임베디드시스템 설계 및 실험 보고서

[002 분반 - 2 조 - 7 주차]

조원	202055531 김후겸 202055584 이태경 202155540 김채현 202255535 김진우
실험날짜	2024-10-16

1. 실험 주제

- Interrupt 방식을 활용한 GPIO 제어 및 UART 통신

2. 실험 목적

- Interrupt 방식을 활용한 GPIO 제어 및 UART 통신
- 라이브러리 함수 사용법 숙지

3. 세부 목표

- Datasheet 및 Reference Manual 을 참고하여 해당 레지스터 및 주소에 대한 설정 이해
- NVIC 와 EXTI 를 이용하여 GPIO 에 인터럽트 핸들링 세팅
- 버튼 S1, S2 동작 설정
  (S1: A, S2: B, A: 1 -> 2 -> 3 -> 4, B : 4 -> 3 -> 2 -> 1)
- PC 의 Putty 에서 a, b 문자를 입력하여 보드 제어 (‘a’ : A 동작, ‘b’ : B 동작)
- Putty 로 “TEAM02.₩r₩n” 출력하도록 버튼 S3 설정

4. 실험 장비

- STM32F107VCT6
- IAR Embedded Workbench (EW)

5. 실험 과정(코드설명)

1) 프로젝트 생성 및 파일 구조 설정

3 주차와 동일

2) RCC Configuration

```
void RCC_Configure(void)
{
    // TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); // KEY1(PC4)
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); // KEY2(PB10)

    /* UART TX/RX port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);

    /* Button S1, S2, S3 port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    /* LED port clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

이번 실습부터는 RCC_APB2PeriphClockCmd 라는 함수를 사용하여 APB2 관련 포트들을 enable 한다. Enable 하고 싶은 포트를 첫 번째 인자로, enable 여부를 두 번째 인자로 받는다.

예를 들어, RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); 코드는 RCC 의 APB2GPIO A 포트를 Enable 하겠다는 의미이고, 나머지도 장치에 맞게 포트들을 enable 해준다.

첫 번째 인자로 들어가는 포트들의 명칭은 stm32f10x_rcc.h 파일에서 찾을 수 있다.

3) GPIO Configuration

```
void GPIO_Configure(void)
{
    GPIO_InitTypeDef KEY1_3_InitStructure;
    KEY1_3_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_13;
    KEY1_3_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOC, &KEY1_3_InitStructure);

    GPIO_InitTypeDef KEY2_InitStructure;
    KEY2_InitStructure.GPIO_Pin = GPIO_Pin_10;
    KEY2_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOB, &KEY2_InitStructure);

    GPIO_InitTypeDef LED_InitStructure;
    LED_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_7;
    LED_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    LED_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &LED_InitStructure);
```

주어진 GPIO_InitTypeDef 구조체와 GPIO_Init 함수를 사용하여 GPIO Pin 을 설정한다.

GPIO_InitTypeDef 구조체에서는 Pin, Mode, Speed 를 변수로 갖기 때문에 각 장치에 맞게 Pin, Mode, Speed 를 설정해준다.

그 후 GPIO_Init 을 통해 해당 포트의 해당 핀을 설정에 맞게 초기화 시킨다.

예를 들어 LED 의 경우, 출력으로 쓰일 LED 가 총 4 개이기 때문에 각 LED 의 Pin 을 or 을 통해 할당하고, Speed 를 50MHz 로 설정, Mode 를 push pull 출력 모드로 설정하고 초기화하는 코드이다.

구조체의 구조, 변수, 변수로 들어오는 값들은 모두 stm32f10x_gpio.h 파일에서 확인할 수 있다.

4) EXTI Configuration

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

    /* Button S1 */
    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);
```

이번 실습은 Interrupt 를 발생하기위해 EXTI 를 사용한다.

GPIO 핀들은 EXTI line 을 통해 연결되어 있어, 어떤 EXTI Line 에서 우리가 사용할 GPIO 핀들을 사용할지 선택하고 설정을 한다.

GPIO EXTILineConfig 함수를 통해 GPIO 핀을 선택하고, EXTI InitTypeDef 라는 구조체를 이용하여 구조체의 변수 Line, Mode, Trigger, LineCmd 를 설정한다.

예를 들어, 버튼 1 의 경우 GPIO C 포트의 4 번 핀을 Configuration 하고 있다. 그리고 Line 은 핀 번호와 동일한 4 를, Mode 는 Interrupt, Trigger 은 Falling, LineCmd 는 ENABLE 로 설정한다.

구조체의 구조, 변수, 변수로 들어오는 값들은 모두 stm32f10x_exti.h 파일에서 확인할 수 있다.

5) USART1 설정

```
void USART1_Init(void)
{
    USART_InitTypeDef USART1_InitStructure;

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

USART 통신을 사용하기위해 USART1 을 사용한다.

USART_Cmd 를 사용해 USART1 를 ENABLE 하고 USART_InitTypeDef 구조체를 이용하여 필요한 변수들을 설정해준다. 값은 저번주차를 참고하여 설정하였다.

설정한 USART Interrupt 를 enable 하기 위해 USART_IFConfig 함수를 사용하였다.

구조체의 구조, 변수, 변수로 들어오는 값들은 모두 stm32f10x_usart.h 파일에서 확인할 수 있다.

6) NVIC 설정

```
void NVIC_Configure(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    NVIC_InitTypeDef KEY1_InitStructure;
    KEY1_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
    KEY1_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0;
    KEY1_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
    KEY1_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&KEY1_InitStructure);
```

NVIC PriorityGroupConfig 함수를 통해 NVIC 의 우선순위를 설정하였다.

NVIC InitTypeDef 구조체와 NVIC Init 을 통해 버튼, US ART1 의 NVIC 를 설정하였다.

주 우선순위와 부 우선순위를 모두 0 으로 설정하여 동일한 우선순위를 갖게 하였고, 버튼 1 은 PC 4 라서 EXTI4_IRQn 를, 버튼 2,3 은 각각 PB10, PC13 여서 EXTI15_10_IRQn 를, USART1 은 USART1_IRQn 을 값으로 넣어주었다.

7) 인터럽트 핸들러

```c
void USART1_IRQHandler()
{
    uint16_t word;
    if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        word = USART_ReceiveData(USART1);

        if (word == 'a')
        {
            sendDataUART1('a');
            wave = 1;
        }
        else if (word == 'b')
        {
            sendDataUART1('b');
            wave = -1;
        }

        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
```

```c
void EXTI15_10_IRQHandler(void)
{

    if (EXTI_GetITStatus(EXTI_Line10) != RESET)
    {
        if (GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_10) == Bit_RESET)
        {
            wave = -1;
        }
        EXTI_ClearITPendingBit(EXTI_Line10);
    }

    if (EXTI_GetITStatus(EXTI_Line13) != RESET)
    {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13) == Bit_RESET)
        {
            char msg[] = "Team02\r\n";
            for (int i = 0; i < sizeof(msg); i++)
            {
                sendDataUART1(msg[i]);
            }
        }
        EXTI_ClearITPendingBit(EXTI_Line13);
    }
```

```c
void EXTI4_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line4) != RESET)
    {
        if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_4) == Bit_RESET)
        {
            wave = 1;
        }
        EXTI_ClearITPendingBit(EXTI_Line4);
    }
}
```

우리는 버튼 1,2,3, USART1 의 인터터럽트를 사용하므로 USART1_IRQHandler, EXTI15_10_IRQHandler, EXTI4_IRQHandler 라는 그림에서 정의된 이름으로 함수를 사용한다.

인터럽트 핸들러에서 많은 연산이 진행디면 효율이 좋지 않으므로 간단한 조건문을 통해 처리하도록 하였다.

USART1_IRQHandler 에서는 putty 에서 a 를 누르면 물결 방향, b 를 누르면 반물결방향이 되도록 설정하였다.

EXTI4_IRQHandler 에서는 버튼을 누르면 A 인터럽트 발생을 판단하도록 하였고,

EXTI15_10_IRQHandler 에서는 버튼 2 를 누르면 B 인터럽트 발생을 판단하도록, 버튼 3 을 누르면 TEAM02 를 출력하도록 하였다.

8) Main 설정

```
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);

    i += wave;
    i = (i + 4) % 4;

    //GPIO_ResetBits(GPIOD, led[i]);

    if (i == 0) {
        GPIOD->BSRR = ~GPIO_Pin_2;
    }
    if (i == 1) {
        GPIOD->BSRR = ~GPIO_Pin_3;
    }
    if (i == 2) {
        GPIOD->BSRR = ~GPIO_Pin_4;
    }
    if (i == 3) {
        GPIOD->BSRR = ~GPIO_Pin_7;
    }

    Delay();
}
```

while 문을 사용하여 led 가 순서에 맞게 출력되도록 하였다.

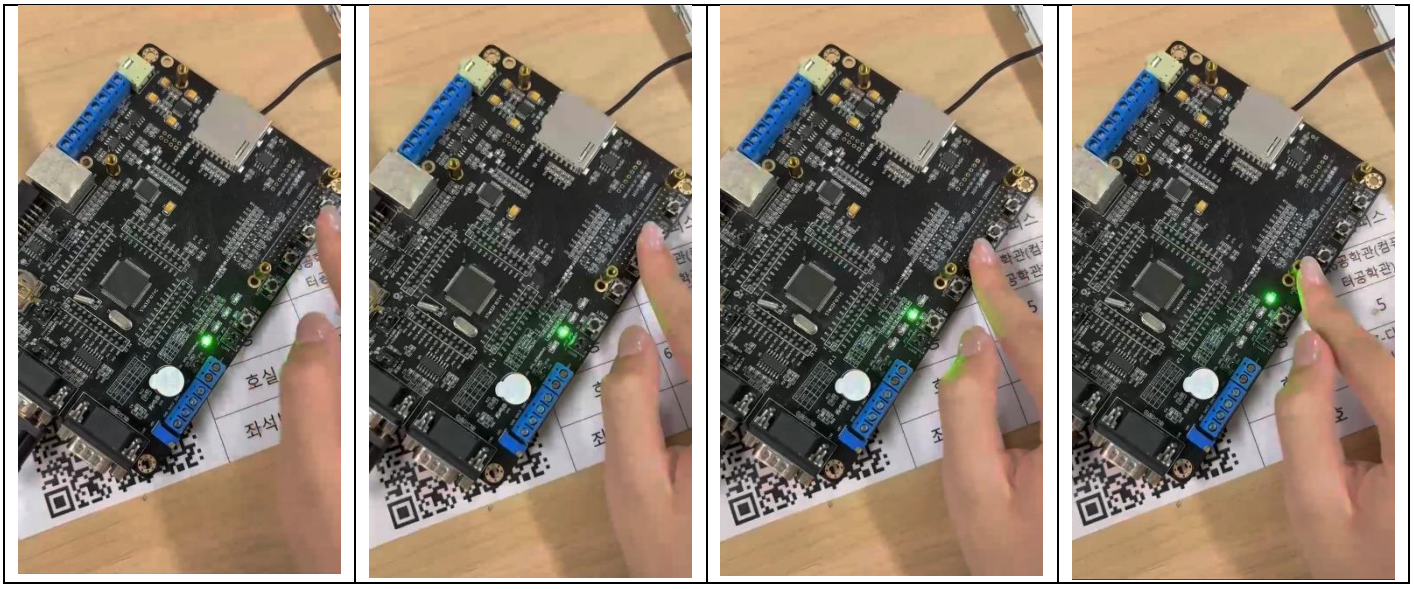
버튼 S1 을 누르거나 Putty 에 a 를 입력 했을 경우 인터럽트 핸들러에 의해 wave 가 1 로 설정되어 1->2->3->4->1 순서로 출력이 되고,

버튼 S2 를 누르거나 Putty 에 b 를 입력했을 경우 인터럽트 핸들러로 인해 wave 가 -1 로 설정되어 1->4->3->2->1 순ㄴ서로 출력이 된다.

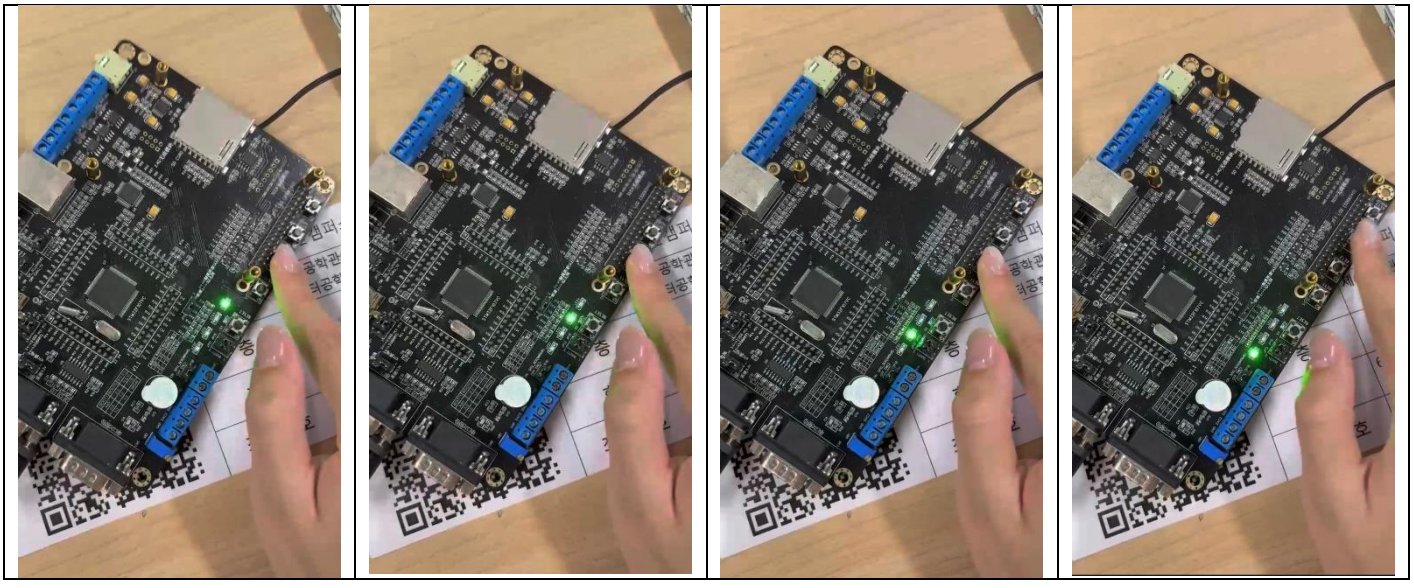
버튼 S3 을 누르면 위에서 설정한대로 Team02 가 출력된다.

6. 실험 결과

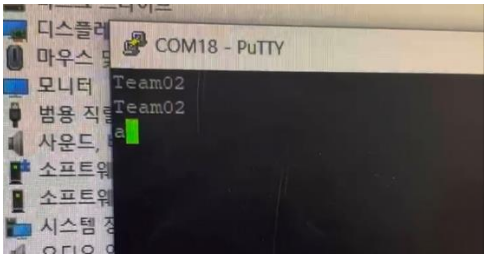
1) S1 버튼



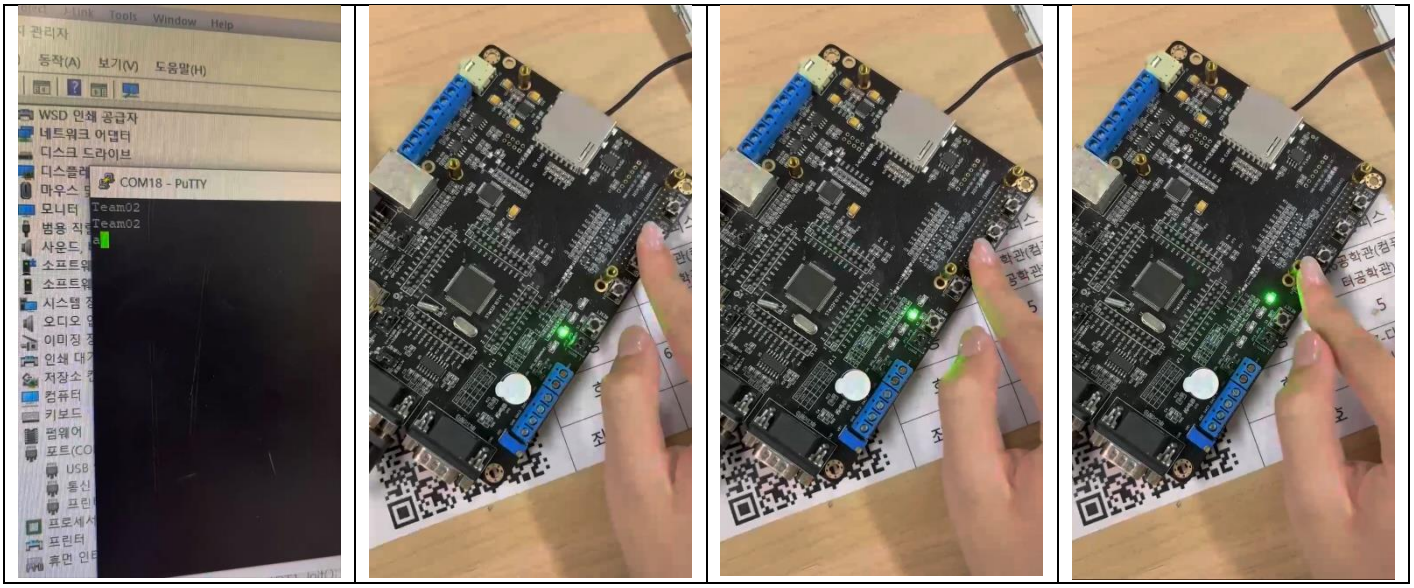
2) S2 버튼



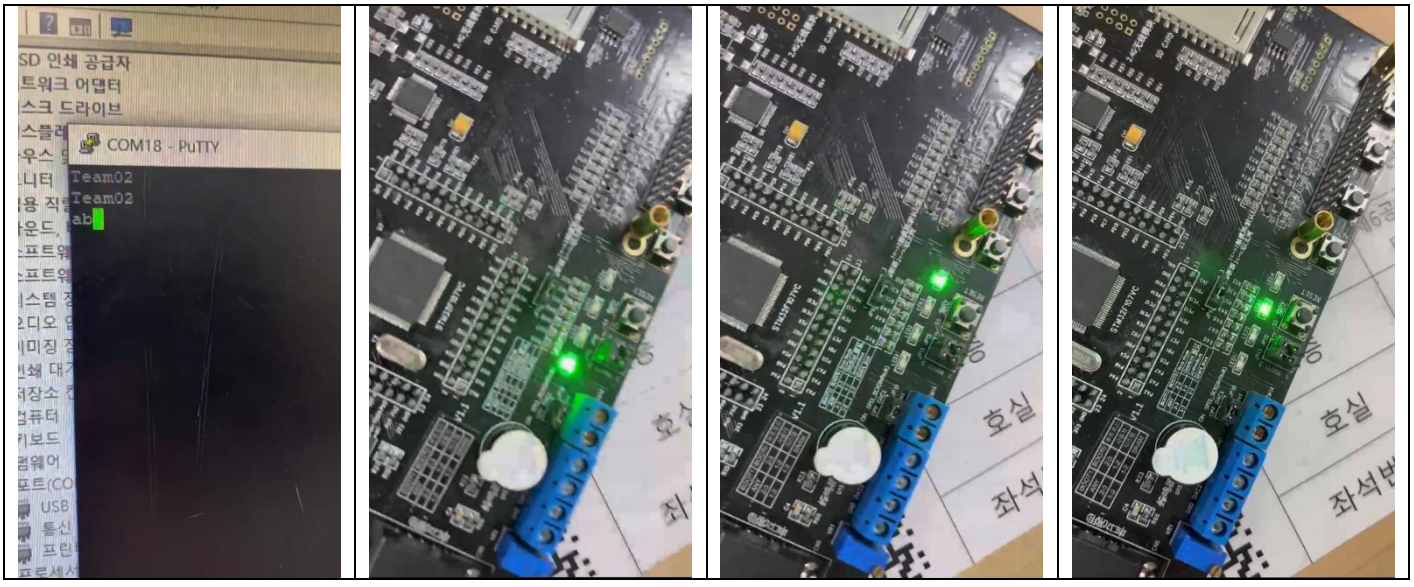
3) S3 버튼



4) a 입력



5) b 입력



7. 분석 및 결론

- 7 주차 실험은 라이브러리를 사용하여 다양한 구조체와 함수들을 사용하였다. 구조체의 변수값과 함수의 인자들을 사용하여 이전에 일일히 주소 값을 찾아야 했던 것과 다르게 보다 쉽게 설정을 할 수 있었다.

인터럽트 방식을 사용해보니 어떤 동작을 수행하는 도중에도 NVIC 를 통한 우선순위를 설정하여 우리가 먼저 하길 원하는 동작을 수행할 수 있었다.

구조체 파일의 주석을 이해하는데 전반적으로 시간이 쓰였지만 인터럽트가 동작하는 과정을 보면서 하니 직관적이어서 이해가 쉬웠다.

8. 참고자료

- stm32_Datasheet.pdf
- stm32_ReferenceManual.pdf
- STM32107VCT6_Schematic.pdf