임베디드 시스템 설계 및

실험 004 분반 - 2 조 6

주차

실험 보고서

임베디드시스템설계및실험
2 조 7 주차결과보고서

실험자	202055606 주우성 202055623 허치영 202255632 벌드바타르 아마르투브신 201724637 오치어 자미안퓨레브 202055629 밧툴가 바잘삿
실험날짜	2024-10-17
제출날짜	2024-10-30

1. 실험 제목

GPIO control and UART

2. 실험 목적

- Interrupt 방식과 라이브러리의 구조체를 활용한 GPIO 제어 및 UART 통신 라이브러리 함수 사용법을 숙지한다.

3. 실험 원리 및 이론

1. Polling 과 Interrupt

Polling 방식은 CPU 가 특정 이벤트를 처리하기 위해 이벤트가 발생할 때까지 모든 연산을 이벤트가 발생하는지 감시한다.

Interrupt 방식은 CPU 가 특정 이벤트 발생 시 현재 작업을 멈추고 해당 인터럽트 서비스 루틴을 수행 후 다시 이전 작업으로 돌아간다.

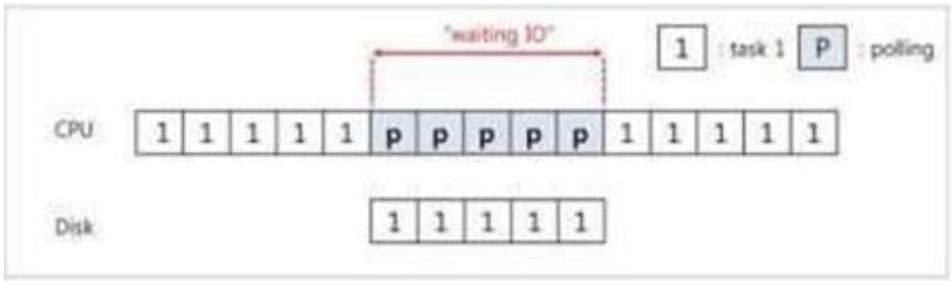


그림 1:Polling

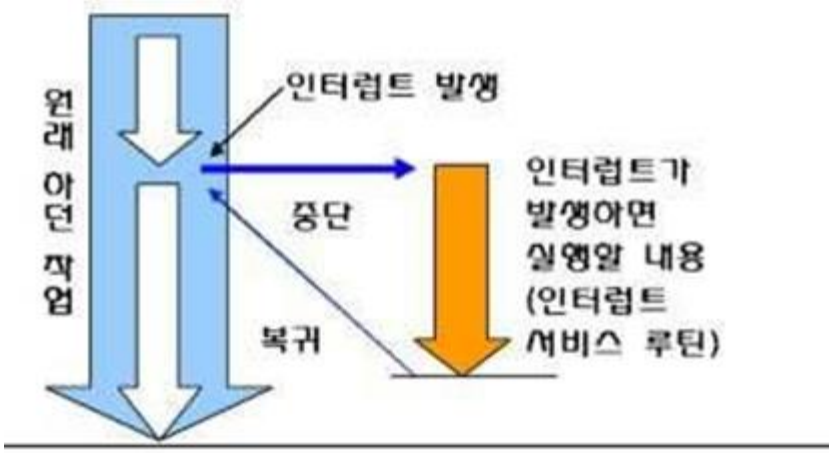


그림 2:Interrupt

실습이 이 방식을 활용한다.

2. EXTI 의개념

EXTI 는 External Interrupt 로 외부에서 신호가 입력될 경우 디바이스에 event 나 interrupt 가 발생된다. 입력 받을 수 있는 클락 신호는 RisingEdge, FallingEdge, Rising & FallingEdge 이다.

모든 GPIO 핀들은 EXTI line 을 통해 연결되어 있다. 외부 Interrupt 는 EXTI0 EXTI15 까지 각 Port 의 Pin 번호가 Interrupt Pin 과 매치된다. 예를 들어 PA0, PB0 은 EXTI0 에 연결된다.

EXTI 는 Event Mode 와 Interrupt Mode 를 선택하여 설정 가능한데 실 험에서는 Interrupt 만 사용했다. Interrupt Mode 로 설정할 경우 Interrupt 가 발생해 해당 Interrupt Handler 가 동작한다.

프로세서는 Interrupt 를 처리하기 전에 우선순위를 체크하여 우선순위가 가장 높은 인터럽트를 처리한다.

EXTICR1 레지스터를 통해 입력받을 포트를 선택하며 같은 번호의 핀들

은 같은 라인을 공유한다.

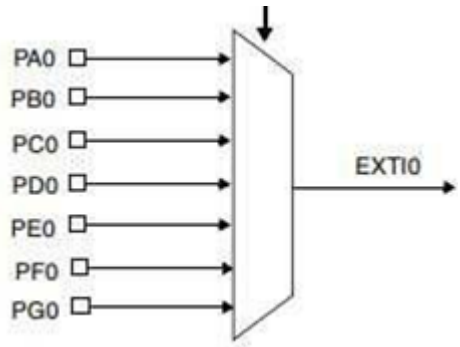


그림 3: EXTI 예시

3. NVIC 의개념

NVIC 는 Nested Vectored Interrupt Controller 로 처리를 위해 벡터에서 대기하는 인터럽트를 제어하는 것이다. 인터럽트 처리 중 또 다른 인터럽트 발생 시 우선순위를 사용한다. 우선순위가 높은 인터럽트부터 처리 후 다른 인터럽트 처리한다. ARM 보드에서 인터럽트 사용 시 NVIC 통하여 우선순위를 결정한다.

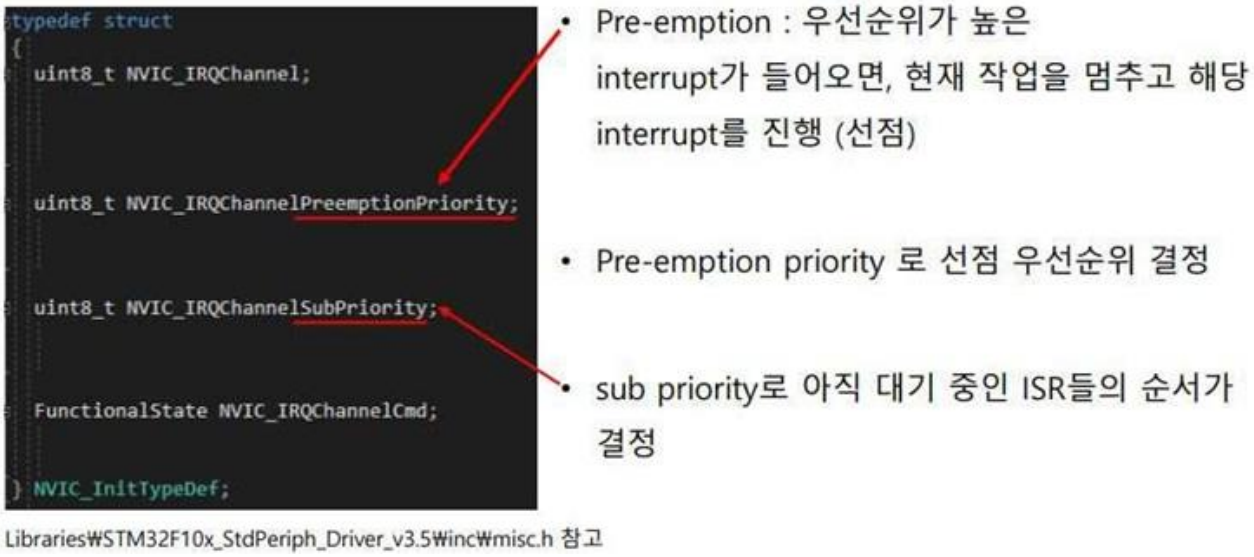


그림 4: misc.h 파일의 NVIC InitTypeDef 구조체

4. 세부 실험 내용

1. TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'

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

    /* Button 1,2,3 port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC |
RCC_APB2Periph_GPIOB, 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);
```

그림 5 에서는 이전과 달리 RCC_APB2PeriphClockCmd(uint32t RCC_APB2Periph, FunctionalState NewState)라는 함수를 사용하여 더욱 쉽게 APB2 관련 포트들을 enable 시키는 모습을 볼 수 있다. 하나만 살펴보면, RCC APB2PeriphClockCmd(RCC APB2Periph GPIOA, ENABLE);는 RCC 의 APB2GPIOA 포트를 활성화시킨다는 의미이다.

2. GPIOConfiguration

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

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

    /* Button 1,2,3 pin setting */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    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_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_InitStructure.GPIO_Mode |= GPIO_Mode_IPU;

    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

그림 5 에서 RCC, APB2 를 enable 하였으니, 그림 6 에서는 사용할 포트의 Pin 들을 setting 한다. GPIO Pin 설정을 위해 추가 라이브러리의 GPIO_InitTypeDef 구조체와 GPIO Init 을 사용하였다. Go to definition 을 이용하여, 구조체를 확인하고 구조체 변수들(Pin, Mode, Speed)을 설정해 주었다. 그 후 GPIO Init 을 통해 어떤 포트의 핀인지를 알려주는 모습이다.

3. EXTIConfiguration

```
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 */
    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
}
```

이번 실습은 **Interrupt(인터럽트)**를 이용하여 장치에 입력을 처리한다. 이론에서 언급했듯이 모든 **GPIO 핀들은 EXTI 라인**과 연결될 수 있다. 따라서 **어떤 GPIO 핀을 EXTI 라인에 매핑**할지 선택해야 한다. 또한, **EXTI 설정**도 함께 진행해야 한다.

코드에서 `GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource4);`를 보면 **GPIOC 포트의 4 번 핀을 EXTI 라인 4 에 매핑**하고 있다.

EXTI 설정은 `EXTI_InitTypeDef` 구조체를 통해 이루어지며, 이 구조체의 변수들을 다음과 같이 설정한다:

- **EXTI_Line**: 사용할 EXTI 라인 (예: 4, 10, 13 번)
- **EXTI_Mode**: 인터럽트 모드로 설정
- **EXTI_Trigger**: 하강 엣지에서 트리거 (버튼이 눌릴 때 발생)
- **EXTI_LineCmd**: 해당 EXTI 라인 활성화

이렇게 설정한 구조체는 `EXTI_Init()` 함수를 통해 적용한다.

4. USART 설정 (USART1_Init)

```
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_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    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);
}
```

UART 통신을 위해서 우리는 USART1 장치를 사용한다.

따라서 USART Cmd(USART1, ENABLE);으로 USART1 을 활성화시키고 USART InitTypeDef 구조체를 이용하여 그림 8 에서처럼 Baud Rate 와 같은 통신에 필요한 값들을 설정해 주었다.

(통신에 필요한 값은 지난 주차 설정했던 것들을 참고했다.)

```
/**
  * @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
  */
```

그림 9: Todo USART IT value

설정한 **USART Interrupt**를 활성화시키기 위해서 **USART ITConfig(USART1, USART IT RXNE, ENABLE);**를 진행하였다.

여기서 **USART IT RXNE**는 그림 9에서 보는 것처럼 **"Receive Data register not empty interrupt"**라는 뜻이다.

5. NVICConfiguration

```
void NVIC_Configure(void)

{


    NVIC_InitTypeDef NVIC_InitStructure;


    // TODO: fill the arg you want
    // 선점을 위해 그룹 1 이상 사용


    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
```

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


    // Button1
    NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =
0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);


    // Button2,3
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =
0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    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);
}
```

`NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);`

이 함수는 **NVIC 의 우선순위 그룹**을 설정한다. 이번 실습에서는 **우선순위 설정이 크게 중요하지 않기 때문에**, **Priority Group 1**을 선택하였다. 이 그룹은 **선점 우선순위**와 **서브 우선순위**를 모두 사용할 수 있도록 한다.

`NVIC_InitTypeDef` 구조체와 **`NVIC_Init` 함수**를 사용해 **버튼과 USART1**에 대한 NVIC 를 설정하였다. 코드에서는 **선점 우선순위와 서브 우선순위 모두 0**으로 설정하여 동일한 우선순위를 가지도록 하였다.

이전 실험에서는 조이스틱의 입력을 **PC2(EXTI2_IRQn)**와 **PC5(EXTI9_5_IRQn)**으로 처리했지만, 이번 실습에서는 **버튼(EXTI4_IRQn, EXTI15_10_IRQn)**과 **USART1**에 대해 동일한 방법으로 설정하였다.

6. 메인 함수 구현 (main)

TODO: 인터럽트 발생 시 실행되는 핸들러를 구현한다.

- **EXTI4_IRQHandler**: 버튼 1 이 눌리면 `mode` 를 0 으로 설정한다.
- **EXTI15_10_IRQHandler**: 버튼 2 와 3 이 눌리면 `mode` 와 `usart_signal` 을 변경한다.
- **USART1_IRQHandler**: UART 로 데이터가 수신되면 `mode` 를 변경한다.

7. main 함수

```c
int main(void)
{

    SystemInit();

    RCC_Configure();

    GPIO_Configure();

    EXTI_Configure();

    USART1_Init();

    NVIC_Configure();

    int led = 0;
    char *msg = "TEAM02.\r\n";
    while (1)
    {
        // TODO: implement
        if (mode == 0)
        {
            led = (led + 1) % 4;
        }
        else if (mode == 1)
        {
            if (led == 0)
            {
                led = 4;
            }
            led = (led - 1) % 4;
        }

        // Turn off all led
        GPIO_SetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_4 | GPIO_Pin_7);
        // Turn on the led
        if (led == 0)
        {
            GPIO_ResetBits(GPIOD, GPIO_Pin_2);
        }
        else if (led == 1)
        {
            GPIO_ResetBits(GPIOD, GPIO_Pin_3);
```

```
        }
        else if (led == 2)
        {
            GPIO_ResetBits(GPIOD, GPIO_Pin_4);
        }
        else if (led == 3)
        {
            GPIO_ResetBits(GPIOD, GPIO_Pin_7);
        }

        if (usart_signal == 1)
        {
            for (int i = 0; i < 9; i++)
            {
                sendDataUART1(msg[i]);
            }
            usart_signal = 0;
        }

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

1. **기본 설정**

먼저 시스템 초기화와 함께 **클럭(RCC)**, **GPIO 핀**, **외부 인터럽트(EXTI)**, **UART 통신(USART)**, 그리고 **NVIC** 설정을 완료한다.

2. **LED 및 메시지 초기화**
   - **`led` 변수**는 현재 켜질 LED 의 번호를 저장한다.
   - **`msg` 변수**는 UART 를 통해 **"TEAM02.₩r₩n"** 메시지를 전송하기 위해 사용된다.

3. **LED 제어 반복 루프**
   - 프로그램은 **무한 루프**(`while (1)`) 안에서 동작한다.
   - **mode 가 0**일 때, LED 가 **순차적으로 켜지는 물결**을 진행한다.
   - **mode 가 1**일 때, LED 가 **역순으로 켜지며** 물결이 반대로 흐른다.
   - 이때 `led` 번호가 **0 보다 작거나 4 를 넘지 않도록** 모듈로 연산을 사용해 제어한다.

4. **LED 상태 업데이트**
   - 반복문이 돌 때마다 **모든 LED 를 끈 후**, 현재 켜야 할 LED 만 **킨다**.
   - LED 는 4 개(PD2, PD3, PD4, PD7)를 사용하며, **모드에 따라 켜지는 순서**가 달라진다.

5. **UART 메시지 전송**

- **`usart_signal`이 1**로 설정되면 UART 를 통해 **"TEAM02.₩r₩n" 메시지가 Putty 콘솔에 출력**된다.

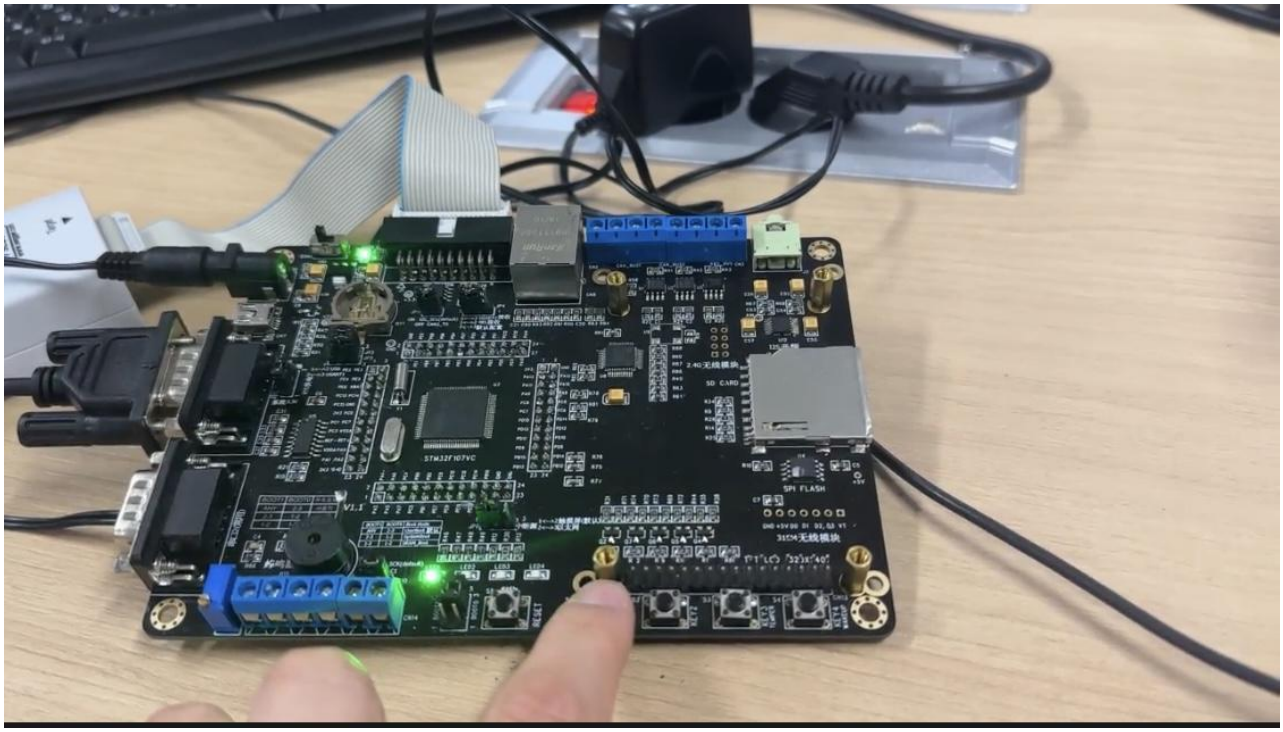
- 전송이 완료되면 `usart_signal`을 다시 0 으로 초기화한다.

6. **딜레이 처리**

- **`Delay()` 함수를 통해 LED 물결의 속도를 조절**하며 반복 루프가 너무 빠르게 실행되지 않도록 한다.

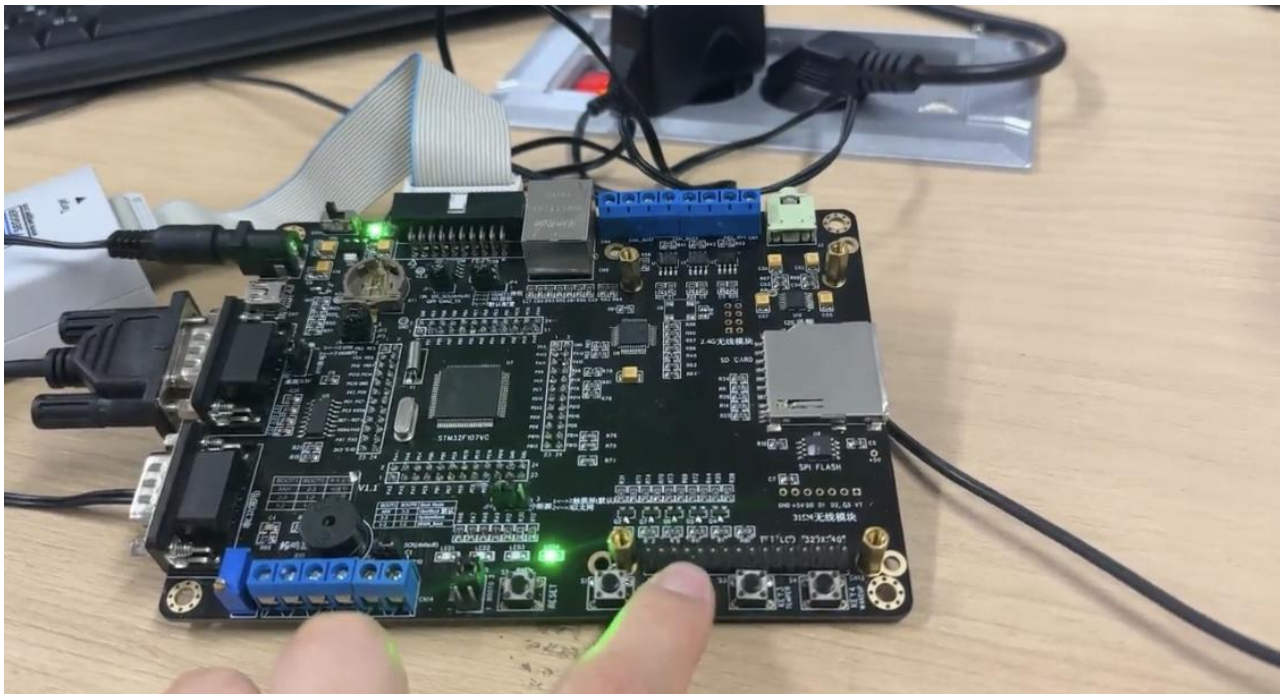
5. 실험 결과

버튼 1 누르다.



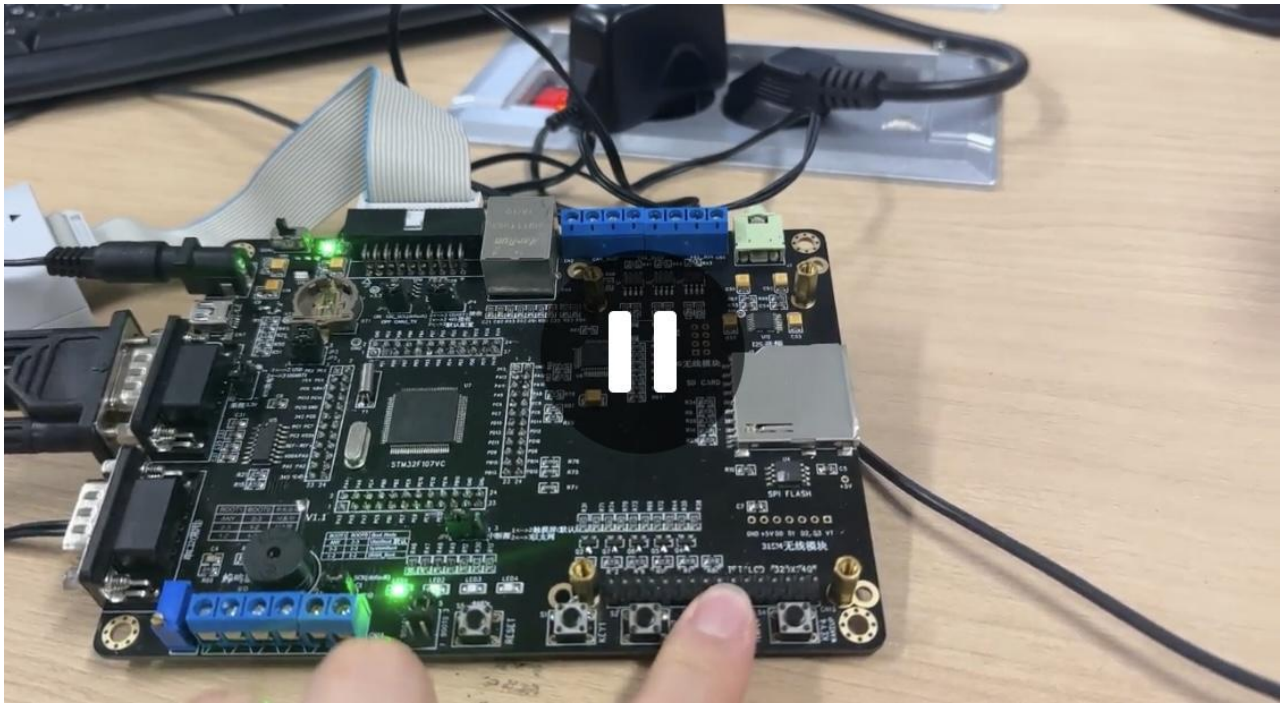
LED 가 오른쪽으로 간다

버튼 2 누르다.

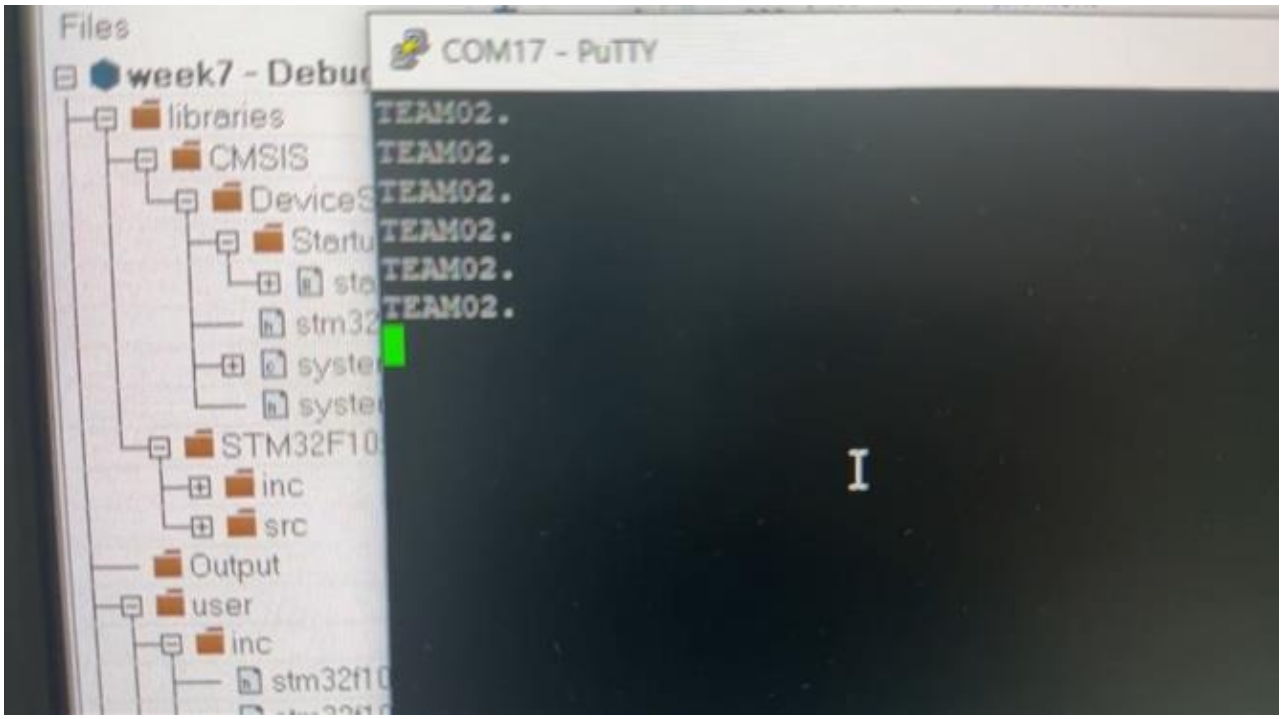


LED 가 왼쪽으로 갑니다

버튼 3 누르다.

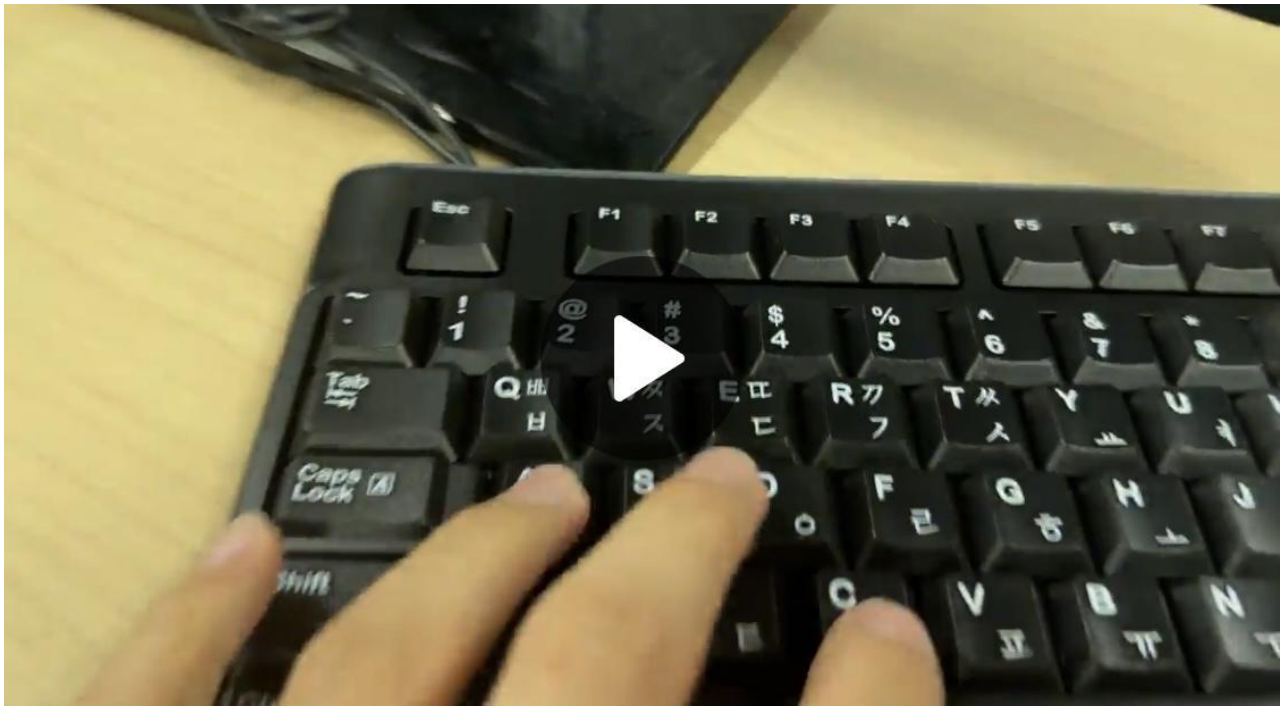


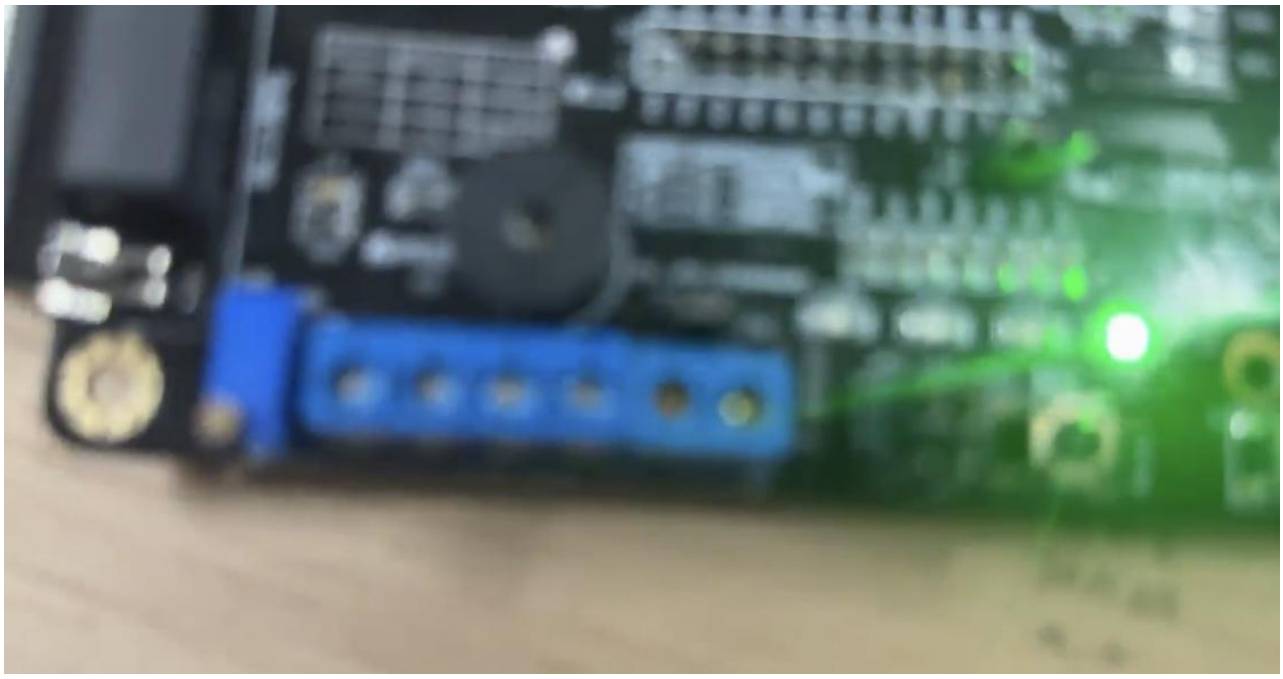
버튼을 누르면 PuTTY 에 "Team02"가 출력됩니다.



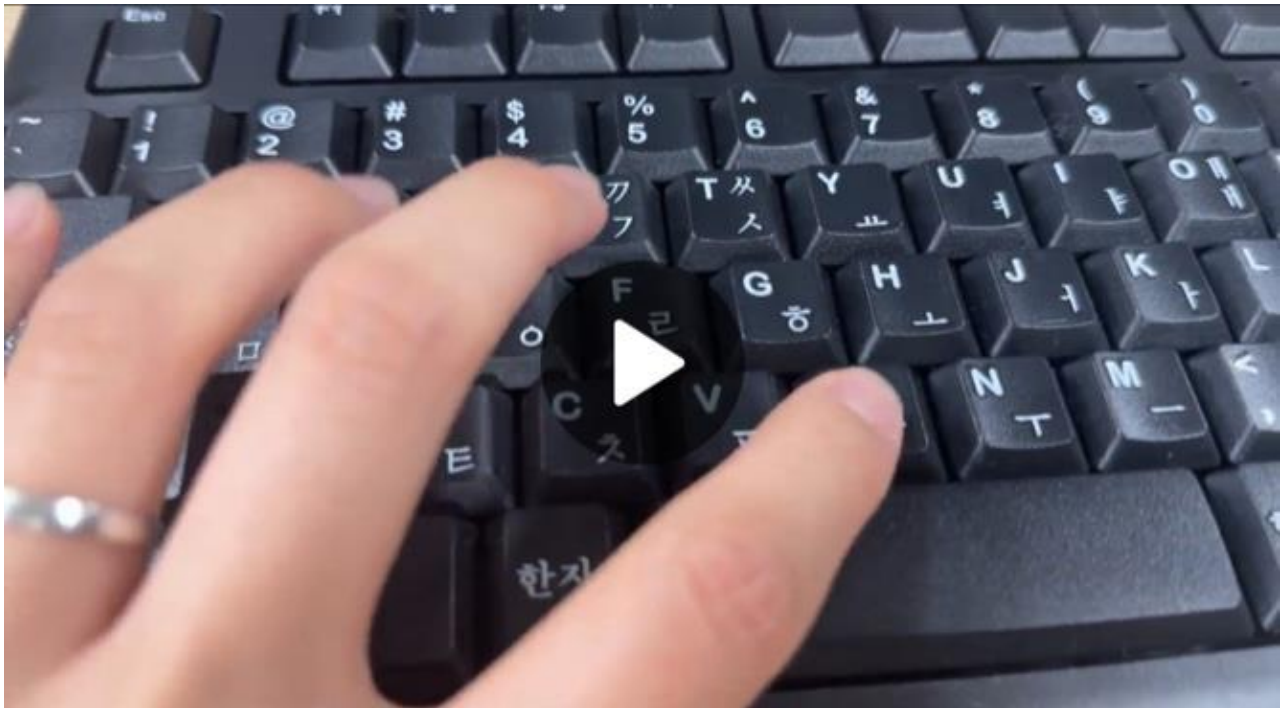
Putty 를통한 LED 물결제어

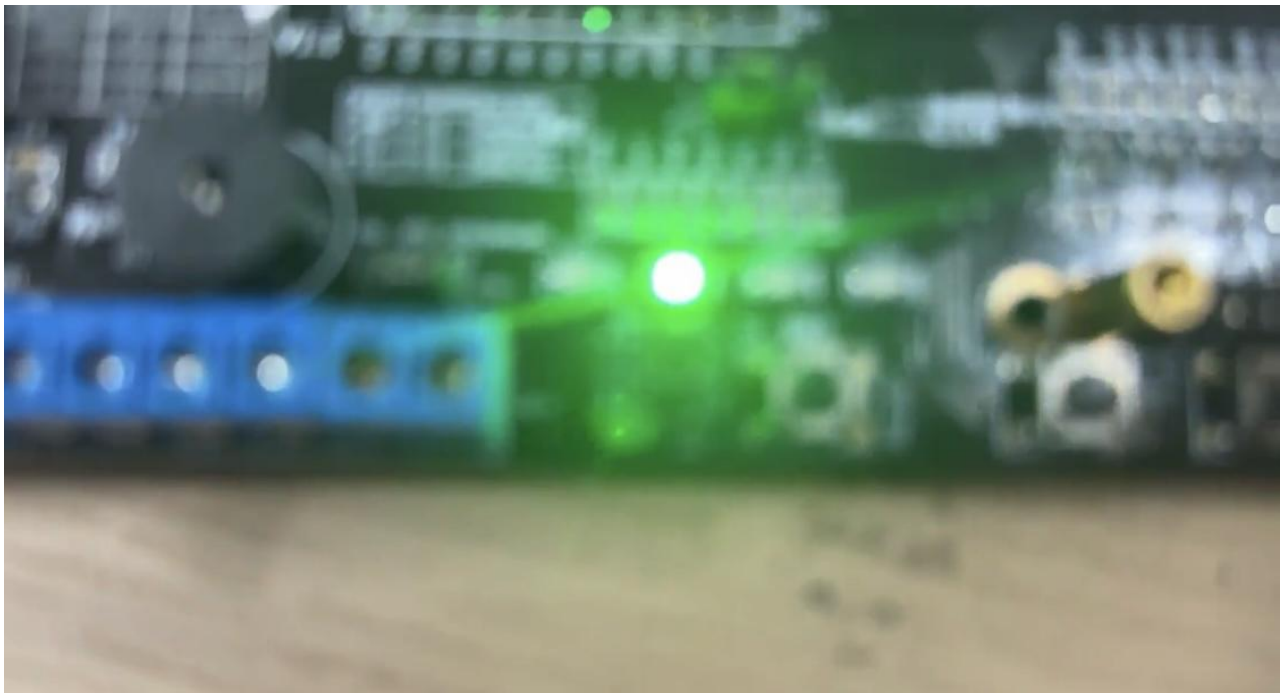
Key A LED 가 오른쪽으로 간다





Key B LED 가 왼쪽으 로 갑니다





6. 분석 및 고찰

이번 실습을 통해 **STM32 마이크로컨트롤러**에서 **GPIO 제어, 외부 인터럽트(EXTI), USART 통신**, 그리고 **NVIC 우선순위 설정**의 중요성을 이해할 수 있었습니다. **버튼 입력과 UART 통신**에 따라 LED 를 제어하는 기능을 구현하며, 각 모듈이 상호 작용하는 방식을 실습했습니다.

특히, **모드 전환을 통한 LED 제어**와 **UART 메시지 전송**을 통해 실시간 데이터 처리와 인터럽트 기반 제어의 원리를 익혔습니다. NVIC 우선순위 설정은 단순한 시스템에서는 큰 영향을 주지 않았지만, 복잡한 시스템에서는 매우 중요한 역할을 한다는 것을 확인했습니다.

이번 실습을 통해 STM32 의 다양한 기능을 실습하며 **임베디드 시스템의 동작 원리**를 이해하는 데 큰 도움이 되었습니다. 앞으로는 **타이머를 활용한 더 정밀한 시간 제어**를 적용해보며 시스템의 안정성과 정확도를 높이는 방법을 연구해 볼 필요가 있습니다.

7. 참고자료

STM32107VCT6 schematic

stm32 Datasheet

stm32 Reference Manual