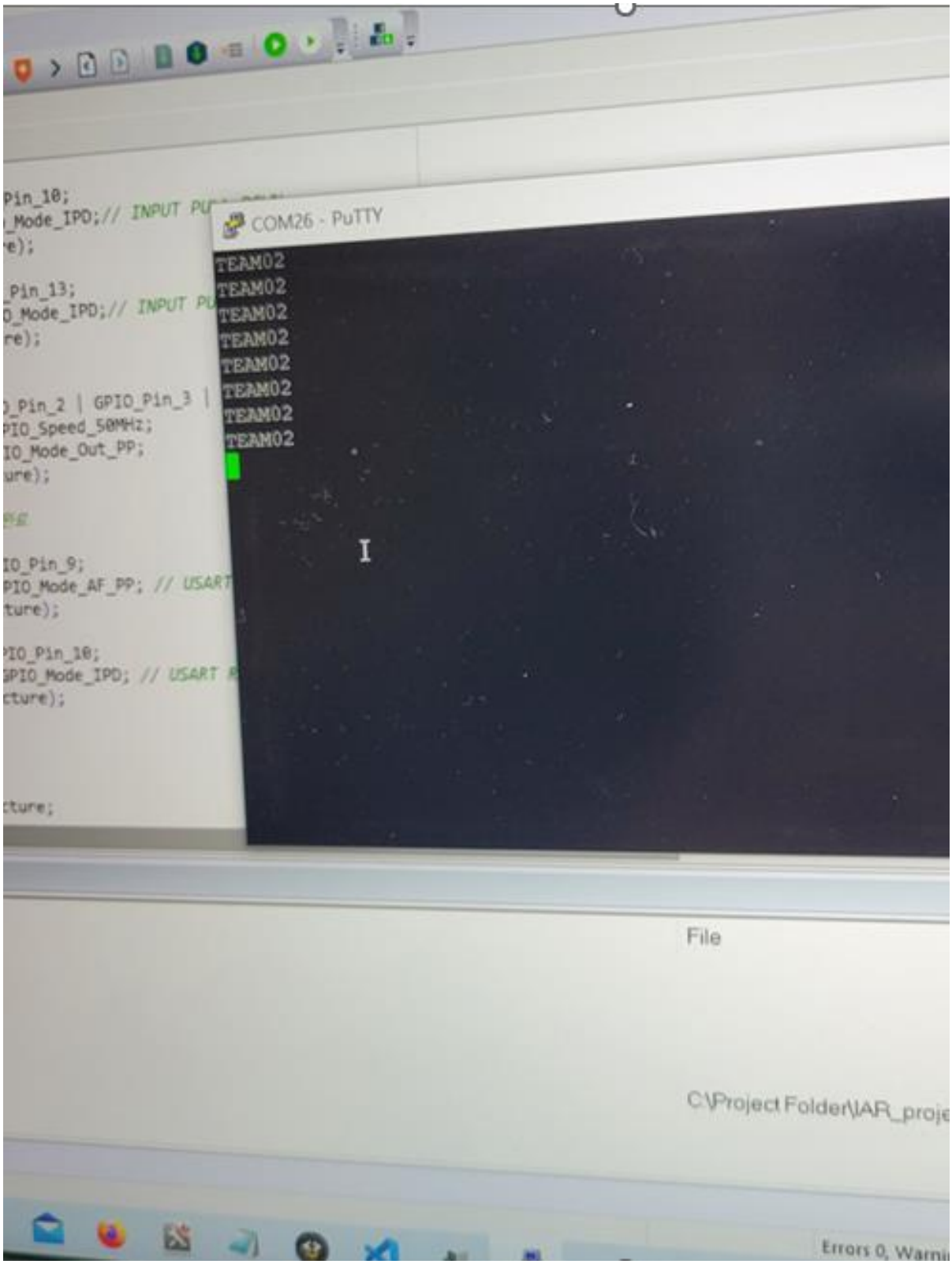

그림9. Putty 동작

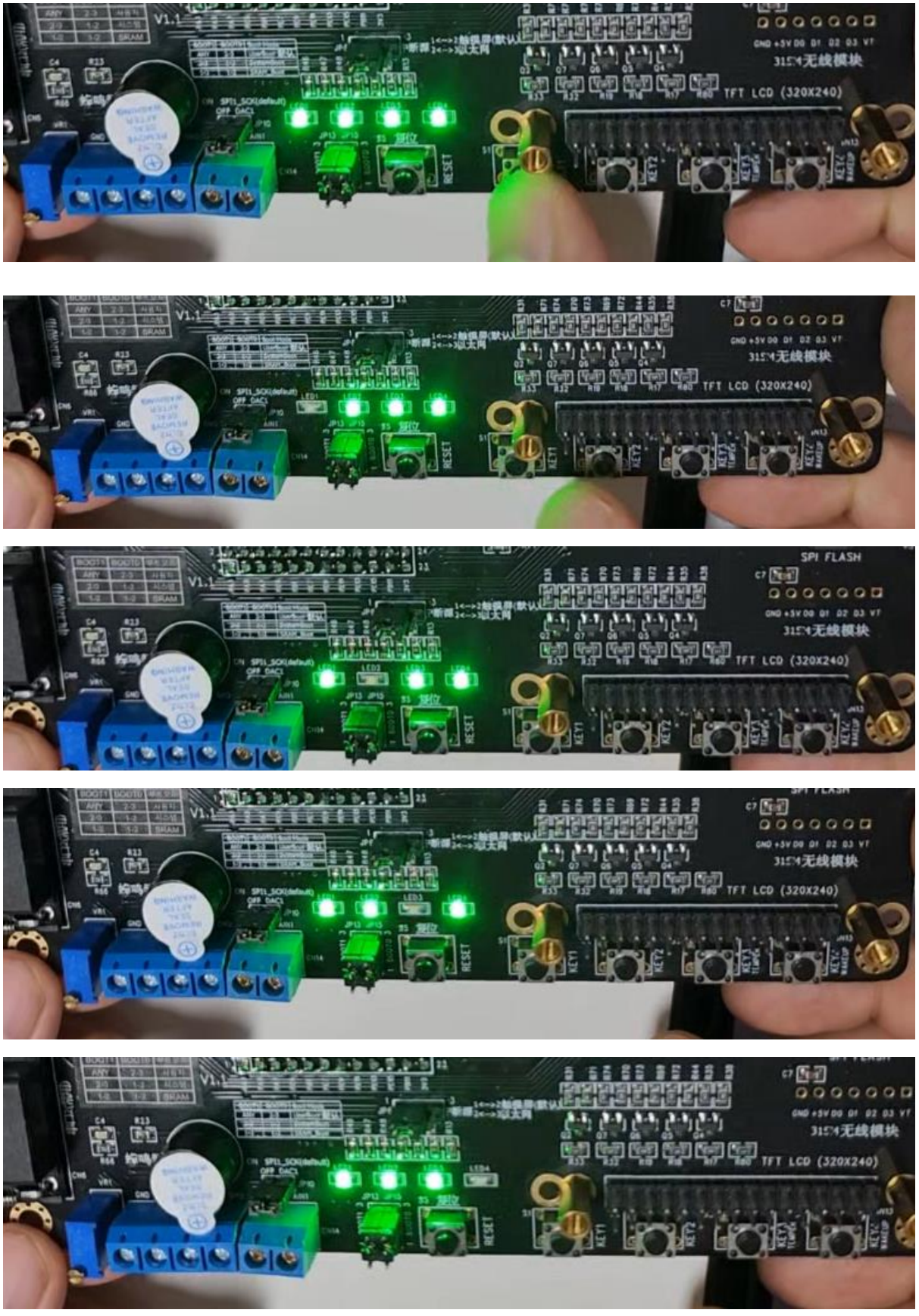

그림10. A 동작

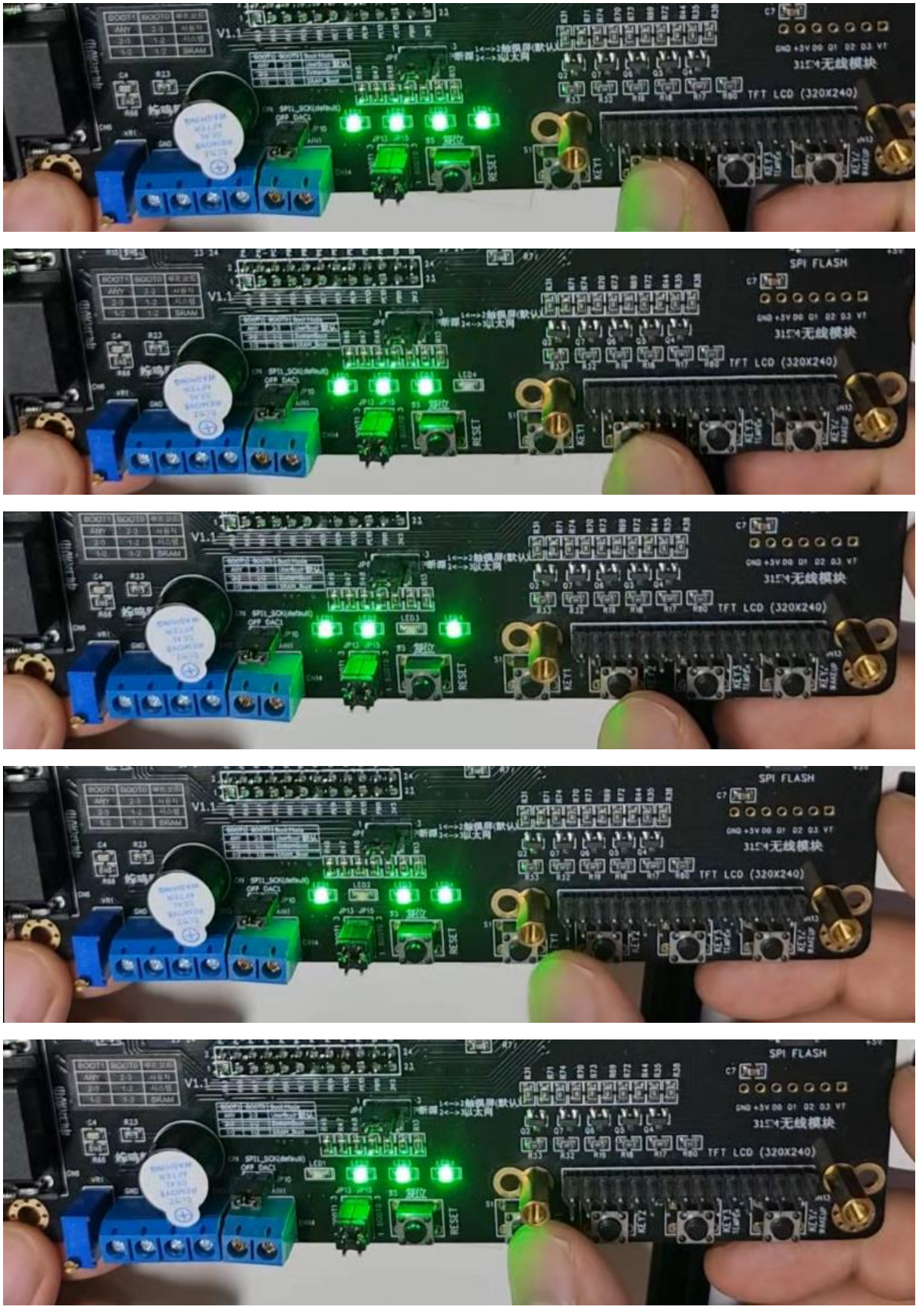

그림11. B 동작