임베디드 시스템

실험 및 설계

10주차 실험 보고서

2조 노윤정, 박건우, 이동근, 조영진

1. 실험목표

a) 라이브러리를 활용해 코드를 작성

b) Clock Tree 의 이해 및 사용자 Clock 설정

c) UART 통신의 원리를 배우고 실제 설정 방법 파악

2. 실험미션

a) 라이브러리를 사용해 Main.c 파일 작성

b) TFT LCD 에 Team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼 생성

c) LED 버튼 터치시 TIM2 intterupt 활용해 LED 2개 제어

- 1초마다 LED1 Toggle

- 5초마다 LED2 Toggle

d) LED 버튼 다시 터치시 LED Toggle 동작 해제

3. 주의사항

a) 기기 작동중 케이블 뽑기 금지

b) 실험 후 자리 및 PC 깔끔히 뒷정리

4. 실험과정

이번 실험에서는 하드웨어 구성하지 않고, 소프트웨어만 구성하면 되기에, 해당하는 코드를 작성하는것이 주된 목표이다. 따라서 우리팀은 main.c 파일만을 이용해 실험을 진행했으며 해당 문단에서는 이에대해 설명하도록 하겠다.

1) Configure

a) RCC Configure

```
void RCC_Configure(void) {
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); // Enable LED
    // Enable the TIM2 clock
    RCC_APB1PeriphClockCmd(RCC_APB1ENR_TIM2EN, ENABLE);
}
```

TIM2 를 비롯해 LED 와 기타 클럭을 인가한다. 이때 TIM2 는 APB1 PeriphClock 을 사용하므로 LED를 Enable 시키는 함수와는 다른 함수인 RCC_APB1PeriphClockCmd 를 사용해야한다.

b) GPIO Configure

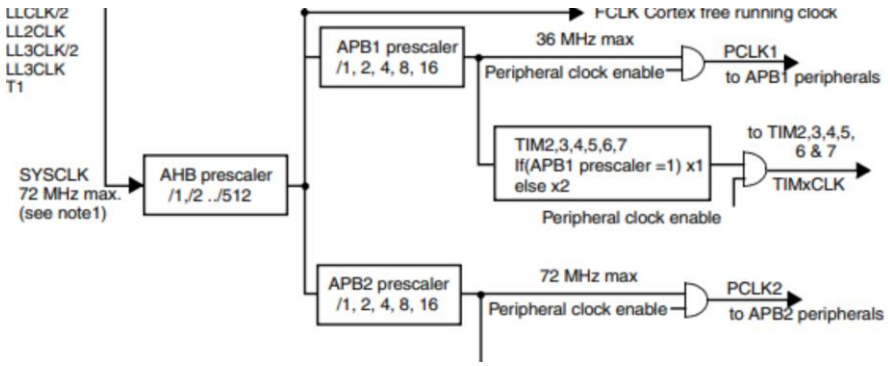
```
void GPIO_Configure(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    /* LED pin setting*/
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
}
```

사용할 LED인 1,4번 LED를 Enable 시켜준다.

c) TIM Configure

우리팀은 TIM2를 이용해 Interrupt 를 사용하기로 논의했고, 이를 위해서 적절한 prescaler 값과 period 값을 얻어야 했다.



실험제공 자료로 주어진 해당 clock tree를 살펴본다면 AHB의 prescaler 값이나 APB1 prescaler 값을 따로 조정하지 않았다면 TIM2CLK로 SYSCLK가 그대로 나가는 것을 확인할 수 있다.

```
/* #define SYSCLK_FREQ_HSE    HSE_VALUE */
/* #define SYSCLK_FREQ_24MHz  24000000 */
/* #define SYSCLK_FREQ_36MHz  36000000 */
/* #define SYSCLK_FREQ_48MHz  48000000 */
/* #define SYSCLK_FREQ_56MHz  56000000 */
#define SYSCLK_FREQ_72MHz  72000000
#endif
```

System_stm32f10x.c 파일에서 Sysclk 가 72MHz로 설정된 것을 확인할 수 있다. 따라서 TIM2CLK = SYSCLK = 72MHz 이다.

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

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

마찬가지로 강의자료의 해당 수식을 사용하면 count의 주기를 계산 할 수 있는데, 현재 f_clk = 72MHz 임을 확인했으므로 위의 prescaler 와 period 값을 각각 7200, 10000 으로 설정하게 된다면 counter 는 1초마다 인터럽트를 발생시키게 된다.

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

    TIM_TimeBaseStructure.TIM_Period = 10000;
    TIM_TimeBaseStructure.TIM_Prescaler = 7200;
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

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

Prescaler 와 period 를 구했다면 이제 TIM_TimeBaseStructure 를 이용해 타이머를 구동시키면 된다. 이때 구해둔 period 와 prescaler 값을 기입하고, clockDivision 을 1로, counterMode 를 increase 로 설정해 주었다. 이후 설정을 토대로 TIM2를 실행시켜 주었으며, TIM_IT_Update는 타이머가 인터럽트를 일으킬 때(1초가 경과되었을때) 마다 업데이트 함을 뜻한다.

d) NVIC Configure

```
void NVIC_Configure(void) {
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn; // TIM2 interrupt
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

타이머의 인터럽트를 NVIC Configure 로 설정한다. 사용되는 인터럽트는 터치와 타이머 2가지 종류이나, 터치 인터럽트는 Main 함수에서 구현하므로 NVIC는 한가지의 인터럽트만 설정하였다.1가지밖에 없으므로 우선순위 또한 아무렇게나 두어도 상관이 없다. 설정을 마친후 NVIC_Init 함수를 이용해 활성화 시킨다.

e) Init Configure

```
105  void Init_Configure(void){
106      SystemInit();
107      RCC_Configure();
108      GPIO_Configure();
109      TIM_Configure();S
110      // EXTI_Configure();
111      NVIC_Configure();
112      LCD_Init();
113      Touch_Configuration();
114  }
```

앞서 기술한 함수들 말고도 SystemInit 함수나 LCD, Touch 함수와 같은 기본 설정 함수들을 동작시키는 함수를 작성했다.

2) Interrupt

구현해야할 인터럽트의 종류는 2가지로 첫번째는 타이머 인터럽트이고 두번째는 터치 인터럽트이다. 우선 1초마다 LED1이, 4초마다 LED2가 Toggle 된다고 했으므로 1초마다 값이 증가하는 변수가 정의 되어야 할것이다. 이를 아래와 같은 전역변수로 설정하였다.

```
14   uint16_t count = 0;
```

이와 같은 count 가 1초마다, 즉 TIM2가 인터럽트를 발생시킬 때 마다 증가해야 하므로 다음과 같이 핸들러 함수를 구현하였다.

```
93  void TIM2_IRQHandler(void) {
94      // TIM2_IRQn;
95      /* ITStatus */
96      if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
97          count++;    // count is incremented;
98          TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
99      }
100 }
```

타이머의 인터럽트가 발생했을시 작동되는 핸들러 함수이다. 인터럽트 발생시 (TIM2 TIM_IT_Update가 SET 상태일시) 전역변수인 count 를 증가시키고 TIM_IT_Update 의 값을 다시 RESET으로 만들어준다.

```
if(!T_INT){                    // LCD touch interrupt
    doToggling = !doToggling;  // toggle switch
    count = 0; }
```

터치 인터럽트는 Main 함수 내에서 구현했는데, 해당 부분만을 살펴보자면, LCD의 터치 인터럽트 값을 저장하는 T_INT를 수시로 확인하여 인터럽트가 발생했다면 Toggle 의 상태를 ON 또는 OFF 로 전환하며, Count 값을 초기화 시킨다.

3) Main

```
129  int main(void) {
130      Init_Configure();
131
132      LCD_ShowString(30, 30, "MON_Team02", BLUE, WHITE);  // team
133      LCD_ShowString(45, 105, "BUT", RED, WHITE);  // Button
134      LCD_DrawRectangle(30, 90, 60, 120);
```

앞에서 기술한 Init_Configure 함수를 작동 시키고, LCD 화면에 "MON_Team02"와 버튼을 출력한다.

```
141      char** state = ["OFF","ON"];
142      while (1) {
143          if(!T_INT){                    // LCD touch interrupt
144              doToggling = !doToggling;  // toggle switch
145              count = 0; }
146
147          if(count != precount && doToggling == ON)
148              Set_LED(GPIO_Pin_2,LED1_state);
149
150          if(!(count % 5) && doToggling == ON)
151              Set_LED(GPIO_Pin_7,LED4_state);
152
153          LCD_ShowString(30, 60, state[doToggling],RED,WHITE);
154          precount = count;
155      }
```

143 ~ 145 Line 은 위의 Interrupt 문단에서 설명했으므로 생략하도록 한다. 147-8 Line은 count 값이 precount (이전의 count값) 과 다르고, Toggle 이 ON 된 상태라면 LED1을 ON 시킨다. 타이머 인터럽트가 발생하지 않는다면 항상 precount와 count 값이 같을것이고, 인터럽트가 발생한 순간부터 1초까지는 precount와 count 값이 같을것이기 때문에 이렇게 한다면 1초마다 Toggle 하는 기능을 구현할 수 있다. 150-1 Line은 count 의 값이 5의 배수일 때 마다 작동한다. 5의 배수일때마다 LED4를 Toggle 하므로 실험 조건을 만족한다.

위의 Toggling 이 끝난 후, LCD에는 ON/OFF의 상태를 표시하며 precount 역시 count 값으로 갱신한다.

6. 결론 및 제언

처음엔 Touch Interrupt 를 EXTI 를 이용해 구현하려 하였으나 예상대로 되지 않았다. 때문에 그냥 T_INT 값을 main 에서 인식하는 방안으로 구현하였는데 해당방법은 loop 뒤에 delay 가 있다면 제대로 작동하지 않을것이기 때문에 불안정한 방법이다. 이를 제대로 EXTI로 구현해야 훌륭한 프로그램이 될 것이다.

타이머 인터럽트를 구현할 때, 함수가 실행되었을때 count 값을 증가시키기만 하고 함수를 종료시켰더니 count 값이 1초에 수천, 수만번 증가하는 것을 볼 수 있었다. 이는 타이머의 IT 값을 RESET으로 해주지 않았기 때문에 발생하는 문제였다. IT 값을 Clear 해주는 함수를 추가시키니 이와 같은 오류는 해결되었다.

RCC Configure 을 통해서 TIM2에 클럭을 인가해야 됐는데 여기서 RCC_APB2PeriphClockCmd 함수를 이용해서 TIM2 를 Enable 시키려했다. 하지만 TIM2는 APB1 Clock 을 사용하므로 해당 함수로는 클럭이 인가될 수 없었다. 때문에 타이머가 작동하지 않았고 이 때문에 한시간을 골머리를 싸맸었다. 이를 RCC_APB1PeriphClockCmd를 이용하는 코드로 올바르게 고치니 TIM2가 올바르게 동작하였다.

위의 두 문제는 모두 레퍼런스를 제대로 읽지 않아서 발생한 것이다. 심지어 라이브러리의 파일들만 읽어도 해결되는 문제들인데 이를 디버깅으로 해결하려 했다가 괜히 시간만 버리기 일쑤였다. 첫 단추부터 잘 꿰야 결과도 잘나온다는 말이 있듯, 레퍼런스를 잘 보고 처음부터 근거있는 코드를 짜는 것이 중요하고 무언가 잘못되었다면 레퍼런스를 참고하는 습관을 들여야 될 듯 하다.