2016년 2학기
컴퓨터하드웨어 실험 보고서
7조 10주차

한성희 이호욱 한경수

Table Of Contents

개요

ADC 의 원리를 이해하고 TFT LCD 를 제어한다.

기본 개념 (배경 지식)

TFT LCD

LCD 구조

LCD 는 구동 방식에 따라 Passive matrix 구조와 Active matrix 구조로 나뉜다.

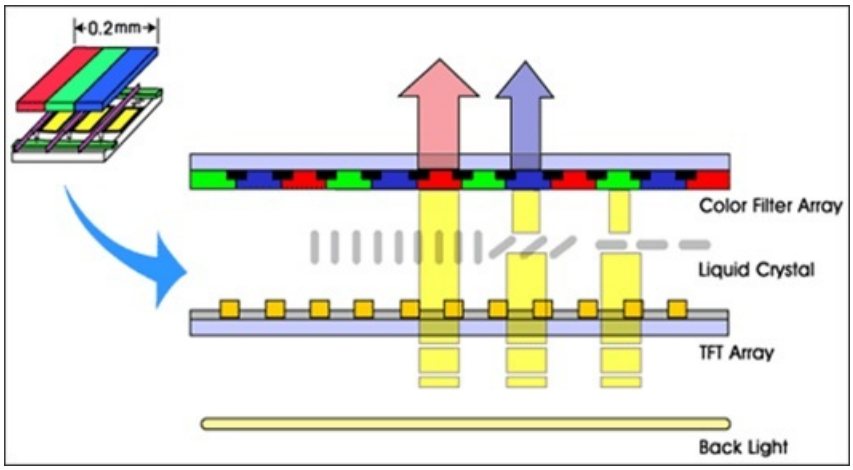
1. Passive matrix LCD

각 화소점이 두 개의 겹쳐지는 전극 단자에 의해 구동되므로 독립적으로 작동하지 못한다. Contrast 와 응답속도, 색 표시 특성의 확보가 어렵다. TN, STN LCD 가 이에 해당된다.

2. Active matrix LCD (AMLCD)

각 화소 하나 하나에 트랜지스터가 형성되어 있어 화소가 켜짐과 꺼짐 동작을 각각 독립적으로 제어할 수 있다. 상대적으로 고화질, 빠른 응답속도를 얻을 수 있으며 TFT LCD 가 이에 해당된다.

TFT LCD 원리



TFT(Thin Film Transistor : 박막 트랜지스터) 는 액정 표시 방식 중 하나로 액정 화소 하나 하나에 반도체 스위치를 붙여 표시를 제어하는 방식이다.

LCD(Liquid Ctystal Display) 는 2개의 얇은 유리판 사이에 액체와 고체의 중간적 특성을 가지는 액정(Liquid Crystal)을 주입하여 전원 공급 시 액정 분자의 배열을 변화시켜 명암의 변화를 이용하여 영상을 표시하는 디스플레이다.

TFT LCD 는 크게 비정질 실리콘 등의 반도체 박막이 형성되어 있는 아래 유리기판과 Color Filter 가 형성되어 있는 윗 유리기판 그리고 그 사이에 주입된 액정(Liquid Crystall) 로 구성되어 있다.

액정은 인가된 전압에 따라 분자구조를 달리하여 빛의 투과를 제어하는데, 이때 두 전극 사이의 전압은 아래 유리기판에 존재하는 TFT를 통해 전압을 조절한다.

아래 유리기판 밑의 백라이트 유닛에서 빛을 공급하면 빛은 액정의 배열을 따라 지나가 윗 유리기판의 Color Filter 를 통해 나가면서 색을 표한하게 된다.

TFT LCD Timing characteristics

TFT LCD 는 사용하는 interface 에 따라 Timing 특성이 바뀐다.

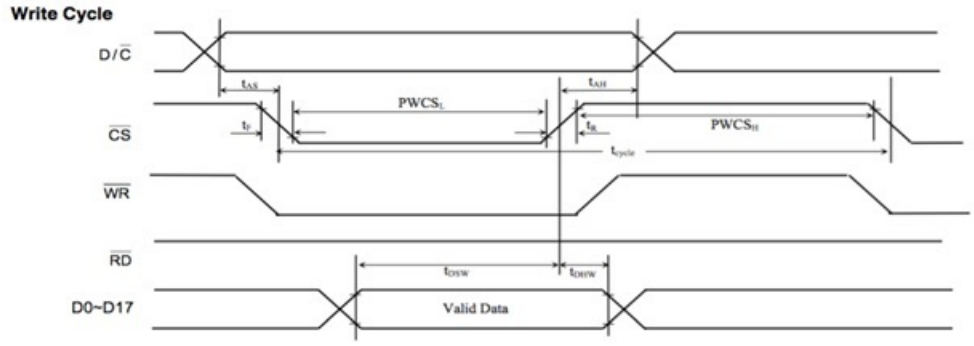
- Interface 종류
- Serial Peripheral Interface : master/slave 방식의 통신. Clock 을 동기화 한 후, 8 bits 의 데이터를 통신
- Parallel 6800 series Interface : read/write(R/W) 와 enable(E) 제어신호를 사용하여 read, write 를 제어
- Parallel 8080 series Interface : read(RD), write(WR) 제어신호를 사용하여 read, write 를 제어

이 중 이번 실험에서 Parallel 8080 series Interface 를 사용한다.

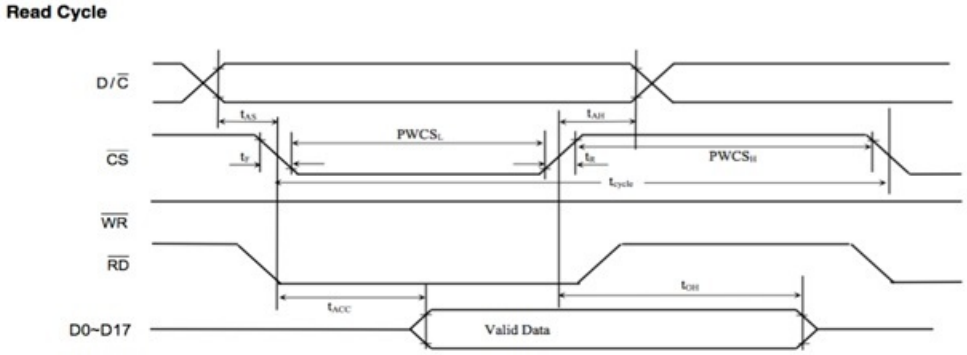
Parallel 8080 series Interface 의 제어신호

- CS (Chip Select) : low 일 때, chip이 동작되도록 하는 신호
- RS = D/C (Data/Command) : low 시, command 전송. high 시, data 전송
- WR (Write) : low 에서 high 로 변할 때, data 를 display RAM 에 write 한다.

- RD (Read) : low 에서 high 로 변할 때, display RAM 에서 data 를 read 한다.

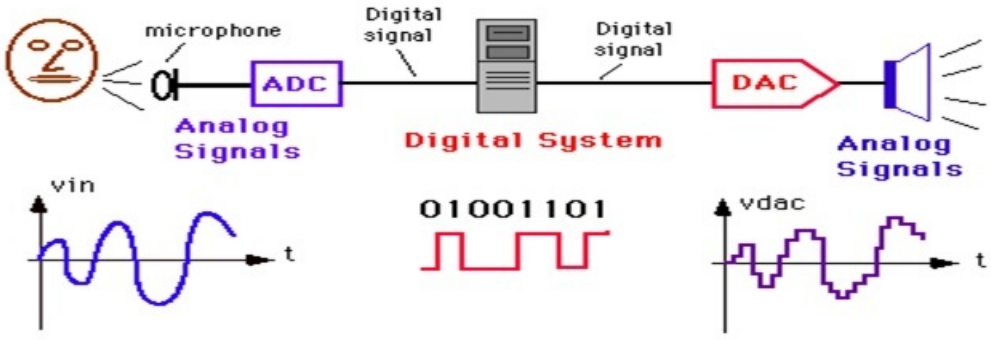


- CS 를 low 로 하여 chip 을 동작시킨다.
- RS(=D/C) 가 low 이면 command, high 이면 display data 가 register 에 write 된다.
- write cycle 이므로 RD 는 항상 high 로 둔다.
- WR 이 low 에서 high 로 display RAM 에 data 또는 register 에 command 를 write 한다.



- CS 를 low 로 하여 chip 을 동작시킨다.
- RS 를 high 로 하여 display RAM 에 저장된 display data 를 읽어오도록 한다.
- read cycle 이므로 WR 은 high 를 유지한다.
- RD 가 low 에서 high 로 바뀔 때 display data 를 읽는다.

ADC & DAC



컴퓨터는 2진 digital data 를 사용하므로 컴퓨터가 신호를 감지할 수 있도록 analog 신호를 digital 신호로 변환한다.

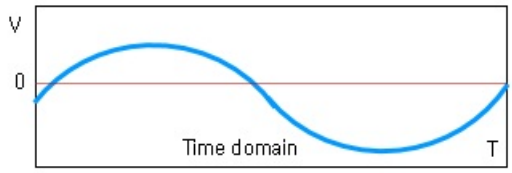
digital 신호는 analog 신호보다 명확하고 규칙적이므로 손실 없이 data 를 저장 또는 이용할 수 있다.

ADC (Analog to Digital Conversion)

온도, 습도, 조도 등의 analog 물리량을 digital 신호로 변환하는 것.

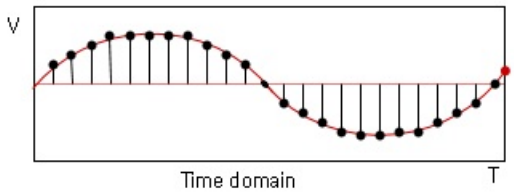
A/D Converter 의 과정

1) Analog Siganl

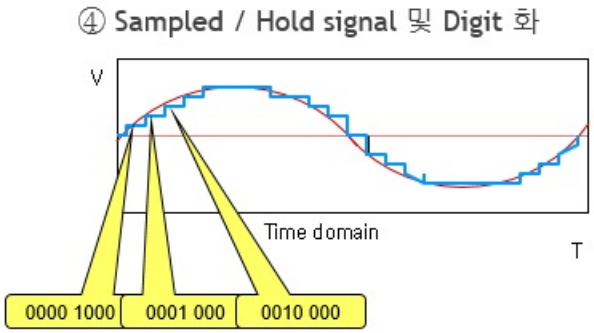


2) Sampling Pulse (sampling 속도 결정)

3) Sampled Signal



4) Sampled / Hold Signal 및 Digit 화

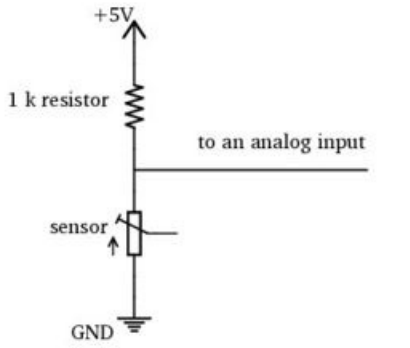


ADC 사용 목적

물리적 양을 측정하는 센서의 값들은 Analog 값이다. 예를 들면 소리의 크기, 빛의 양, 압력 등이 이에 속한다. 이 값들을 쓰기 위해선 디지털로 샘플링을 해야하는데, 이를 ADC 라고 한다.

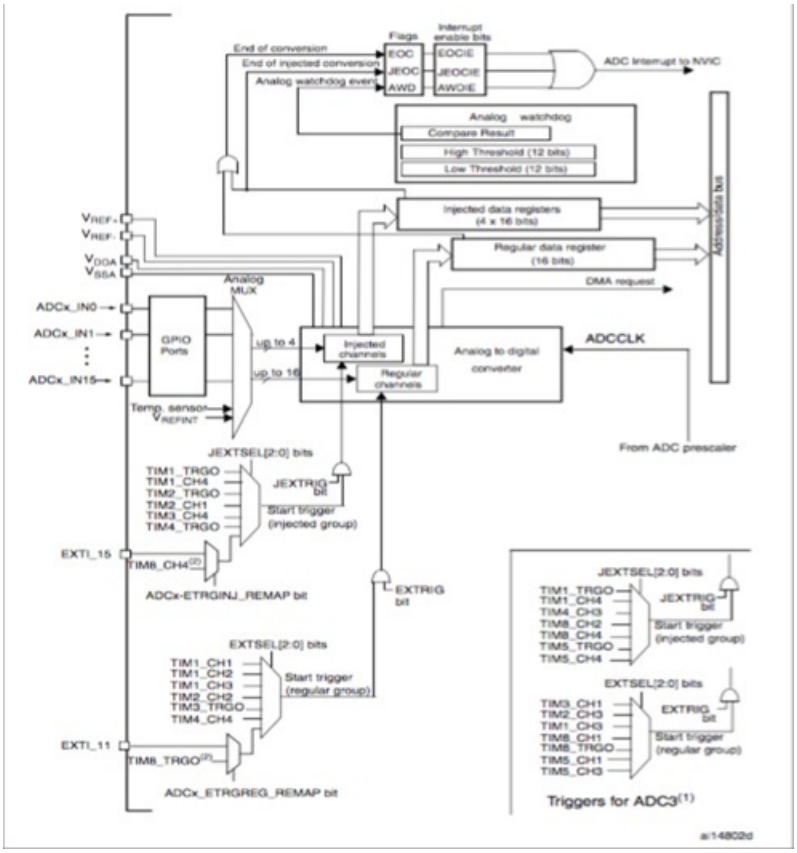
ADC 사용 예시

- 조도 센서



센서에서 전달받은 analog 신호를 ADC 가 digital 신호로 바꿔준다.

- Single ADC Block diagram



실험 보드에는 2개의 12bits ADC 가 존재한다. 각 ADC 는 16 개의 체널을 가진다.

외부에서 들어온 신호를 ADC 를 통해 ADC interrupt 를 발생시킨다.

DAC (Digital to Analog Conversion)

ADC 의 역방향 처리 과정으로 0과 1로 표현되는 digital 신호를 사람이 들을 수 있는 analog 신호로 변환시켜주는 장치이다.

정해진 bit 수로 인해 정밀도는 제한적이다.

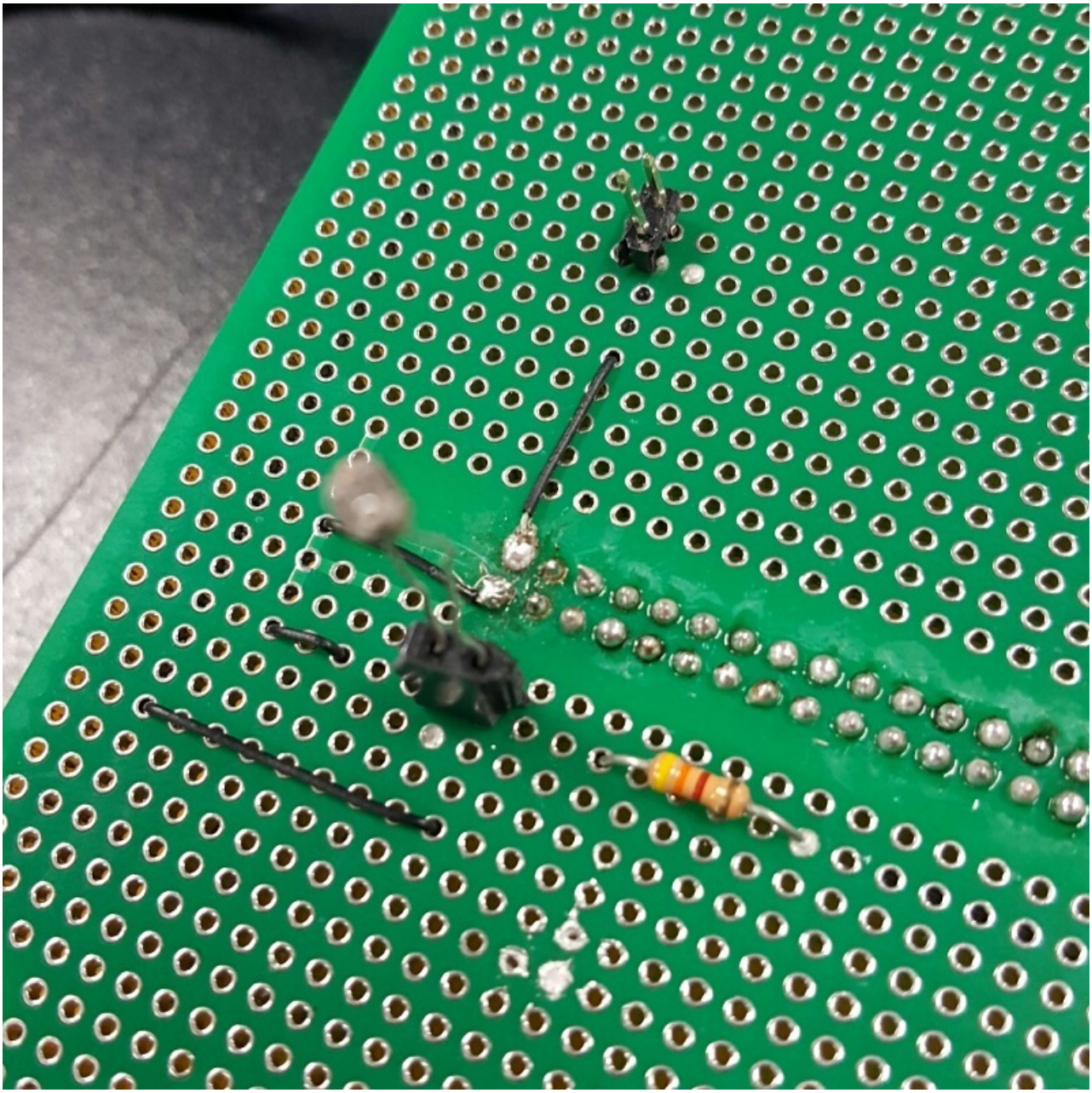
D/A Converter 의 과정

1) 증폭기(Amplifier)를 통해 n 개의 digital 신호를 analog 신호의 크기로 바꾸어 스텝 신호를 출력

2) 필터 회로를 사용하여 출력된 스텝 신호를 원래 신호에 가깝게 복원

실험 방법

그림과 같이 TFT-LCD 를 보드에 결합한다.



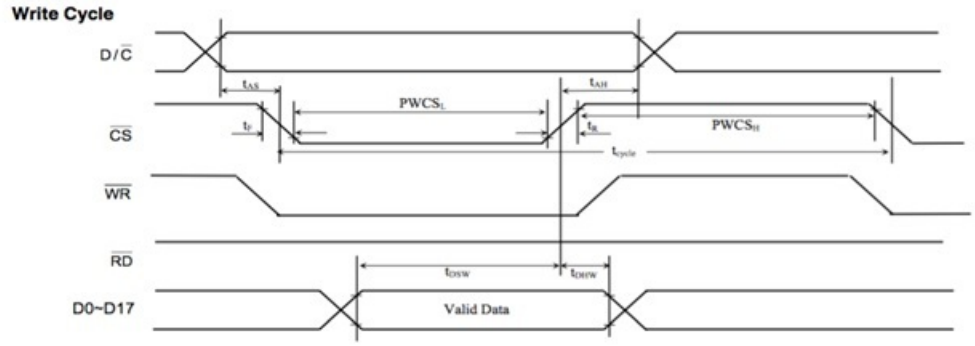
그리고 조도센서를 보드에 납땜한다. 출력가 입력포트를 주의해서 납땜해야 한다.

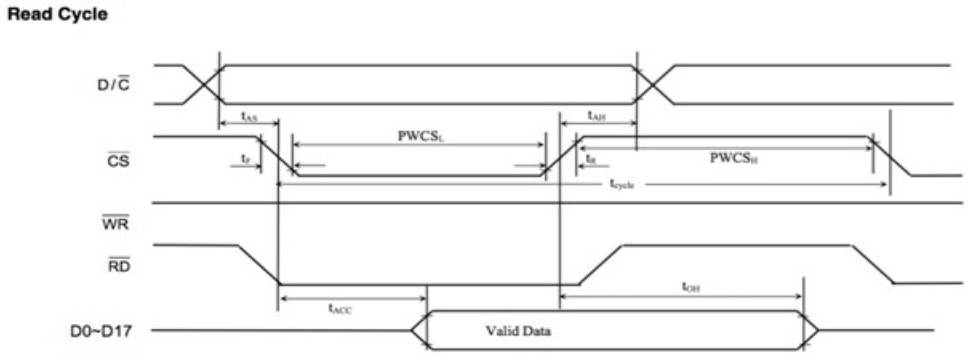
lcd.h, lcd.c, touch.h, touch.c, font.h, font.c 를 라이브러리에 추가한다.

```
LR_IROM1 0x08000000 0x00080000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00080000  {  ; load address = execution address
   *.o(RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00008000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

플래시 업로드를 위한 sct 파일을 위와 같이 바꿔준다. 지금까지는 0x00008000 으로 실험했다. 그 이유는 이전보다 더 많은 라이브러리를 사용하기 때문에, 더 넓은 메모리 공간이 필요해서 확장한 것이다.

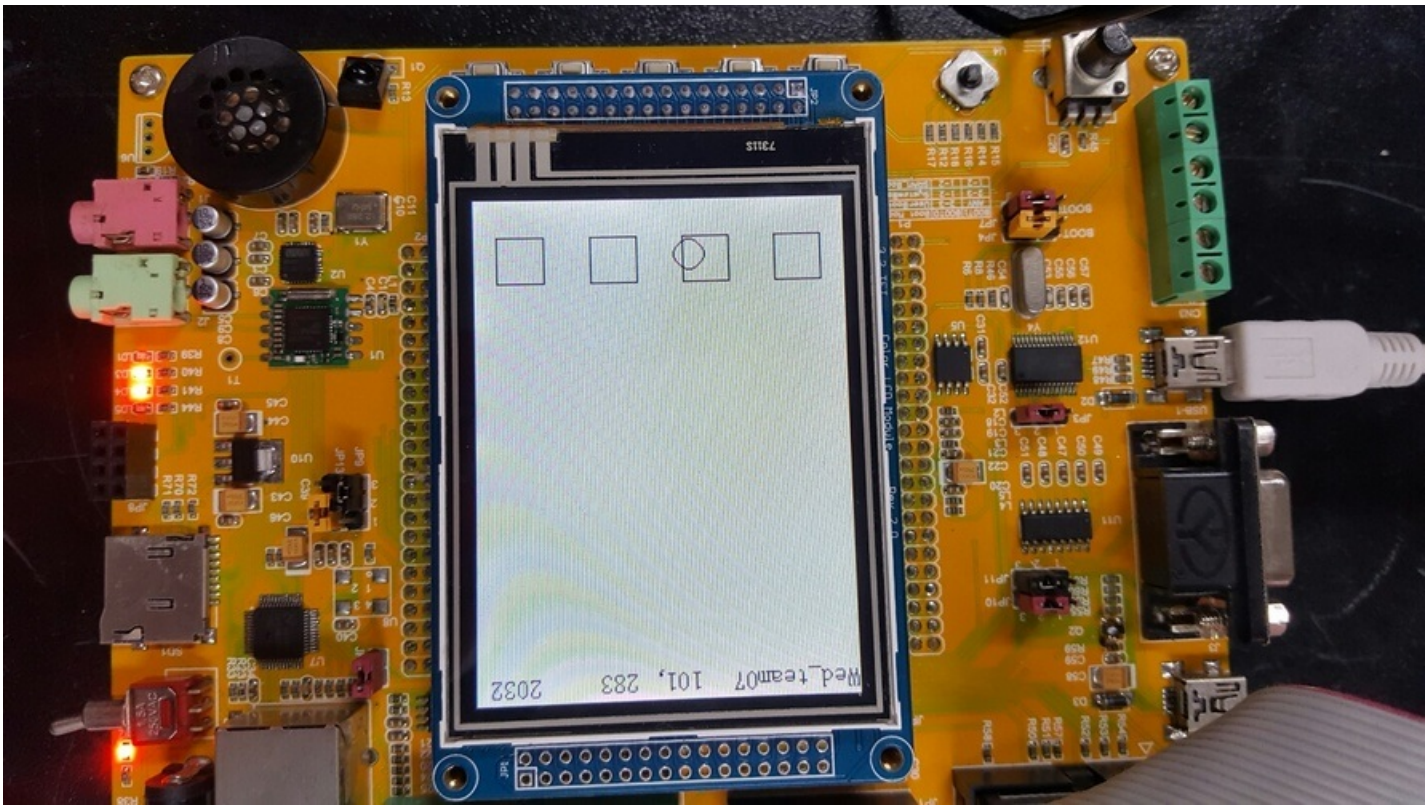
만약 sct 파일을 이전과 같이 0x00008000 처럼 사용한다면, no space 에러가 발생한다.





그림과 같이 타이밍 그래프를 참고하여 lcd.c 의 LCD*WR*REG, LCD*WR*DATA, LCD*ReadReg*, *LCD*WriteReg 함수를 작성한다.

```
void LCD_WR_REG(uint16_t LCD_Reg) {
        GPIO_SetBits(GPIOD, GPIO_Pin_15); //LCD_RD(1)
        GPIO_ResetBits(GPIOD, GPIO_Pin_13); //LCD_RS(0);
        GPIO_ResetBits(GPIOC, GPIO_Pin_6); //LCD_CS(0);
        GPIO_ResetBits(GPIOD, GPIO_Pin_14); //LCD_WR(0);
        DataToWrite(LCD_Reg);
        GPIO_SetBits(GPIOC, GPIO_Pin_6); //LCD_CS(1);
        GPIO_SetBits(GPIOD, GPIO_Pin_14); //LCD_WR(1);
}

void LCD_WR_DATA(uint16_t LCD_Data) {
        GPIO_SetBits(GPIOD, GPIO_Pin_15); //LCD_RD(1)
        GPIO_SetBits(GPIOD, GPIO_Pin_13); //LCD_RS(1);
        GPIO_ResetBits(GPIOC, GPIO_Pin_6); //LCD_CS(0);
        GPIO_ResetBits(GPIOD, GPIO_Pin_14); //LCD_WR(0);
        DataToWrite(LCD_Data);
        GPIO_SetBits(GPIOC, GPIO_Pin_6); //LCD_CS(1);
        GPIO_SetBits(GPIOD, GPIO_Pin_14); //LCD_WR(1);
}


/* LCD ReadReg */
uint16_t LCD_ReadReg(uint16_t LCD_Reg)
{
        uint16_t temp;
        GPIOF->CRL=0x88888888;
        GPIOF->CRH=0x88888888;

        //LCD_WriteReg(LCD_Reg);
        GPIO_SetBits(GPIOD, GPIO_Pin_14);//LCD_WR(1);
        GPIO_SetBits(GPIOD, GPIO_Pin_13); //LCD_RS(1);
        GPIO_ResetBits(GPIOC, GPIO_Pin_6); //LCD_CS(0);
        GPIO_ResetBits(GPIOD, GPIO_Pin_15);//LCD_RD(0);
        temp = DataToRead();
        GPIO_SetBits(GPIOC, GPIO_Pin_6);//LCD_CS(1);
        GPIO_SetBits(GPIOD, GPIO_Pin_15);//LCD_RD(1);

        ////////////////////////////
        GPIOF->CRL=0x33333333;
        GPIOF->CRH=0x33333333;
        return temp;
}


void LCD_WriteReg(uint16_t LCD_Reg ,uint16_t LCD_RegValue)
{
        LCD_WR_REG(LCD_Reg);
        LCD_WR_DATA(LCD_RegValue);
}

```



프로그램을 돌리면 다음과 같이 지정 위치의 좌표와 조도 값, 아래의 버튼을 이용하여 LED 를 조작할 수 있다.

결론

TFT-LCD 를 동작시키고, 화면에 글자와 도형을 표시하고 터치를 입력받아서 처리하는 실험이었다. 이 실험의 경우 Term-Project 에서 매우 중요하게 쓰일것 같다. 특히 스캐터 파일의 메모리 공간을 크게 잡는게 인상적이었다.

조도센서의 값을 lcd 에 표시해봤는데 생각보다 민감한 빛의 변화를 감지하지 못했다. 이는 하드웨어적 문제인것 같다.

또한 lcd.c 내부를 살펴보면서 Touch*GetXY 내부에서 T* INT 의 역할을 알게됐고, 다른 기본적인 draw 함수들도 매우 간단한 wrapper function 에 불과하다는것을 알게 됐다. 우리가 곧 구현할 키패드 구현에도 크게 작용할 것 같다.

하지만 이 실험에서 해결못한 이전에 찍었던 점을 지워주는 방법에 대해서는 좀 고민해봐야할것 같다. draw fill 을 통해서 이전 영역을 지워주면 될것 같았는데 잘 안됐다. 아마 내부적으로 좌표의 범위의 문제가 있어서 잘못된 동작으로 빠진것으로 추정된다.

전체 소스

```
#include <misc.h>
#include <stm32f10x.h>
#include <stm32f10x_exti.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
#include <stm32f10x_usart.h>
#include <stm32f10x_adc.h>
#include <lcd.h>
#include <Touch.h>
#include <stdio.h>
#include <stdlib.h>
uint16_t pos_x,pos_y;
uint16_t pix_x,pix_y;
uint16_t jodo;

// 버튼의 위치를 저장하는 변수
int button_pos[4][2];

// led 켜지고 꺼진 상태 토글
int button[] = {0, 0, 0, 0};

void delay(int i){
        int j;
        for(j=0; j<=i * 100000; j++);
}
void SysInit(void)
{
        /* Reset the RCC clock configuration to the default reset state(for de
bug purpose) */
```

```
        /* Set HSION bit */
        RCC->CR |= (uint32_t)0x00000001;

        /* 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 = 48MHz */
                RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;

                /* PCLK2 = HCLK = 48MHz */
                RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1;

                /* PCLK1 = HCLK  = 24MHz  */
                RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;

                /* Configure PLLs ------------------------------------------
----------*/
                /* PLL configuration: PLLCLK = PREDIV1 * 6 = 48MHz */
                RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC |
RCC_CFGR_PLLMULL);
                RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_P
LLSRC_PREDIV1 |
                                        RCC_CFGR_PLLMULL6);

                /* PLL2 configuration: PLL2CLK = HSE/5 * 8 = 40MHz  */
                /* PREDIV1 configuration: PREDIV1CLK = PLL2 / 5 = 8MHz */
                RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MU
L |
                                        RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
                RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PL
L2MUL8 |
                                        RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_
DIV5);

```

```
                /* 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)
                {
                }
        }
        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 set_ENABLE(void) {
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);         // interr
upt
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);     // RCC GPIO E
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);     // RCC GPIO C
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE);     // RCC GPIO D
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);         // ADC1
}

void set_PC1(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);
}

void set_ADC(void) {
        ADC_InitTypeDef ADC_InitStructure;
        ADC_DeInit(ADC1);
        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
        ADC_InitStructure.ADC_ScanConvMode = ENABLE;
        ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
        ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
        ADC_InitStructure.ADC_NbrOfChannel = 1;
        ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1, ADC_SampleTime_239Cy
cles5);
        ADC_Init(ADC1, &ADC_InitStructure);
}

void set_NVIC(void) {
        NVIC_InitTypeDef NVIC_InitStructure;
        NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
        ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
        ADC_Cmd(ADC1, ENABLE);
}

void ADC_start(void) {
        ADC_ResetCalibration(ADC1);
        while(ADC_GetResetCalibrationStatus(ADC1));
        ADC_StartCalibration(ADC1);
        while(ADC_GetCalibrationStatus(ADC1));
        ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}


void ADC1_2_IRQHandler(void) {
        uint16_t input;
        u8 str[10];
        double result;
//
////        while(ADC_GetFlagStatus(ADC1, 0x2)==RESET);
```

```
204         input = ADC_GetConversionValue(ADC1);
205         result = (double)input;
206         jodo = result;
207         ADC_ClearITPendingBit(ADC1,ADC_IT_EOC);
208         //        LCD_DrawRectangle(pix_x, pix_y, pix_x+10, pix_y+10);
209 }
210
211 int main() {
212         int rSize = 15;
213         int i;
214         unsigned int i2BS[] = { GPIO_BSRR_BS2, GPIO_BSRR_BS3, GPIO_BSRR_BS4,
215                         GPIO_BSRR_BS7 };
216         unsigned int off[] = { GPIO_BRR_BR2, GPIO_BRR_BR3, GPIO_BRR_BR4,
217                         GPIO_BRR_BR7 };
218         char str[10];
219         int prevCircleX = -1, prevCircleY = -1;
220
221         SystemInit();
222         set_ENABLE();
223         set_PC1();
224         set_ADC();
225         set_NVIC();
226         LCD_Init();
227         Touch_Configuration();
228         Touch_Adjust();
229         LCD_Clear(WHITE);
230         ADC_start();
231         GPIOD->CRL = (GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0 |
232 GPIO_CRL_MODE7_0);
233
234         while (1) {
235                 LCD_ShowString(1, 1, "Wed_team07", BLACK, WHITE);
236                 LCD_ShowNum(200, 1, jodo, 4, BLACK, WHITE);
237
238                 Touch_GetXY(&pos_x, &pos_y, 0);
239                 ///
240
241                 for (i = 0; i <= 3; i++) {
242                         button_pos[i][0] = 60 * i + 30;
243                         button_pos[i][1] = 280;
244                         LCD_DrawRectangle(60 * i + 30 - rSize, 280 - rSize,
245                                         60 * i + 30 + rSize, 280 + rSize);
246                 }
247
248                 Convert_Pos(pos_x, pos_y, &pix_x, &pix_y);
249
250                 for (i = 0; i <= 3; i++) {
251                         if (button_pos[i][0] - rSize <= pix_x
252                                         && pix_x <= button_pos[i][0] + rSize
253                                         && button_pos[i][1] - rSize <= pix_y
254                                         && pix_y <= button_pos[i][1] + rSize)
255 {
256                                 button[i] = (button[i] + 1) % 2;
257                                 if (button[i])
258                                         GPIOD->BSRR = i2BS[i]; // led 켜기
259                                 else {
260                                         GPIOD->BRR = i2BS[i]; // led  끄기
261                                 }
262                         }
263                 }
264
265                 if(T_INT == 0)
266                 {
267                         if (prevCircleX != -1 && prevCircleY != -1) {
268                                 LCD_Clear(WHITE);
269                         }
270                         Draw_Circle(pix_x, pix_y, 10);
271                         prevCircleX = pix_x;
272                         prevCircleY = pix_y;

                        sprintf(str, "%d, %d", pix_x, pix_y);
                        LCD_ShowString(100, 2, str, BLACK, WHITE);
                        delay(1);

                }

        }

}
```