임베디드시스템 설계 및 실험

8조 10주차 실험 결과 보고서

과 목 : *임베디드설계및실험*

조 원 : *박동진*(201824469) *신재환*(201824514) *예진욱*(201824528) *윤건우*(201824532)

분 반 : 002*분반*

실험 목표

1. *타이머란*

1.1 주기적 시간 처리에 사용하는 디지털 카운터 회로 모듈

1.2 펄스폭 계측, 주기적인 interrupt 발생 등에 사용

1.3 주파수가 높기 때문에 우선 prescaler를 사용하여 주파수를 낮춘 후 낮아진 주파수로 8, 16 비트 등의 카운터 회로를 사용하여 주기를 얻는다.

1.4 STM32 타이머 종류

- SysTick Timer
- Watchdog Timer
- Advanced-control Timer (TIM1, TIM8)
- General-purpose Timer (TIM2 ~ TIM5)
- Basic Timer (TIM6 , TIM7)

2. *타이머의 종류와 특징*

2.1 Systick Timer

: Real-time operating system 전용이지만 standard down counter로 사용 할 수도 있음

- 24bit down counter
- Autoreload capability
- Counter가 0에 도달하면 설정에 따라 인터럽트가 발생
- Programmable clock source

2.2 IWDG/WWDG Timer-Watching timer

: WATCHDOG(WDG)은 임베디드 시스템 등 특수 상황에서 CPU가 올바르게 작동하지 않을 시 강제로 리셋시키는 기능을 의미한다.

- IWDG/WWDG는 모두 소프트웨어 고장으로 인한 오작동을 감지하고 해결하는 역할을
- 카운터가 주어진 시간 초과 값에 도달했을 때 시스템 재설정 또는 인터럽트(only WWDG)를 트리거한다.
- IWDG
  - IWDG는 LSI 클럭 기반으로 메인 클럭 고장에도 활성 상태 유지 가능
  - 타이밍 정확도 제약이 낮은 애플리케이션에 적합하다.
  - 카운터가 0이 되면 Reset
- WWDG

- 7-bit down counter
- WWDG의 클럭은 APB1 클럭을 프리스케일해서 정의 가능
- 비정상적 어플리케이션 동작 감지를 위해 설정 가능한 timer-window가 있다.
- Timer-window 내에서 반응하도록 요구하는 애플리케이션에 적합하다.
- 카운터가 0x40 보다 작을 경우 또는 카운터가 Timer-window 밖에 Reload 됐을 경우 Reset 가능하다.
- Early wakeup interrupt (EWI) : 카운터가 0x40과 같을 때, EWI 인터럽트 발생하게 설정 가능하다.

2.3 Advanced-control timers (TIM1 & TIM8)

- The advanced -control timers는 prescaler를 이용해 설정 가능한 16-bit auto-reload counter를 포함하고 있다.
- 입력 신호 펄스 길이 측정(input capture) 또는 출력 파형 생성(output compare, PWM, complementary PWM with dead-time insertion)등에 사용 가능하다.
- The advanced-control (TIM1 & TIM8) and general-pupose (TIMx)는 자원을 공유하지 않는 독립적인 구조이며, 동기화 시키는 것도 가능하다.

2.4 Basic timer (TIM6 & TIM7)

- 16-bit auto-reload 업카운터
- 설정 가능한 16-bit prescaler를 이용해 the counter clock 주파수를 나눠서 설정 가능
- DAC 트리거에 사용
- 카운터 오버플로우 발생 시 인터럽트/DMA 생성

2.5 General-purpose timers (TIM2 to TIM5)

- General-purpose timer는 prescaler를 이용해 설정 가능한 16-bit up, down, up/down auto-reload counter를 포함하고 있다.
- 입력 신호의 펄스 길이 측정(input capture) 또는 출력 파형 발생(output compare and PWM) 등 다양한 용도로 사용할 수 있다.
- 펄스 길이와 파형 주기는 timer prescaler와 the RCC clock controller prescaler를 사용하여 및 μs에서 몇 ms까지 변조할 수 있다. 타이머들은 완전히 독립적이며, 어떤 자원도 공유하지 않으나 동기화 가능하다.

3. *분주 계산*

- 분주란 MCU에서 제공하는 Frequency를 우리가 사용하기 쉬운 값으로 바꾸어 주는 것을 말합니다.
- Counter clock frequency 를 1~65536의 값으로 나누기 위해 16-bit programmable prescaler 사용
- period로 몇 번 count하는지 설정

4. *PWM*

- 일정한 주기 내에서 Duty ratio를 변화 시켜서 평균 전압을 제어하는 방법
- 대부분의 서브모터는 50Hz ~ 1000Hz의 주파수를 요구하고 데이터 시트를 반드시 확인해서 사용해야함.

실험 미션

● *세부 실험 내용*

1. TFT LCD에 team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼 생성
2. LED ON 버튼 터치 시 TIM2 interrupt를 활용하여 LED 2개 제어
   2.1 1초 마다 LED1 TOGGLE
   2.2 5초 마다 LED2 TOGGLE
3. LED OFF 버튼 터치 시 LED Toggle 동작 해제

실험 과정

1. 기본적으로 8주차에서 했던 내용과 유사하므로 LCD 라이브러리를 등록하고 8주차에 했던 LCD 라이브러리 코드를 그대로 가져온다.

2. main 코드도 8주차에서 했더 내용에서 수정을 하도록 한다.

2.1 RCC_Configure(void)

```
void RCC_Configure(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

: LED1, 2를 사용해야 하기때문에 port D를 활성화시켜 준다. timer 또한 2개를 사용해야 하기 때문에 general purpose timer인 TIM2와 TIM3을 활성화시켜준다.

2.2 GPIO_Configure(void)

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

    /* LED 1, 2 */
    /* PD2, 3     */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    /* TIM3 */
    /* PB0 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

}
```

: LED1,2 는 port D의 2,3번 핀을 사용하기 때문에 초기화시켜준다.
TIM3 또한 port B의 0번 핀을 사용하기로 했으므로 초기화시켜준다,

2.3 TIM_Configure(void)

```
void TIM_Configure(void) {
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  TIM_OCInitTypeDef TIM_OCInitStructure;

  int prescale = (uint16_t)(SystemCoreClock / 10000);

  TIM_TimeBaseStructure.TIM_Period = 10000;
  TIM_TimeBaseStructure.TIM_Prescaler = prescale;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 1500; // us
  TIM_OC3Init(TIM3, &TIM_OCInitStructure);

  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
  TIM_ARRPreloadConfig(TIM2, ENABLE);
  TIM_Cmd(TIM2, ENABLE);

  TIM_TimeBaseStructure.TIM_Period = 20000 - 1;
  TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) (SystemCoreClock / 1000000) - 1;

  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
  TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
  TIM_ARRPreloadConfig(TIM3, ENABLE);
  TIM_Cmd(TIM3, ENABLE);
}
```

: 라이브러리에서 설정된 timer clock frequency가 72000000가 맞는지 확인하고 ppt에서 제공해준 코드를 참고해서 함수를 정의한다. 우선 precale 값을 정의해줘야 하는데

$$\frac{1}{f_{clk}} \times prescaler \times period$$

$$\frac{1}{72Mhz} \times 7200 \times 10000 = 1[s]$$

분주계산식을 사용해서 prescale 값이 7200이 되게 하기 위해서 period 값을 10000으로 정의한다. 그리고 TIM_Pulse의 값을 1500으로 정의한다.

2.4 NVIC_Configure(void)

```
void NVIC_Configure(void) {

    NVIC_InitTypeDef NVIC_InitStructure;

    // TODO: fill the arg you want
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    // TIM2_IRQn
    // 'NVIC_EnableIRQ' is only required for USART setting
    NVIC_EnableIRQ(TIM2_IRQn);
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

}
```

: TIM2의 interrupt 우선순위를 1순위인 0으로 설정해 준다,

2.5 turn_on_off(), TIM2_IRQHandler()

```
int led2 = 0;
int cnt = 0;
uint16_t sub[2] = {1000, 2000};
int isOnOff = 0;
int flag = 0;

void turn_on_off(){
    if(led2 == 5) {
        led2 = 0;
        if(flag == 0){
            GPIO_SetBits(GPIOD, GPIO_Pin_3);
            flag = 1;
        }
        else{
            GPIO_ResetBits(GPIOD, GPIO_Pin_3);
            flag = 0;
        }
    }
}
```

```
void TIM2_IRQHandler() {
  if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET ) {
    if(isOnOff) {
      led2++;
      GPIO_SetBits(GPIOD, GPIO_Pin_2);
      turn_on_off();

      delay();

      led2++;
      GPIO_ResetBits(GPIOD, GPIO_Pin_2);
      turn_on_off();
    }

    change_pulse(sub[cnt++]);
    if (cnt == 2)
      cnt = 0;

    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
  }
}
```

: Tim2의 interrupt handler 함수인 TIM2_IRQHandler() 함수를 통해 LED1,2를 제어하기로 했다. main문의 무한루프문에서 LCD에 그린 네모 박스안에 터치할때마다 isOnOff 변수를 1과 0을 반복하게끔 코드를 짰다. 즉 1일 때는 ON, 0일 때는 OFF가 되도록 하게 했다. 그리고 여러개의 다른 interrupt가 같은 interrupt handler를 공유할 수 있기 때문에 interrupt handler가 호출된 것을 재확인 해주어야 하기 때문에 TIM_GetITStatus함수를 통해서 확인해 준다. 그리고 다시 interrupt를 발생시킬 수 있도록 초기화 해주어야 하기 때문에 TIM_ClearITPendingBit함수로 초기화시켜 주도록 한다. 타이머 인터럽트 함수에서 ON상태일 때 LED1이 1초마다 상태가 변하게 delay문 사이에 SetBit와 ResetBit함수를 넣었다. 그리고 LED2는 5초마다 상태가 변해야 하기 때문에 led2라는 변수를 정의하고 1초마다 1씩 증가하게끔 하고 1초마다 turn_on_off()함수를 호출해서 led2의 값이 5인지 확인하고 5일때마다 LED2의 상태를 변경시켜준다.

2.6 main()

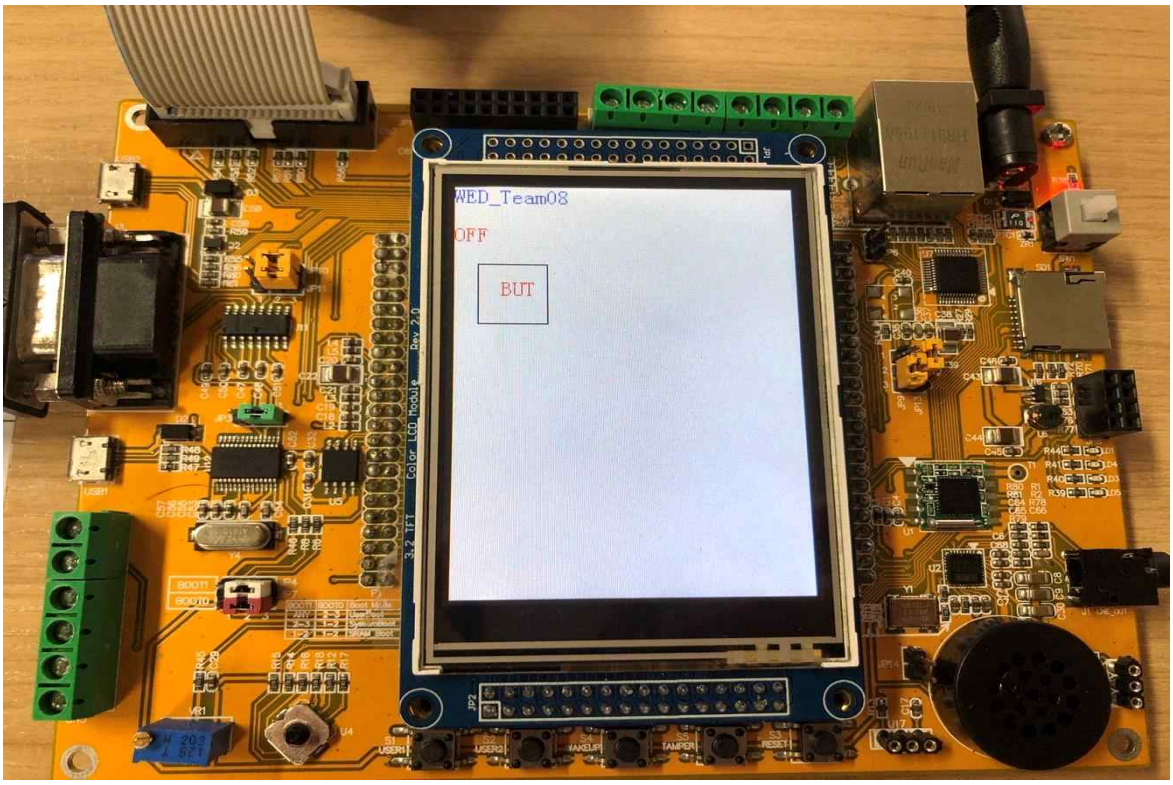
```
char msg1[] = "WED_Team08";
LCD_ShowString(0, 0, msg1, color[2], color[0] );

char msgOn[] = "ON ";
char msgOff[] = "OFF";

char msgBtn[] = "BUT";
LCD_DrawRectangle(16, 64, 64, 112);
LCD_ShowString(32, 76, msgBtn, color[3], color[0] );

uint16_t x, y;

while (1) {
  if(isOnOff) {
    LCD_ShowString(0, 32, msgOn, color[3], color[0] );
  }
  else {
    LCD_ShowString(0, 32, msgOff, color[3], color[0] );
  }

  Touch_GetXY(&x, &y, 1);
  Convert_Pos(x, y, &x, &y);
  if(x >= 16 && x <= 64 && y >= 64 && y <= 112) {
    isOnOff = !isOnOff;
    led2 = 0;
  }
}
return 0;
```

: ON/OFF를 변경 시킬 수 있도록 터치하는 부분을 만들기 위해서 LCD_DrawRectangle함수를 불러서 네모 박스를 만든다. 그리고 BUT라는 문장이 네모 박스에 들어갈 수 있도록 적절하게 X,Y좌표를 인자값으로 넣어준다. 그리고 isOnOff 라는 전역변수를 정의하고 네모 박스안을 터치할때마다 ON과 OFF 단어가 바뀔 수 있도록 if문을 걸어주었다. 그 후 Touch_GetXY, Conver_Pos 함수를 통해 터치하는 x,y 좌표를 얻어서 그 좌표가 네모 박스 안에 들어갈 때마다 isOnOff 값이 바뀌고, led2의 값을 0으로 초기화 해주도록 if문을 걸어주도록 한다.

초반에 led2가 5초동안 on상태가 유지되도록 요구하는지 몰라서 5초에 한 번씩 1초동안 켜지도록 코드를 짰었다. 문제에 대해 제대로 이해하지 못하고 코드를 짜는 상황이 벌어졌었다.

실험 결과

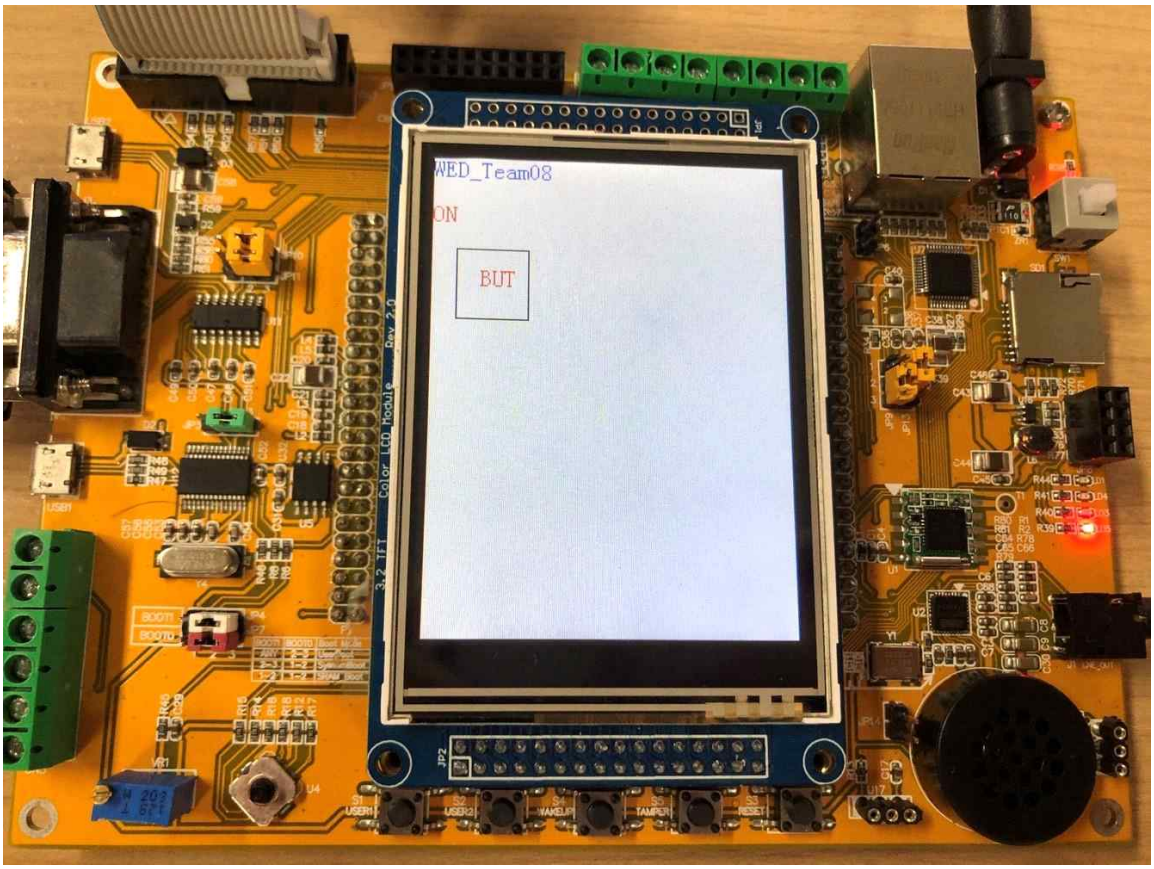
1. OFF일 때



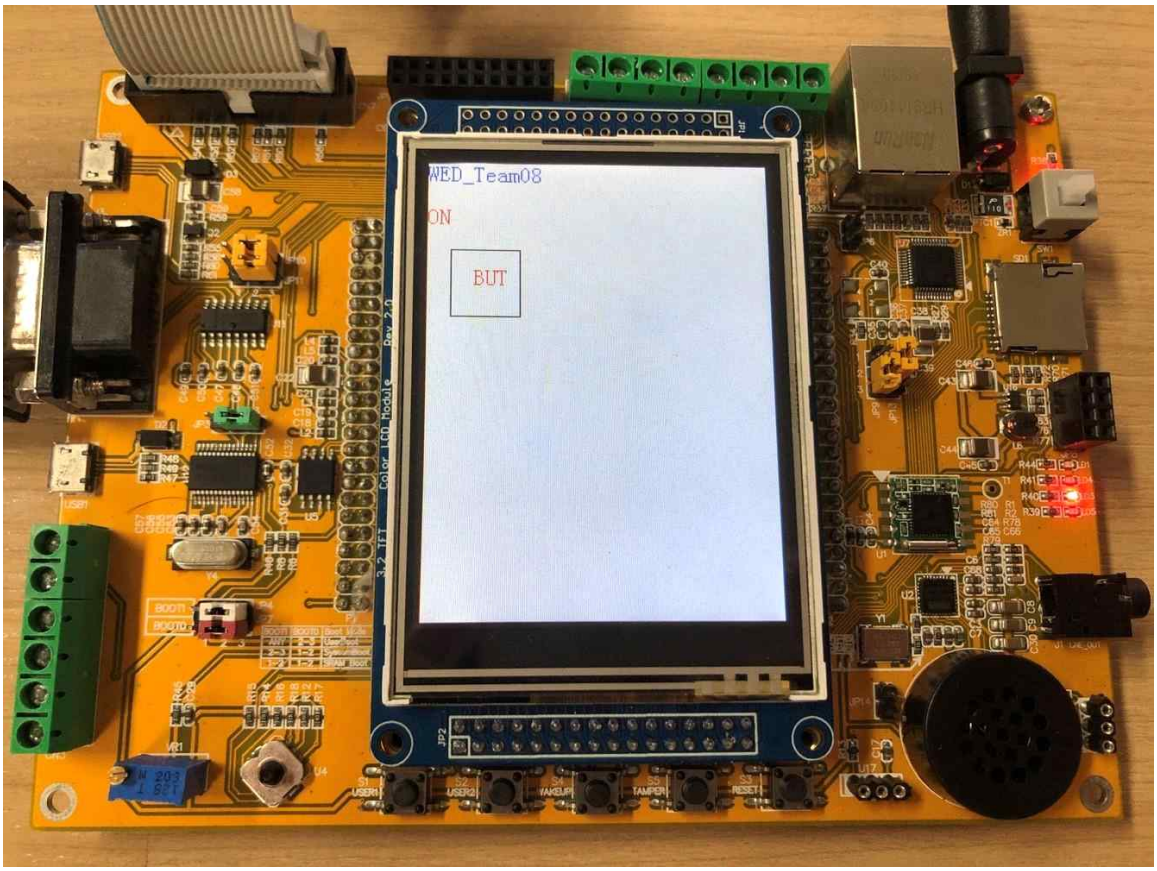
: LED1,2 둘 다 off되는 것을 확인할 수 있다

2. ON일 때

2.1 LED1 on, LED2 off



2.2 LED1 off, LED2 on



2.3 LED1, LED2 on