임베디드 시스템 설계 및 실험
11주차 실험 결과 보고서

조 : 8조

조원 : 서진욱, 이승민, 하연지, 하태훈

● 실험목표

1. **Timer의 기본적인 개념, 종류와 특징에 대해 이해하고 실습에 사용한다.**
2. **PWM에 대한 개념을 이해하고, 이를 활용하여 서보모터를 사용한다.**

● 개념설명

1. TFT-LCD(10주차 활용)

- 초 박막 액정 표시장치
- 액체와 고체의 중간 특성을 가진 액정의 상태 변화와 편광판의 편광 성질을 이용하여 통과하는 빛의 양을 조절함으로써 정보를 표시
- RGB 픽셀이 유리판에 코딩 되어 컬러 영상을 구현하는 Color Filter
- 액정을 제어하기 위해 초박형 유리 기판 위에 반도체 막을 형성한 회로인 TFT 기판
- Filter와 기판 사이에 주입된 액정과 광원인 Black light unit으로 구성

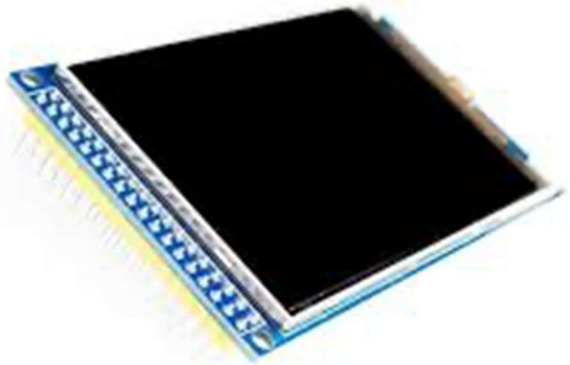

그림 1. TFT-LCD

2. Timer

- 주기적 시간 처리에 사용하는 디지털 카운터 회로 모듈
- 펄스폭 계측, 주기적인 interrupt 발생 등에 사용
- 주파수가 높기 때문에 우선 prescaler를 사용하여 주파수를 낮춘 후 낮아진 주파수로 8,16비트 등의 카운터 회로를 사용하여 주기를 얻음
- STM32 타이머 종류
  - SysTick Timer
  - Watchdog Timer
  - Advanced-control Timer (TIM1, TIM8)
  - **General-purpose Timer (TIM2 ~ TIM5) → 실습에 주 사용**
  - Basic Timer (TIM6, TIM7)

3. General-purpose timer

- Prescaler를 이용해 설정 가능한 16-bit up, down, up/down auto-reload counter를 포함
- 입력 신호의 펄스 길이 측정(input capture) 또는 출력 파형 발생(output compare and PWM) 등 다양한 용도로 사용
- 펄스 길이와 파형 주기는 timer prescaler와 the RCC clock controller prescaler를 사용하여 몇 μs에서 몇 ms까지 변조할 수 있음
  - 타이머들은 완전히 독립적이며, 어떤 자원도 공유하지 않으나 동기화 가능
- 분주
  - 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
  - period 로 몇 번 count하는지 설정

$$\frac{1}{f_{clk}} \times prescaler \times period \quad f_{clk} * \frac{1}{prescaler} * \frac{1}{period} = \text{주파수}[Hz]$$

그림 2. 분주 계산 관련식

4. PWM과 서보모터

- PWM
  - 일정한 주기 내에서 Duty ratio를 변화 시켜서 평균 전압을 제어하는 방법
    ▷ 0~5V의 전력 범위에서 2.5V 전압을 가하고 싶다면, 50% 듀티 사이클 적용
- 대부분의 서보모터는 50Hz ~ 1000Hz 의 주파수를 요구
  - 데이터 시트를 반드시 확인해서 사용할 필요가 있음

그림 3. SG90 서보모터

● 실험 진행

● 코드 작성

- RCC_Configure
  - 코드1은 LCD, PWM, LED의 RCC를 ENABLE하는 코드로 LCD를 연결할 Port C, TIM2와 TIM3, TIM3로 서보모터를 제어하기 위해 Port B, LED1과 LED2를 사용하기 위해 Port D를 활성화

```
void RCC_Configure(void) {
  /* LCD */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE); // Port C RCC ENABLE
  /* PWM */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); // TIM2
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); // Port B RCC ENABLE
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); // TIM3
  /* LED */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); // Port D RCC ENABLE
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

코드 1. RCC_Configure

- GPIO_Configure
  - 코드2는 GPIO 구성에 대한 코드로 LED1과 LED2를 사용하기 위해 Port D의 Pin2, Pin3을 ENABLE하고 각각 50MHz 속도의 Output Push-Pull 모드로 설정

- PWM은 TIM3로 동작하므로 Port B 의 Pin0를 ENABLE하고 50MHz 속도의 Alternate Output Push-Pull 모드로 설정

```
void GPIO_Configure(void) {
  GPIO_InitTypeDef GPIO_InitStructure3;
  GPIO_InitStructure3.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3; // LED1, LED2 ENABLE
  GPIO_InitStructure3.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure3.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure3);

  /* PWM */
  GPIO_InitTypeDef GPIO_InitStructure2;
  GPIO_InitStructure2.GPIO_Pin = GPIO_Pin_0; // TIM3 Pin
  GPIO_InitStructure2.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure2.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOB, &GPIO_InitStructure2);
}
```

코드 2. GPIO_Configure

- NVIC_Configure
  - 코드3은 TIM2에 의한 Interrupt 제어를 위해 NVIC를 설정하는 코드로 PriorityGroup은 0으로 설정하고 PreemptionPriority와 SubPriority를 모두 0으로 설정

```
void NVIC_Configure(void) {
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
  NVIC_InitTypeDef NVIC_InitStructure;
  /* Enable TIM2 Global Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

코드 3. NVIC_Configure

- TIM2_IRQHandler
  - 코드 4, 5는 TIM2의 주기 1초마다 Interrupt가 발생할 때 동작을 설정하는 코드로 Button을 통해 Mode가 ON일 때와 OFF일 때 동작을 설정
  - ON Mode일 때 LED1은 1초마다 껐다 켜지는 동작을, LED2는 5초마다 껐다 켜지는 동작을 반복하고 서보모터는 +90° 방향으로 회전

```
void TIM2_IRQHandler(void) {
  if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
    /* Mode ON */
    if (flag == 0){
      if (ledCount % 2 == 0) {
        GPIO_SetBits(GPIOD, GPIO_Pin_2);
      }
      else {
        GPIO_ResetBits(GPIOD, GPIO_Pin_2);
      }

      if(ledCount == 0) {
        GPIO_ResetBits(GPIOD, GPIO_Pin_3);
      }
      else if(ledCount == 5) {
        GPIO_SetBits(GPIOD, GPIO_Pin_3);
      }

      ledCount++;
      ledCount %= 10;
      k += 100;
      if (k == 2300) {
        k = 700;
      }

      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 = k ; // us
      TIM_OC3Init(TIM3, &TIM_OCInitStructure);
    }
```

코드 4. TIM2_IRQHandler : Mode ON

- OFF Mode일 때 LED1, 2는 Toggle을 멈추고 서보모터는 -90° 방향으로 회전

```
    /* Mode OFF */
    else {
      k -= 100;
      if (k == 700){
        k = 2300;
      }

      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 = k ; // us
      TIM_OC3Init(TIM3, &TIM_OCInitStructure);
    }
    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
  }
}
```

코드 5. TIM2_IRQHandler : Mode OFF

- PWM_Configure1
  - 코드 6은 LED의 PWM(Pulse Width Modulation)을 설정하는 코드로 그림3의 분주 계산 관련식을 참고해 시스템 클럭 주파수를 10000으로 나눠 prescale을 계산하고 주기를 설정

- 클럭 분할은 0으로 설정해 분할하지 않고 클럭을 그대로 사용해 타이머를 동작
- 타이머 카운트모드를 down으로 설정해 타이머 값이 감소해 0에 도달하면 다시 설정한 주기로 초기화되고 이 과정을 반복
- TIM2 타이머를 사용해 PWM 설정

```
void PWM_Configure1()
{
    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_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    TIM_ARRPreloadConfig(TIM2, ENABLE);
    TIM_Cmd(TIM2, ENABLE);
}
```

코드 6. PWM_Configure1

- PWM_Configure2
  - 코드 7은 서보모터의 PWM(Pulse Width Modulation)을 설정하는 코드로 그림3의 분주 계산 관련식을 참고해 시스템 클럭 주파수를 1000000으로 나눠 prescale을 계산하고 주기를 설정
  - TIM_Pulse(PWM출력 채널의 펄스 폭)를 1500으로 줘 서보모터의 초기값을 설정하고 이를 기준으로 서보모터의 움직임을 조절하도록 함
  - TIM3 타이머를 사용해 PWM 설정

```
void PWM_Configure2()
{
    prescale = (uint16_t) (SystemCoreClock / 1000000);
    TIM_TimeBaseStructure.TIM_Period = 20000;
    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(TIM3, &TIM_TimeBaseStructure);
    TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);
    TIM_ARRPreloadConfig(TIM3, ENABLE);
    TIM_Cmd(TIM3, ENABLE);
}
```

코드 7. PWM_Configure2

- delay
  - delay 를 만드는 함수로 반복문을 사용해 시간을 지연

```
void delay()
{
     for(int j = 0; j < 5000000; j++) {
        continue;
     }
}
```

코드 8. Delay

- main 내 while
  - Touch_GetXY 함수로 터치 좌표를 받아 배열에 입력하고 Convert_Pos로 받은 터치 좌표를 LCD의 크기에 맞게 변환
  - 특정 좌표 범위에 따라 LCD에 메세지가 출력되고 flag 를 통해 ON/OFF를 표시
  - TIM2 Interrupt 활성화
  - Flag의 값은 0이나1로 유지되며 1일때는(터치 입력 전 OFF 상태) ON 문자열 표시되며 0일때는(터치 입력 전 ON 상태) OFF 문자열이 표시됨

```
while (1) {
  Touch_GetXY(&pos_temp[0], &pos_temp[1], 1); // 터치 좌표 받아서 배열에 입력
  Convert_Pos(pos_temp[0],pos_temp[1],&pos_temp[0],&pos_temp[1]); // 받은 좌표를 LCD 크기에 맞게 변환
  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); // interrupt ENABLE

  /* If Button Touch event Occured*/
  if ((uint16_t)40 < pos_temp[0] && pos_temp[0] < (uint16_t)80 && \
      (uint16_t)80 < pos_temp[1] && pos_temp[1] < (uint16_t)120) {
    /* If current Mode is OFF Change Mode to ON */
    if (flag) {
      LCD_ShowString(90, 50, "    ", RED, WHITE);
      LCD_ShowString(40, 50, "ON", RED, WHITE);
    }

    /* If current Mode is ON Change Mode to OFF */
    else {
      LCD_ShowString(90, 50, "OFF", RED, WHITE);
      LCD_ShowString(40, 50, "    ", RED, WHITE);
    }

    flag++;
    flag %= 2;
  }
  delay();
}
return 0;

}
```

코드 9. 터치입력 처리 및 LCD 제어

● 결과 및 결론

● 실험 결과

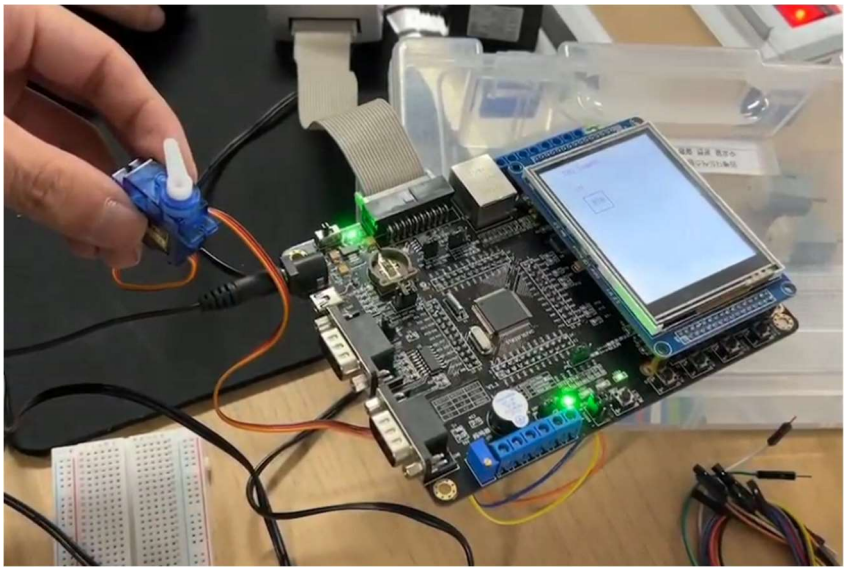

그림 5. 동작 초기 상태

- 초기 상태(LED off, 화면에 조 이름 출력 및 버튼off 상태로 출력.
- 서보모터 중앙에서 정지)에서 LCD에 출력된 버튼의 좌표에 터치가 감지되면 화면을 리프레시 하고, 버튼의 상태를 ON으로 출력한다.
- 1번 LED는 1초, 2번 LED는 5초마다 on/off되고, 서보모터가 +90°방향으로 1초마다 pulse값 100만큼 이동한다.
- 서보모터가 더이상 이동할 수 없는 위치에 도달하면 반대쪽 끝으로 이동 후 다시 1초마다 +90°방향으로 이동한다

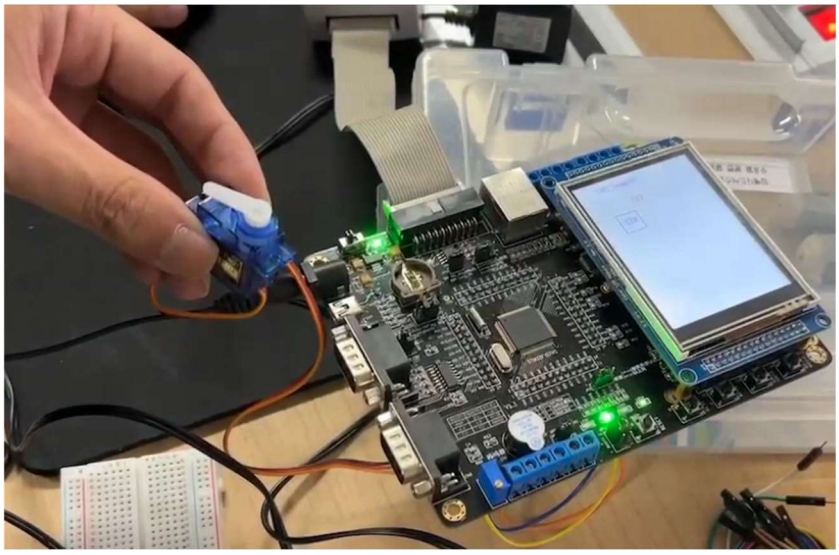

그림 6. 동작 상황(On → OFF)

- 버튼 ON 상태에서 다시 한번 버튼에 터치가 감지되면 LCD 화면을 리프레시 하여 버튼을 OFF 상태로 출력하고, LED의 토글을 멈추며, 서보모터가 반대방향으로 1초에 pulse값 100만큼 이동한다.

- 반대방향 역시 서보모터가 더 이상 움직일 수 없는 위치에 도달하면 반대쪽 끝으로 이동 후 다시 1초마다 -90°방향으로 이동한다.

- **결론**
  - Timer를 통해 interrupt를 발생시켜 보드를 제어할 수 있음을 알게 되었다.
  - Timer를 통해 단순히 delay 함수로 반복문을 반복시켜 타이밍을 제어하는 것보다 정확한 타이밍 제어가 가능함을 알게 되었다.
  - PWM을 설정하여 pulse 값을 변경하여 서보모터의 제어가 가능함을 알게 되었다.