# 임베디드 시스템 설계 및 실험

## 11주차

7조
201424470 서민영
201424421 김시은
201424533 정종진
201424532 정재광

# 목차

## 1. 실험목표

1) Timer의 원리와 동작을 이해한다.
2) Interrupt를 통하여 LED를 제어한다.
3) 오실로스코프의 트리거 기능을 익힌다.

## 2. 배경지식

1) Timer

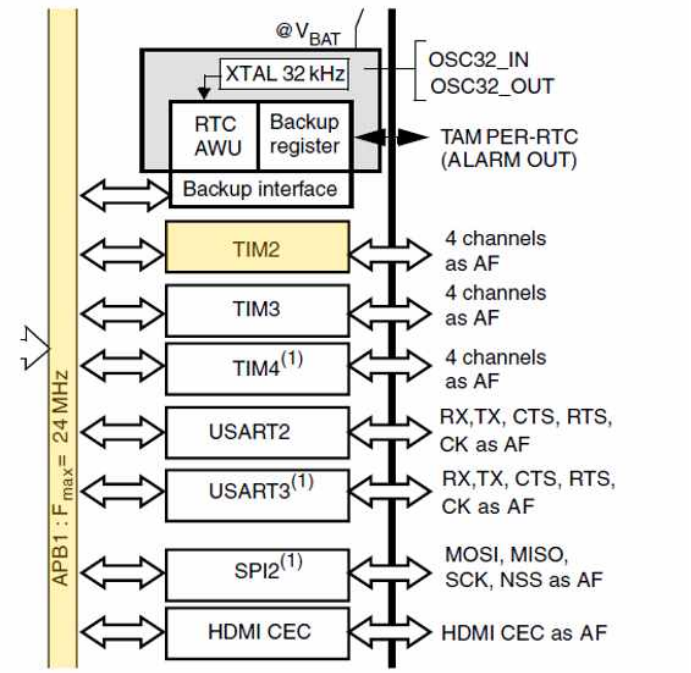
① 타이머의 종류

- TIM&TIM8: Advanced-control timers
- TIM2~TIM5, TIM9~TIM14: General purpose timers
- TIM6&TIM7: Basic timers

② Mode의 종류

- Up mode: 0부터 period까지 숫자를 count
- Down mode: period부터 0까지 숫자를 count
- Auto-reload: period까지 도달 후 다시 0으로 돌아가 반복

③ 클럭 버스와 연결

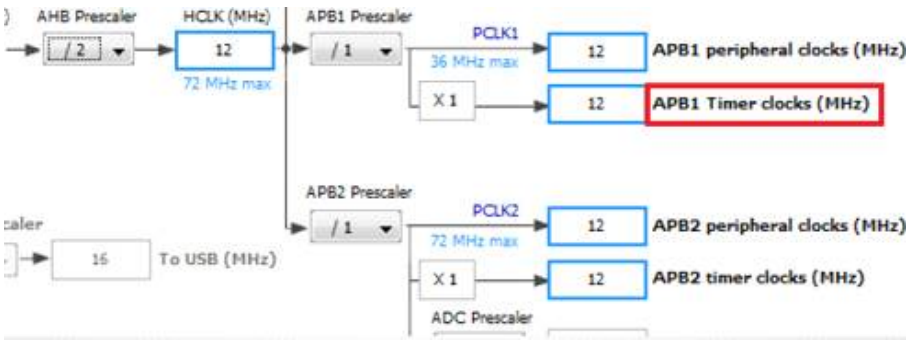



< 그림1 : TIM과 클럭 버스 연결 >

- TIM2,3,4는 APB1클럭 버스에 연결되어 있다.

2) Clock

① Clock 생성




< 그림2 : Clock발생 과정 >

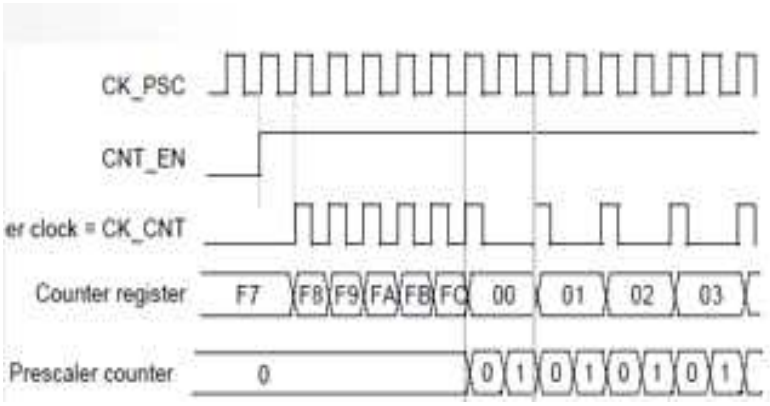
- 12MHz의 경우 1초에 12,000,000번 클럭 발생
- prescalar를 이용하여 사용하기 쉬운값으로 조절가능

② 분주

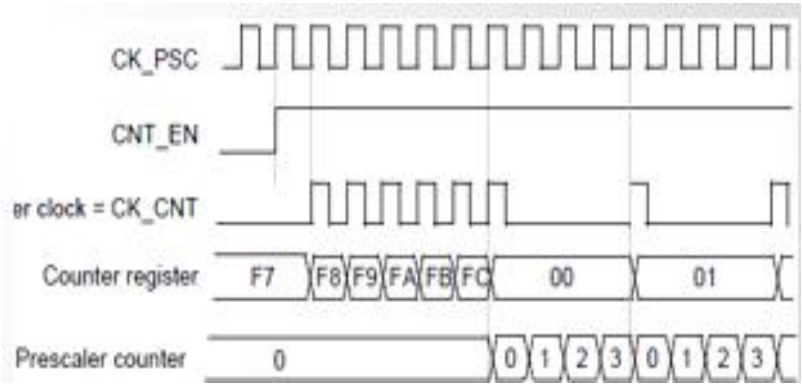
- 분주란 MCU에서 제공하는 Frequency를 우리가 사용하기 쉬운 값으로 바꾸어 주는 것

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

- prescalar에 따른 분주




< 그림3 : prescalar가 1일 때 >




< 그림4 : prescalar가 3일 때 >

3) 오실로스코프


< 그림5 : 오실로스코프 >

① 포지션: 신호를 화면상의 어디에 놓을 것인지에 관한 것
② 스케일: 파형의 크기를 조절
- 수직 스케일은 높이(진폭)
- 수평 스케일은 길이(파장)
③ 트리거: 시작지점을 조절(라이징엣지/폴링엣지)

## 3. 실험과정

1) 보드 연결

<그림 6 : Coretex M3/JTAG/DSTREAM을 연결한 모습>

1-1) Coretex M3/JTAG/DSTREAM 연결

다음과 같은 순서로 보드를 연결한다. 이 때 연결 및 분리 순서를 제대로 지키지 않으면 장비가 망가질 수 있으니 주의해야한다.

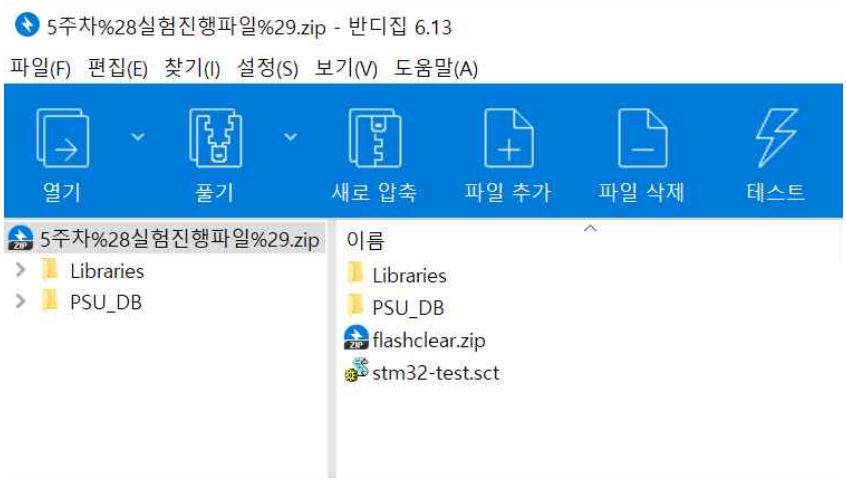
| |
|---|
| ① 보드와 DSTREAM JTAG 연결 |
| ② 보드 전원선만 연결 |
| (보드의 전원은 OFF 상태) |
| ③ DSTREAM 전원 연결 및 ON |
| ④ DSTREAM Status LED 점등 확인 |
| ⑤ 보드 전원 ON |
| ⑥ DSTREAM Target LED 점등 확인 |
| ⑦ DS-5에서 'connect target' |

<표 1 : 보드 연결 순서>

2) DS-5 디바이스 데이터베이스 추가

2-1) 수업게시판에서 실습파일로 제공되는 PSU_DB, LIBRARIES, SCATTER FILE을 다운

2-2) 이번 실험에서 include하는 코드의 양이 커서 기존에 scatter file에 설정된 값을 사용하면 load되지 않기 때문에 Data Sheet를 참고하여 scatter file의 메모리 범위를 수정해야 한다.




<그림 7 : 실습파일로 제공된 파일들>

```
LR_IROM1 0x08000000 0x00008000 {    : load region size_region
  ER_IROM1 0x08000000 0x00008000  {  : load address = execution address
   *.o(RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00008000  {  : RW data
   .ANY (+RW +ZI)
  }
}
```

Data Sheet를 참고하여 0x00080000 으로 수정함

<그림 8 : 스캐터 파일 수정>

2-3) Eclipse에 데이터베이스 추가

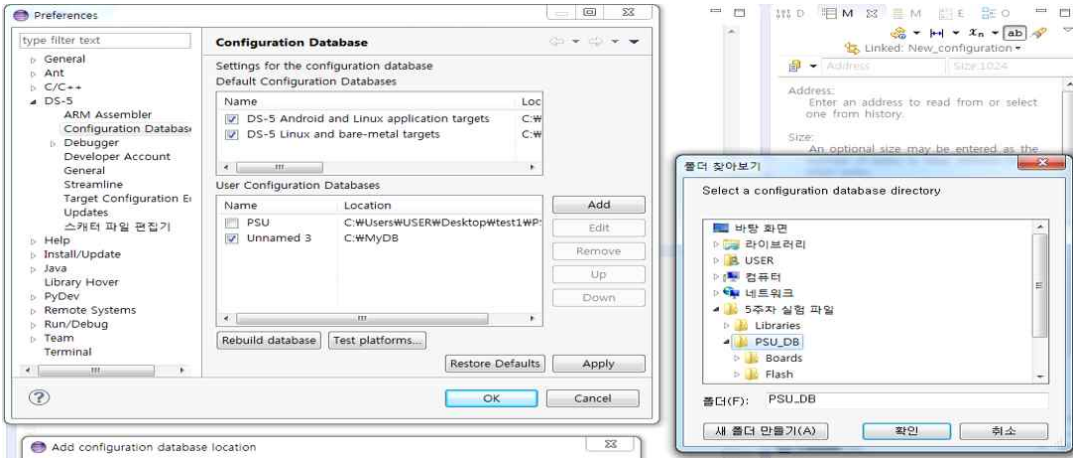
① 시작 → 모든 프로그램 → ARM DS-5 → Eclipse for DS-5 메뉴를 선택

② Windows → Preferences

③ DS-5 → Configuration Database 항목을 선택

④ Add 버튼을 클릭하여 사용자 데이터베이스의 디렉터리를 지정

⑤ Rebuild database 버튼을 클릭하여 데이터베이스 추가를 완료




<그림 9 : Eclipse에 데이터베이스 추가>

3) C Project 생성 및 환경설정

3-1) C Project 생성

① New Project → C project → Excutable → Empty Project를 선택해주고 Toolchains는 ARM Cmpiler 5(DS-5 built in)로 선택 후 프로젝트 생성

3-2) C Project Properties 설정

① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → ARM Linker 5 → Image Layout → Scatter file 설정

(Scatter file에서 RO/RW base address를 지정해주므로 설정해줄 필요가 없음)

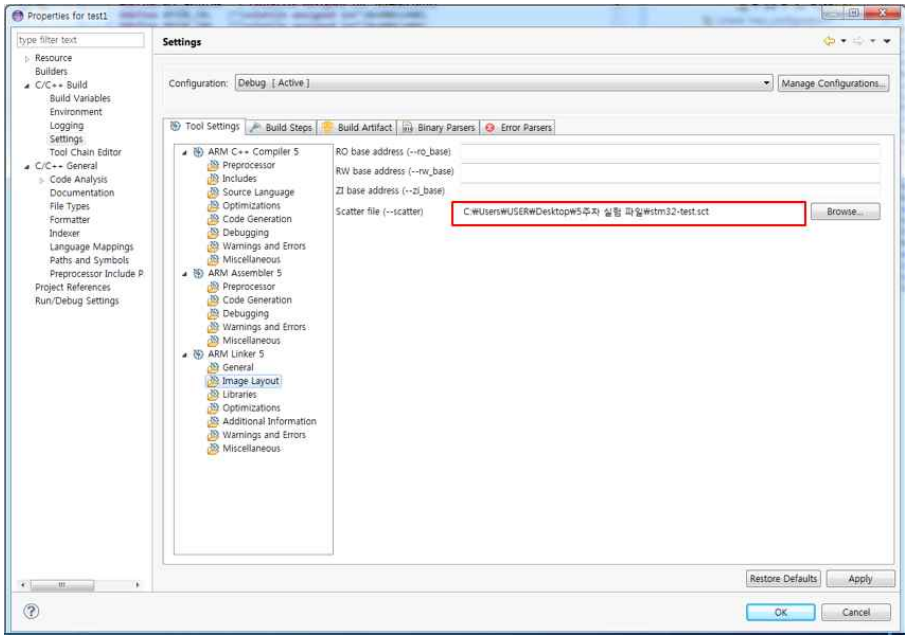
② C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Code Generation과 General의 Target CPU를 Cortex M3로 설정

(entry point를 main으로 지정해주지 않아도 됨)

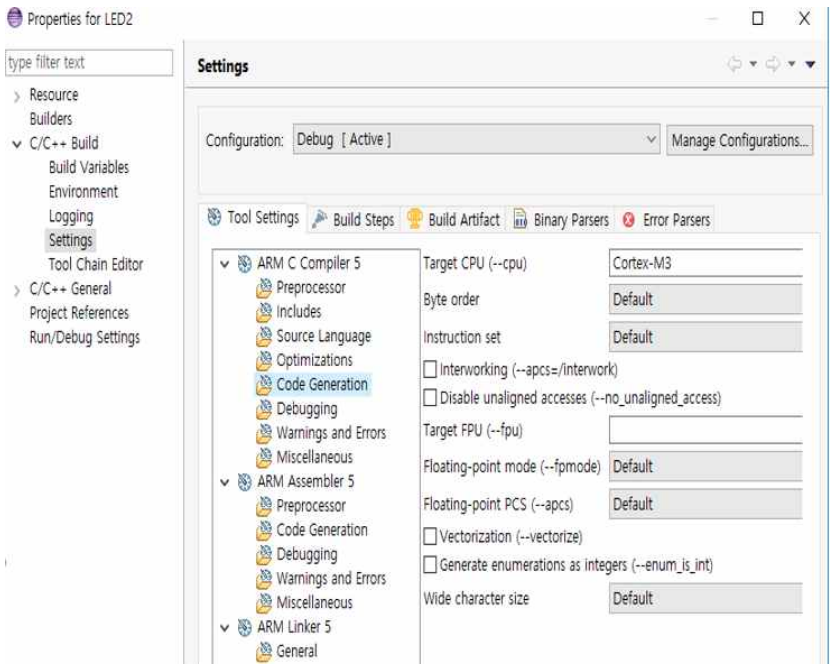
③ C project 우클릭 후 Properties 선택 → C/C++ Build → Settings -> Optimization에서 Optimization level을 High로 설정 (High로 설정 시 컴파일 할 때 코드의 크기가 작아짐)

3-3) LIBRARIES 추가

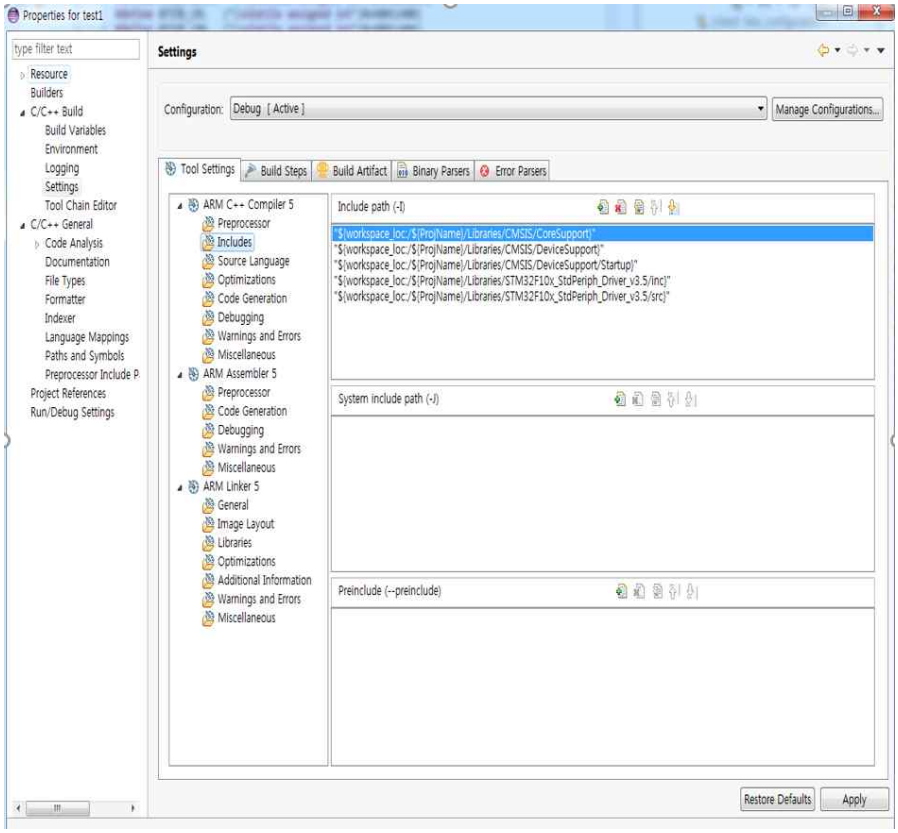
① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Includes에서 제공되는 LIBRARIES 추가




<그림 10 : Scatter file 설정>




<그림 11 : Target CPU 설정>




<그림 12 : 제공되는 LIBRARIES 추가>

4) DS-5 Debugger 연결

4-1) Debug Configuration 설정

① Run → Debug Configuration 메뉴를 선택

② DS-5 Debugger 더블 클릭하여 새로운 하위 오브젝트 생성

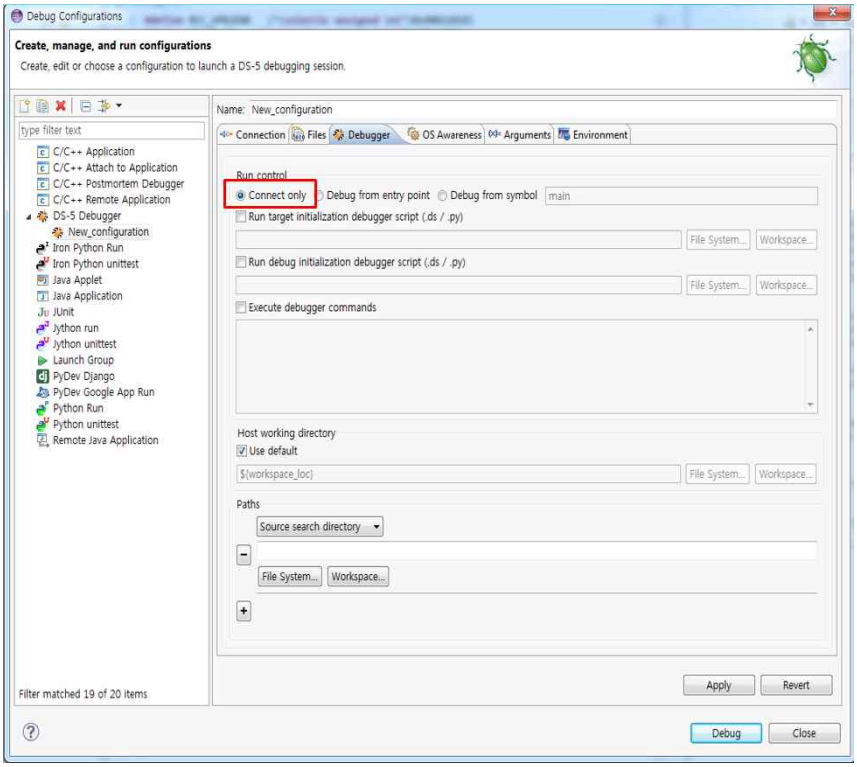
③ Name, Platform 등 Debug 환경설정을 변경

④ Browse 버튼을 클릭하여 DSTREAM 장비를 detection

4-2) Debug

① Debugger 탭에서 Connect only 체크

② Apply 클릭




<그림 13 : Debug Configurations에서 Debugger탭의 설정화면>

5) 세부 실험 내용

5-1) 실험 요구 조건 및 구현 내용

① TIMER 2를 사용하여 10ms 마다 LED 1번 깜빡거리도록 할 것
(10ms high, 10ms low)

② 오차범위는 마이크로(us) 단위까지 허용

③ LED 신호 오실로스코프로 확인하여 정확한 타이밍 출력되는지 확인

④ 6주차 CLOCK 설정 활용하여 TIMER 2로 인가되는 CLOCK 설정

⑤ LCD 화면에 분, 초, 1/10초, 1/100초 단위 표시

⑥ Start, Stop, Reset 버튼 LCD에 구현 및 이용하여 동작

5-2) C source code 작성

① TIM에 대하여 설정하기

② TIM Interrupt에 대하여 설정하기

③ TIM Interrupt의 Handler 구현하기

7) C project 빌드 및 .axf 업로드

7-1) C project 빌드 및 Debug As

① C Project 빌드
(.axf 파일이 생성된 것을 확인할 수 있음)

② C Project 우클릭 → Debug → Debug As

7-2) flashclear.axf 및 생성된 .axf 업로드

① command 라인에 다음과 같은 명령어를 입력해서 flashclear.axf를 업로드

flash load “flashclear.axf 파일경로”

② disconnect한 다음 보드를 껐다가 켬

③ command 라인에 다음과 같은 명령어를 입력해서 생성된 .axf를 업로드
flash load “생성된 .axf 파일경로”

④ disconnect한 다음 보드를 껐다가 켬

(flash load 후에는 반드시 diconnect를 하고 보드를 껐다가 켜야 함)

7-3) LCD에 나타나는 분, 초, 1/10초, 1/100초 확인

7-4) 오실로스코프에 나타나는 LED 신호 주기 확인

① 조도센서에 빛을 비추면서 저항값의 변화를 관찰.

② 제대로 동작하지 않으면 5) C source code 작성으로 돌아감

8) 보드 연결 해체

앞서 보드 연결과 마찬가지로, 보드 연결 해체 시에도 순서를 제대로 지키지 않으면 보드가 망가질 수 있으므로 유의해야한다. 보드 연결 해체 순서는 다음과 같다.

| |
|---|
| ① DS-5에서 ‘disconnect target’<br>② 보드 전원 OFF<br>③ DSTREAM 전원 해제 및 OFF<br>④ 보드 전원선 분리<br>⑤ DSTREAM과 보드 JTAG 분리 |

<표 2 : 보드 연결 해체 순서>

## 4. 작성한 소스코드 및 실험결과

1) 작성한 소스코드

team07.c

```
// flash load "C:\Users\Team07\week11\team07\flashclear\flashclear.axf"
// flash load "C:\Users\Team07\week11\team07\Debug\team07.axf"

#include "stm32f10x.h"
#include "core_cm3.h"
#include "misc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_tim.h"
#include "lcd.h"
#include "touch.h"

int t0=0;
int flag = 0;

void SysInit(void) {
  /* Set HSION bit */
  /* Internal Clock Enable */
  RCC->CR |= (uint32_t)0x00000001; //HSION

                          /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
  RCC->CFGR &= (uint32_t)0xF0FF0000;
```

```
    /* Reset HSEON, CSSON and PLLON bits */
    RCC->CR &= (uint32_t)0xFEF6FFFF;

    /* Reset HSEBYP bit */
    RCC->CR &= (uint32_t)0xFFFBFFFF;

    /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
    RCC->CFGR &= (uint32_t)0xFF80FFFF;

    /* Reset PLL2ON and PLL3ON bits */
    RCC->CR &= (uint32_t)0xEBFFFFFF;

    /* Disable all interrupts and clear pending bits  */
    RCC->CIR = 0x00FF0000;

    /* Reset CFGR2 register */
    RCC->CFGR2 = 0x00000000;
}

void SetSysClock(void)
{
    volatile uint32_t StartUpCounter = 0, HSEStatus = 0;

    /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------------*/
    /* Enable HSE */
    RCC->CR |= ((uint32_t)RCC_CR_HSEON);

    /* Wait till HSE is ready and if Time out is reached exit */
    do
    {
        HSEStatus = RCC->CR & RCC_CR_HSERDY;
        StartUpCounter++;
    } while ((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

    if ((RCC->CR & RCC_CR_HSERDY) != RESET)
    {
        HSEStatus = (uint32_t)0x01;
    }
    else
    {
        HSEStatus = (uint32_t)0x00;
    }

    if (HSEStatus == (uint32_t)0x01)
    {
        /* Enable Prefetch Buffer */
        FLASH->ACR |= FLASH_ACR_PRFTBE;

        /* Flash 0 wait state */
        FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
        FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_0;

        /* HCLK = SYSCLK */
        RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;

        /* PCLK2 = HCLK */
        RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1;

        /* PCLK1 = HCLK */
```

```
	RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;

	/* Configure PLLs ------------------------------------------------------*/
	/* PLL configuration: PLLCLK = ???? */
	RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC |
RCC_CFGR_PLLMULL);
	RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 |
RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL8);

	/* PLL2 configuration: PLL2CLK = ???? */
	/* PREDIV1 configuration: PREDIV1CLK = ???? */
	RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL |
RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
	RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL8 |
RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV8);

	/* Enable PLL2 */
	RCC->CR |= RCC_CR_PLL2ON;
	/* Wait till PLL2 is ready */
	while ((RCC->CR & RCC_CR_PLL2RDY) == 0)
	{
	}

	/* Enable PLL */
	RCC->CR |= RCC_CR_PLLON;

	/* Wait till PLL is ready */
	while ((RCC->CR & RCC_CR_PLLRDY) == 0)
	{
	}

	/* Select PLL as system clock source */
	RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
	RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;

	/* Wait till PLL is used as system clock source */
	while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08)
	{
	}

	/* Select System Clock as output of MCO */
	//@TODO
	RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
	RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK;

  }
  else
  { /* If HSE fails to start-up, the application will have wrong clock
    configuration. User can add here some code to deal with this error */
  }
}

void init_Timer2() {
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  // TIM2 Clock Enable
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

  // Enbale 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);

    // TIM2 Initialize
    // 1/40MHz * prescaler * period
    TIM_TimeBaseStructure.TIM_Period = 10000-1;
    TIM_TimeBaseStructure.TIM_Prescaler = 40-1;

    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    // TIM2 Enable
   // TIM_ARRPreloadConfig(TIM2, ENABLE);
    TIM_Cmd(TIM2, ENABLE);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); // interrupt enable
} // 1/100초에 한 번씩 호출되는 함수 : TIM2_IRQ

void TIM2_IRQHandler(void) {

        if ((TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)){
          t0 = t0 + flag;

          if(t0%2)
              GPIO_SetBits(GPIOD, GPIO_Pin_2);
          else
              GPIO_ResetBits(GPIOD, GPIO_Pin_2);

           TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        }

}

void RCC_Configure(void) {
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}

void GPIO_Configure(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    GPIOD->CRL = GPIO_CRL_MODE2_0 ;
}

void delay(int i){
    int j;
    for(j=0; j<=i * 100000; j++);
}

void blink(void) {

    while(1) {
```

```
        GPIOD->BSRR = GPIO_BSRR_BS2;
        // delay()?
        GPIOD->BRR = GPIO_BRR_BR2;
    }
}

int button_pos[3];
int button[3] = {0, };

int main(void)
{
    int i;
    int rSize = 30, cSize = 15;
    uint16_t pos_x, pos_y;
    uint16_t pix_x, pix_y;
    int button_pos[3];
    int t1 = 0, t2 = 0, t3 = 0, t4 = 0;

    SysInit();
    SetSysClock();
    init_Timer2();
    LCD_Init();
    Touch_Configuration();
    Touch_Adjust();
    LCD_Clear(WHITE);

    RCC_Configure();
    GPIO_Configure();

    for(i=1;i<4;++i)
        LCD_DrawRectangle(65 * i - rSize, 220 - cSize,
                          65 * i + rSize, 220 + cSize);
    for(i=1;i<4;++i)
        button_pos[i] = 65 *i + rSize*2;


    LCD_ShowString(50 * 1 , 210, "Start", BLACK, WHITE);
    LCD_ShowString(50 * 2 + 20 , 210, "Stop", BLACK, WHITE);
    LCD_ShowString(50 * 3 + 20, 210, "Reset", BLACK, WHITE);

    LCD_ShowString(50 * 0 + 10, 150, "min", BLACK, WHITE);
    LCD_ShowString(50 * 1 + 10, 150, "sec", BLACK, WHITE);
    LCD_ShowString(50 * 2 + 10, 150, "1/10", BLACK, WHITE);
    LCD_ShowString(50 * 3 + 10, 150, "1/100", BLACK, WHITE);

    while(1) {
        LCD_ShowString(100, 50, "TUE_TEAM07", BLACK, WHITE);


        if(t0 >= 10){
            t0 = 0;
            t2++;
        }
        if(t2 >= 10){
            t2 = 0;
            t3++;
        }
        if(t3 >= 60){
            t3 = 0;
            t4++;
```

```
        }

//      if(t0%2==0)
//         GPIOD->BSRR = GPIO_BSRR_BS2;
//      else
//         GPIOD->BRR = GPIO_BRR_BR2;

      LCD_ShowNum(50 * 0 + 10, 100, t4, 3, BLACK, WHITE);
      LCD_ShowNum(50 * 1 + 10, 100, t3, 3, BLACK, WHITE);
      LCD_ShowNum(50 * 2 + 10, 100, t2, 3, BLACK, WHITE);
      LCD_ShowNum(50 * 3 + 10, 100, t0, 3, BLACK, WHITE);

      Touch_GetXY(&pos_x, &pos_y,0);
      Convert_Pos(pos_x, pos_y, &pix_x, &pix_y);

      for(i=0;i<3;++i) {
            if(button_pos[i]-rSize <= pix_x  && pix_x <= button_pos[i]+rSize && 220-cSize
<= pix_y && pix_y <= 220+cSize) {

               //button[i] = (button[i]+1)%2;
               if(i == 0){
                  flag = 1;
               }
               else if(i == 1){
                  flag = 0;
               }
               else if(i == 2){
                  t0 = 0;
                  t2 = 0;
                  t3 = 0;
                  t4 = 0;
               }

            }

         }
   }
}
```

※touch.c는 주어진 파일을 수정하지 않고 그대로 활용하였으므로, 따로 소스코드를 첨부안함

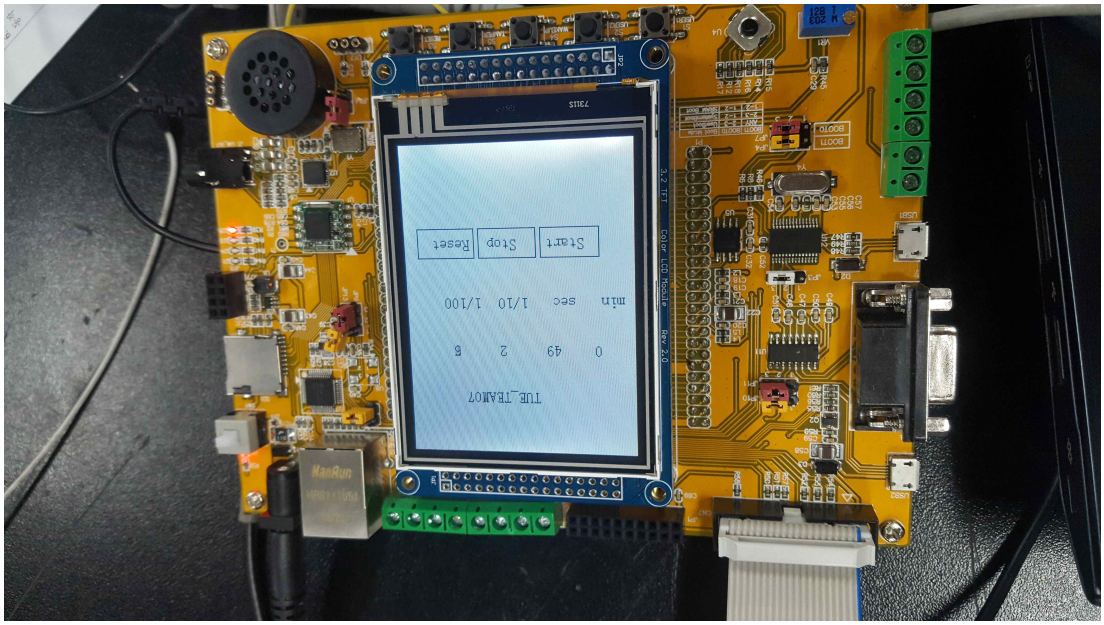
2) 실험결과

2-1) LCD 화면에 분, 초, 1/10초, 1/100초 단위로 정확한 숫자가 출력되는 지 확인

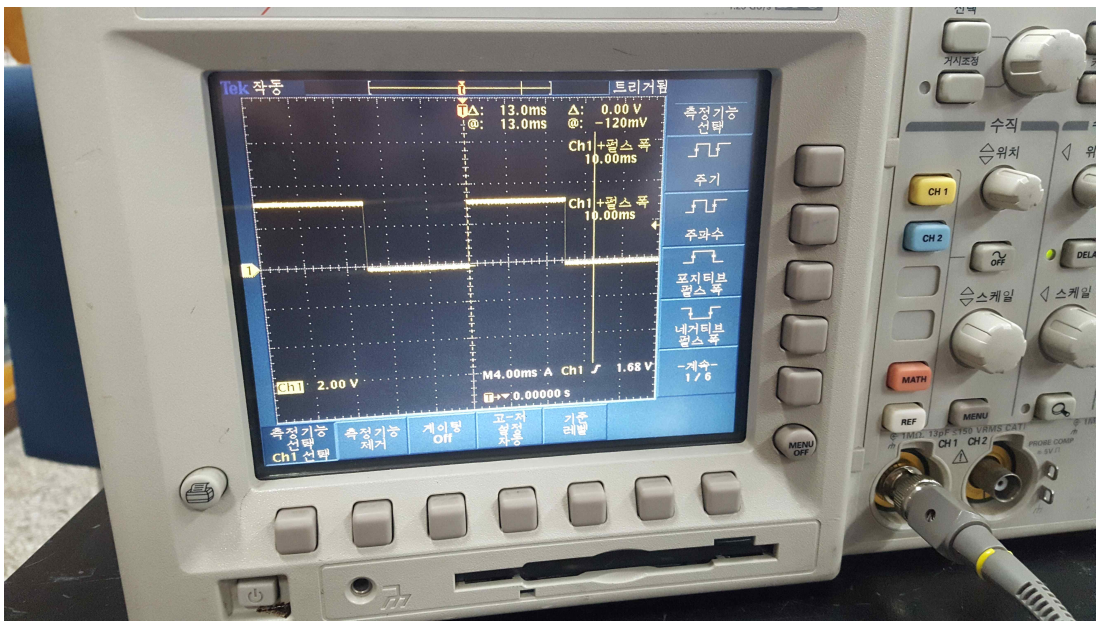
2-2) Start Stop, Reset 버튼에 따른 LCD화면에 시간 변화 확인

2-3) 주어진 주기에 맞춰 LED가 깜빡이는지 확인

2-4) 오실로스코프를 이용하여 LED가 깜빡이는 주기 확인




< 그림 14 : LCD에 시간, 버튼 출력 >




< 그림 15 : 오실로스코프를 이용해 LED 주기 확인 >

## 5. 결론 및 느낀점

이번 실험에서는 타이머의 원리와 동작을 배웠다. 타이머를 통해 원하는 시간의 주기마다 인터럽트를 발생시켜 LED를 제어하였다. 또한 이 타이머로 LCD를 통해 스탑워치를 구현하였다. 마지막으로 오실로스코프를 통해 우리가 정한 타이머의 주기가 정확히 일치하는지 확인하였다. 저번 주차 실험과 중복되는 부분이 많아 수월하게 진행할 수 있다고 생각했는데 많은 부분에서 지체가 되었다. 특히 LCD 제어 부분에서 많은 시간이 걸렸는데, 이는 보드와 LCD 장치의 연결 부분이 불안정하고 LCD에 특정 도형을 그리는 함수가 직관적이지 않아 사용하는데 어려움이 있었다. 하지만 이번 실험의 시행착오를 통해 LCD 제어를 숙달할 수 있었고 이것은 나중에 텀프로젝트시 LCD 사용에 많은 도움이 될것이라고 생각한다. 또한 코드를 나눠서 작성후 병합하는 과정에서 누락되는 부분이 생겨 LCD가 제대로 동작하지 않았다. 처음에 이 실수를 발견하지 못해 다른 부분에서 동작 오류의 원인을 찾느라 시간이 많이 지체되었다. 나중에 텀프로젝트를 하면 서로 분담하여 병합하는 과정을 거치게 될 것인데 이 과정에서 이러한 실수를 하지 않도록 주의해야겠다.