임베디드 설계 및 실험

실험 결과 보고서

12 주차 3 조

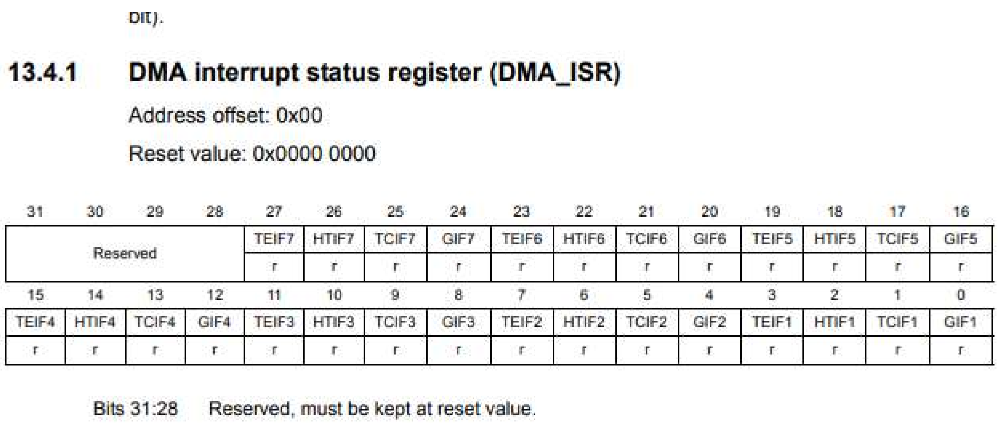
실험 내용

1. DMA 사용하여 ADC 1 개의 채널을 통해 1 개의 조도센서 값을 받아오는 코드 작성.
2. 1 개의 조도센서 값을 TFT-LCD 에 표시(ADC 인터럽트 사용 금지)
3. 센서 값에 적당한 기준(밝음/어두움)을 둬서 어둡다고 판단되면 LED 점등 각각의 조도 센서 값이 밝다고 판단되면 각 LED 소등

DMA(Direct Memory Access 개념

- 주변장치들이 메모리 직접 접근하여 읽거나 쓸 수 있도록 하는 기능
- 메모리 처리 Interrupt 의 사이클 만큼 성능의 향상.

DMA 함수


bit).

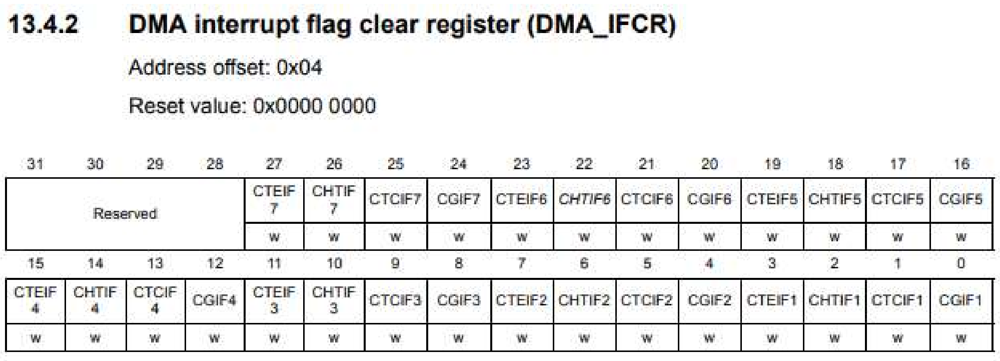
13.4.1 DMA interrupt status register (DMA_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

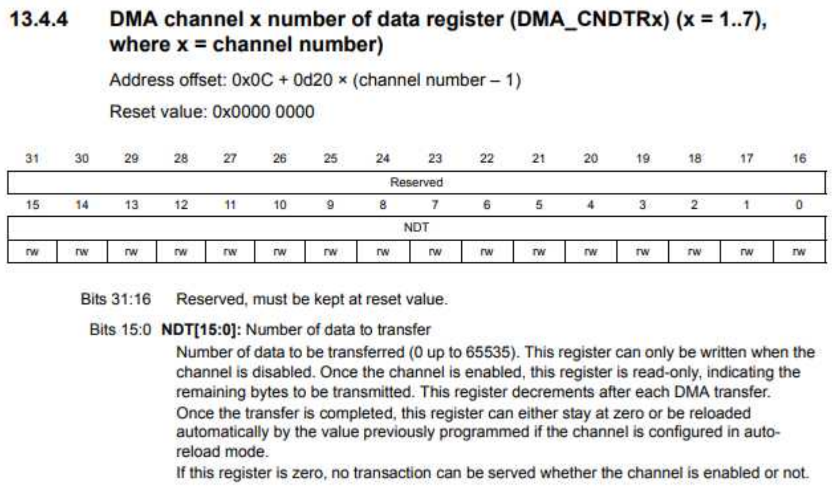
Bits 31:28 Reserved, must be kept at reset value.


13.4.2 DMA interrupt flag clear register (DMA_IFCR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
				w	w	w	w	w	w	w	w	w	w	w	w
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w


13.4.4 DMA channel x number of data register (DMA_CNDTRx) (x = 1..7), where x = channel number)

Address offset: 0x0C + 0d20 × (channel number – 1)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NDT															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

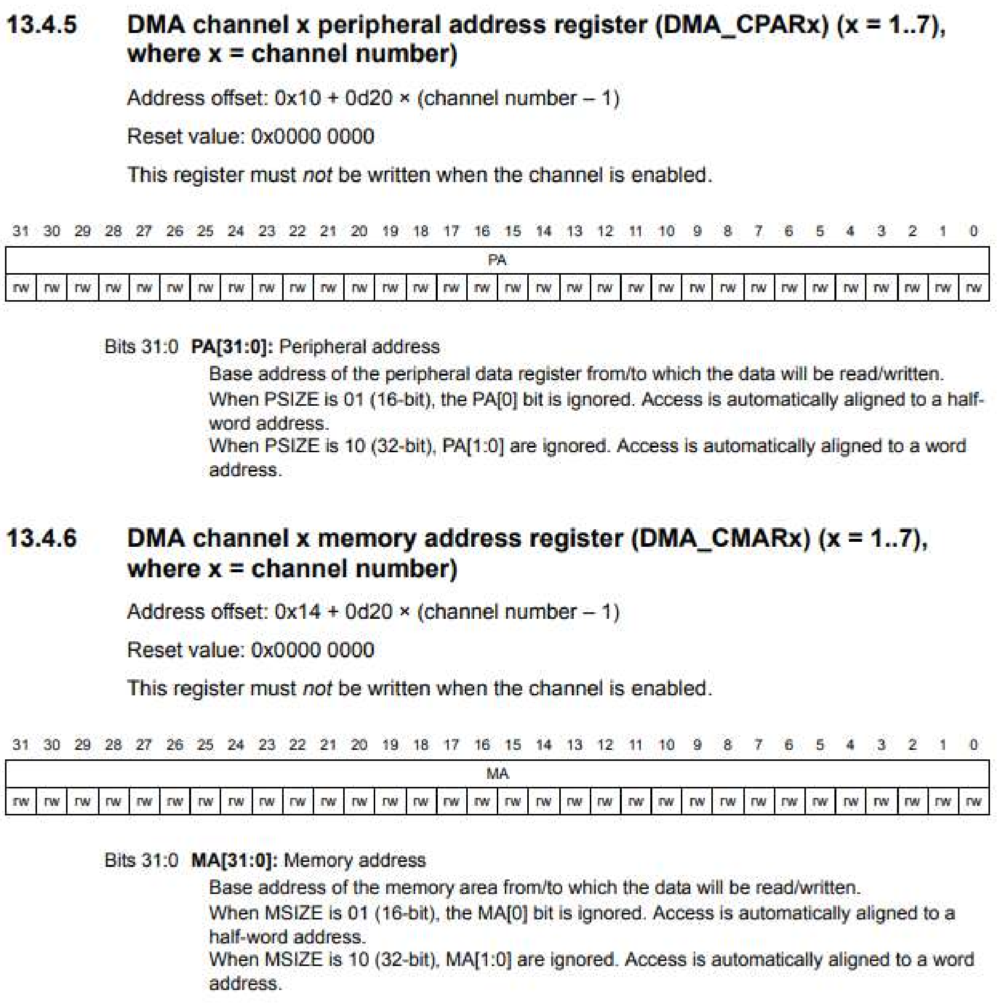
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **NDT[15:0]:** Number of data to transfer

Number of data to be transferred (0 up to 65535). This register can only be written when the channel is disabled. Once the channel is enabled, this register is read-only, indicating the remaining bytes to be transmitted. This register decrements after each DMA transfer.

Once the transfer is completed, this register can either stay at zero or be reloaded automatically by the value previously programmed if the channel is configured in auto-reload mode.

If this register is zero, no transaction can be served whether the channel is enabled or not.



13.4.5 DMA channel x peripheral address register (DMA_CPARx) (x = 1..7), where x = channel number)

Address offset: 0x10 + 0d20 × (channel number – 1)

Reset value: 0x0000 0000

This register must *not* be written when the channel is enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PA																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **PA[31:0]:** Peripheral address
Base address of the peripheral data register from/to which the data will be read/written.
When PSIZE is 01 (16-bit), the PA[0] bit is ignored. Access is automatically aligned to a half-word address.
When PSIZE is 10 (32-bit), PA[1:0] are ignored. Access is automatically aligned to a word address.

13.4.6 DMA channel x memory address register (DMA_CMARx) (x = 1..7), where x = channel number)

Address offset: 0x14 + 0d20 × (channel number – 1)

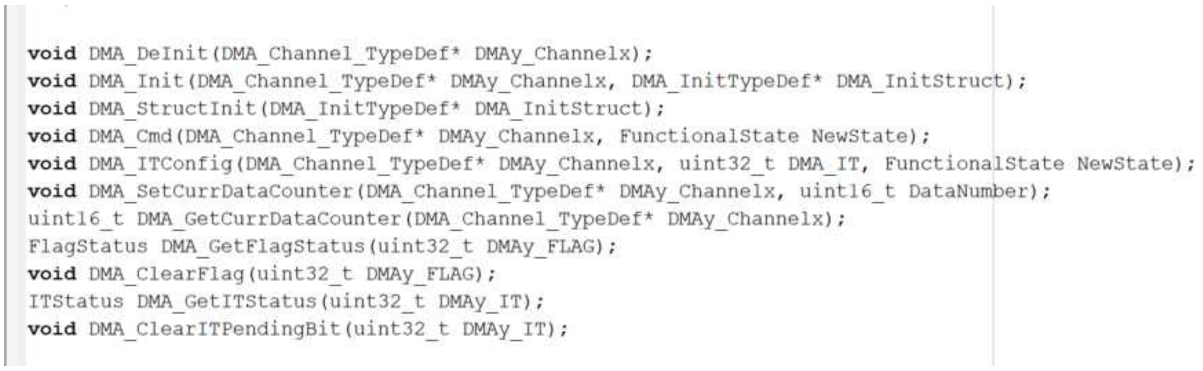
Reset value: 0x0000 0000

This register must *not* be written when the channel is enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MA																															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

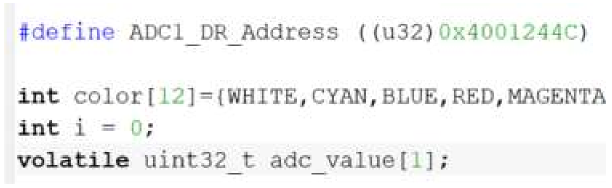
Bits 31:0 **MA[31:0]:** Memory address
Base address of the memory area from/to which the data will be read/written.
When MSIZE is 01 (16-bit), the MA[0] bit is ignored. Access is automatically aligned to a half-word address.
When MSIZE is 10 (32-bit), MA[1:0] are ignored. Access is automatically aligned to a word address.

- Stm32f10x_dma.h 파일에는 아래와 같은 함수가 있음.



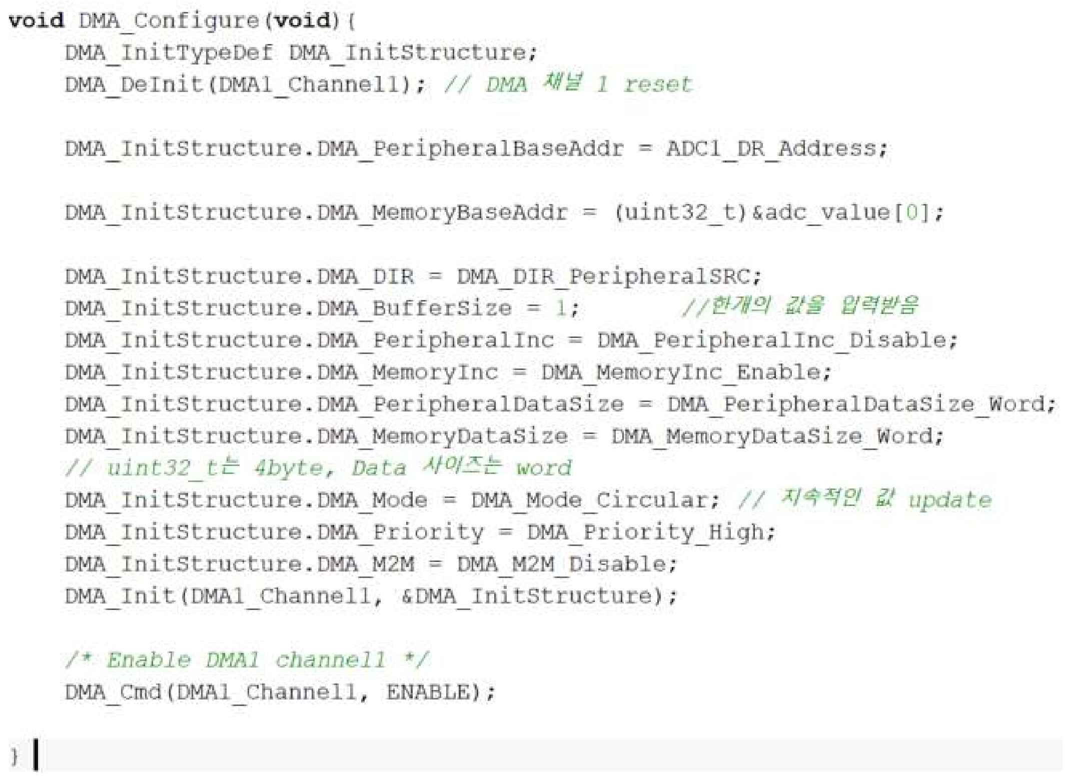
```
void DMA_DeInit(DMA_Channel_TypeDef* DMAy_Channelx);
void DMA_Init(DMA_Channel_TypeDef* DMAy_Channelx, DMA_InitTypeDef* DMA_InitStruct);
void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct);
void DMA_Cmd(DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState);
void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, uint32_t DMA_IT, FunctionalState NewState);
void DMA_SetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx, uint16_t DataNumber);
uint16_t DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMAy_Channelx);
FlagStatus DMA_GetFlagStatus(uint32_t DMAy_FLAG);
void DMA_ClearFlag(uint32_t DMAy_FLAG);
ITStatus DMA_GetITStatus(uint32_t DMAy_IT);
void DMA_ClearITPendingBit(uint32_t DMAy_IT);
```

- ADC1_DR_Address 선언 : 0x40012400 + 4c(offset) : 레퍼런스 문서 참조



```
#define ADC1_DR_Address ((u32)0x4001244C)

int color[12]={WHITE,CYAN,BLUE,RED,MAGENTA
int i = 0;
volatile uint32_t adc_value[1];
```

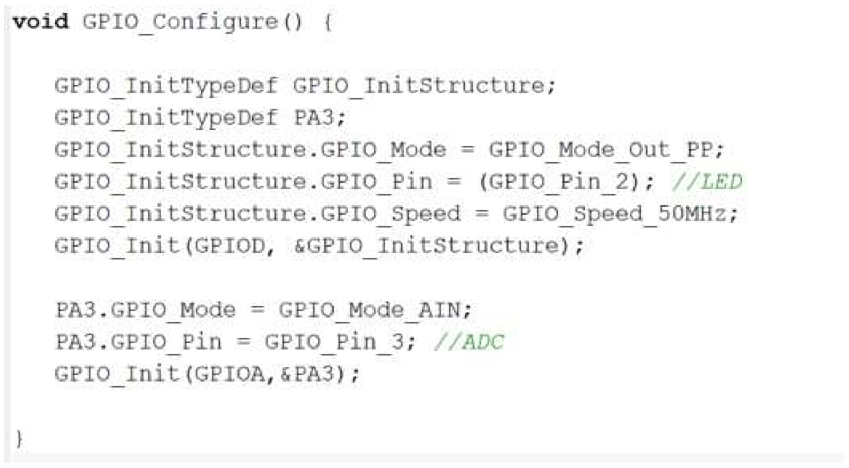
- 헤더파일에 정의된 DMA_InitTypeDef 구조체를 선언한다.
- 여기서 받을 것은 조도 센서 한 개의 값이므로 bufferSize = 1
- Unsinged 32bit int 4byte = 32bit = word,
- 지속적으로 조도센서의 값을 받아와야 되므로 circular 모드로 동작.



```
void DMA_Configure(void){
    DMA_InitTypeDef DMA_InitStructure;
    DMA_DeInit(DMA1_Channel1); // DMA 채널 1 reset

    DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;

    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&adc_value[0];

    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = 1;        //한개의 값을 입력받음
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
    // uint32_t는 4byte, Data 사이즈는 word
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; // 지속적인 값 update
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);

    /* Enable DMA1 channel1 */
    DMA_Cmd(DMA1_Channel1, ENABLE);

}
```

GPIO 설정

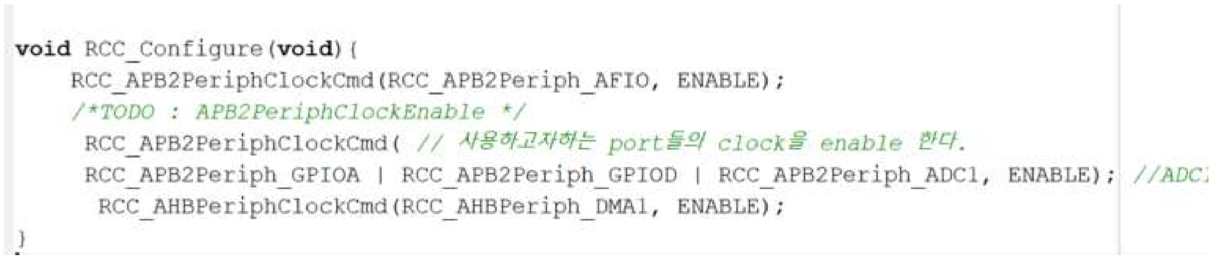
- LED 와 ADC 사용을 위한 GPIO PD2, PA3 설정



```
void GPIO_Configure() {

    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitTypeDef PA3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = (GPIO_Pin_2); //LED
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    PA3.GPIO_Mode = GPIO_Mode_AIN;
    PA3.GPIO_Pin = GPIO_Pin_3; //ADC
    GPIO_Init(GPIOA,&PA3);

}
```

RCC_Configure

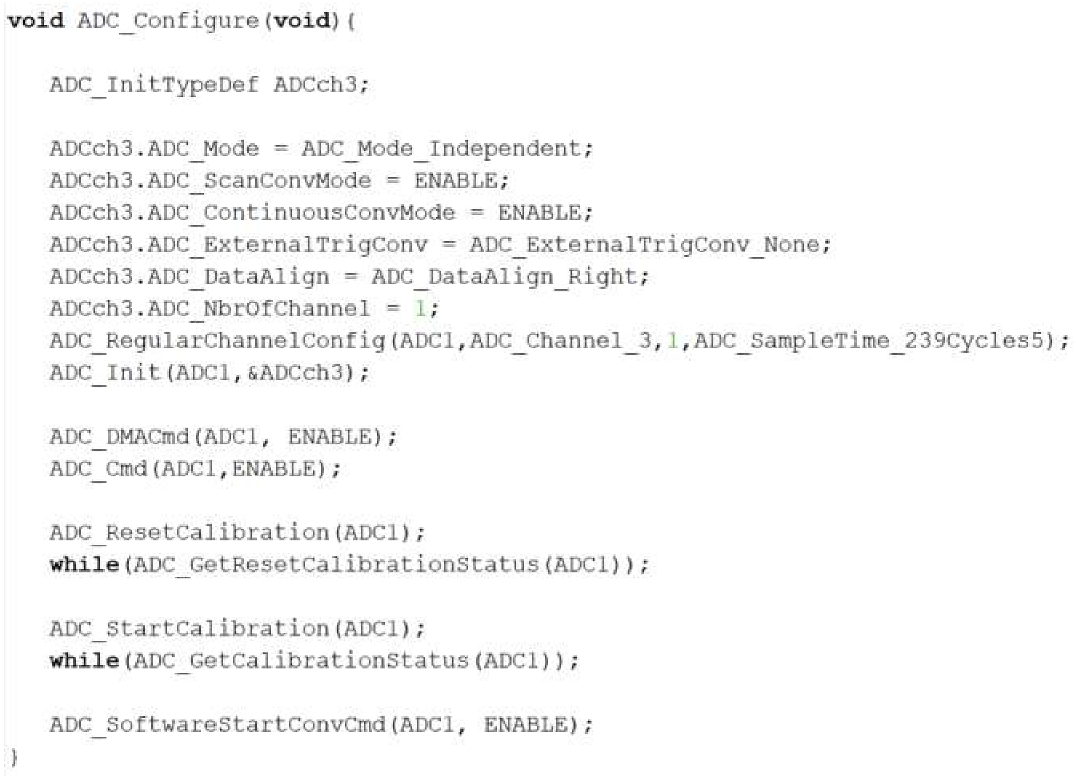
- 사용할 port Clock enable
- ADC, port A D



```
void RCC_Configure(void){
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    /*TODO : APB2PeriphClockEnable */
     RCC_APB2PeriphClockCmd( // 사용하고자하는 port들의 clock을 enable 한다.
     RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOD | RCC_APB2Periph_ADC1, ENABLE); //ADC
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}
```

ADC_Configure

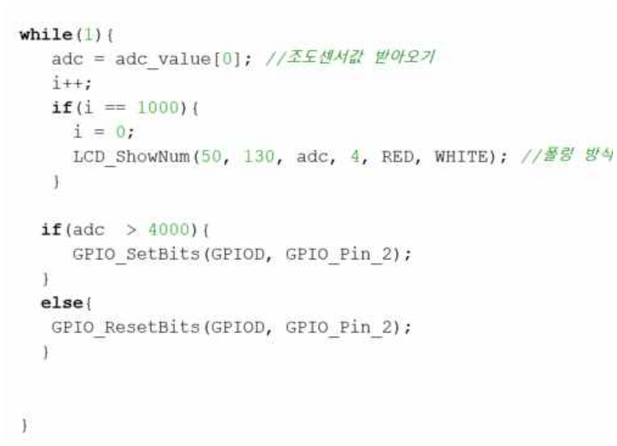
- ADC 채널 설정, ADC Enable/Calibration
- 3 번 채널 사용



```
void ADC_Configure(void){

   ADC_InitTypeDef ADCch3;

   ADCch3.ADC_Mode = ADC_Mode_Independent;
   ADCch3.ADC_ScanConvMode = ENABLE;
   ADCch3.ADC_ContinuousConvMode = ENABLE;
   ADCch3.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
   ADCch3.ADC_DataAlign = ADC_DataAlign_Right;
   ADCch3.ADC_NbrOfChannel = 1;
   ADC_RegularChannelConfig(ADC1,ADC_Channel_3,1,ADC_SampleTime_239Cycles5);
   ADC_Init(ADC1,&ADCch3);

   ADC_DMACmd(ADC1, ENABLE);
   ADC_Cmd(ADC1,ENABLE);

   ADC_ResetCalibration(ADC1);
   while(ADC_GetResetCalibrationStatus(ADC1));

   ADC_StartCalibration(ADC1);
   while(ADC_GetCalibrationStatus(ADC1));

   ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

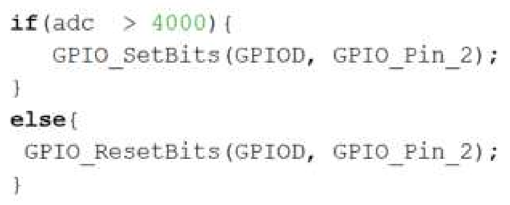
Main

- 조도 센서 값 받아와서 LCD 에 표시
- 값 자주 바뀌는 것을 막기위해 i = 1000 될때만 변경



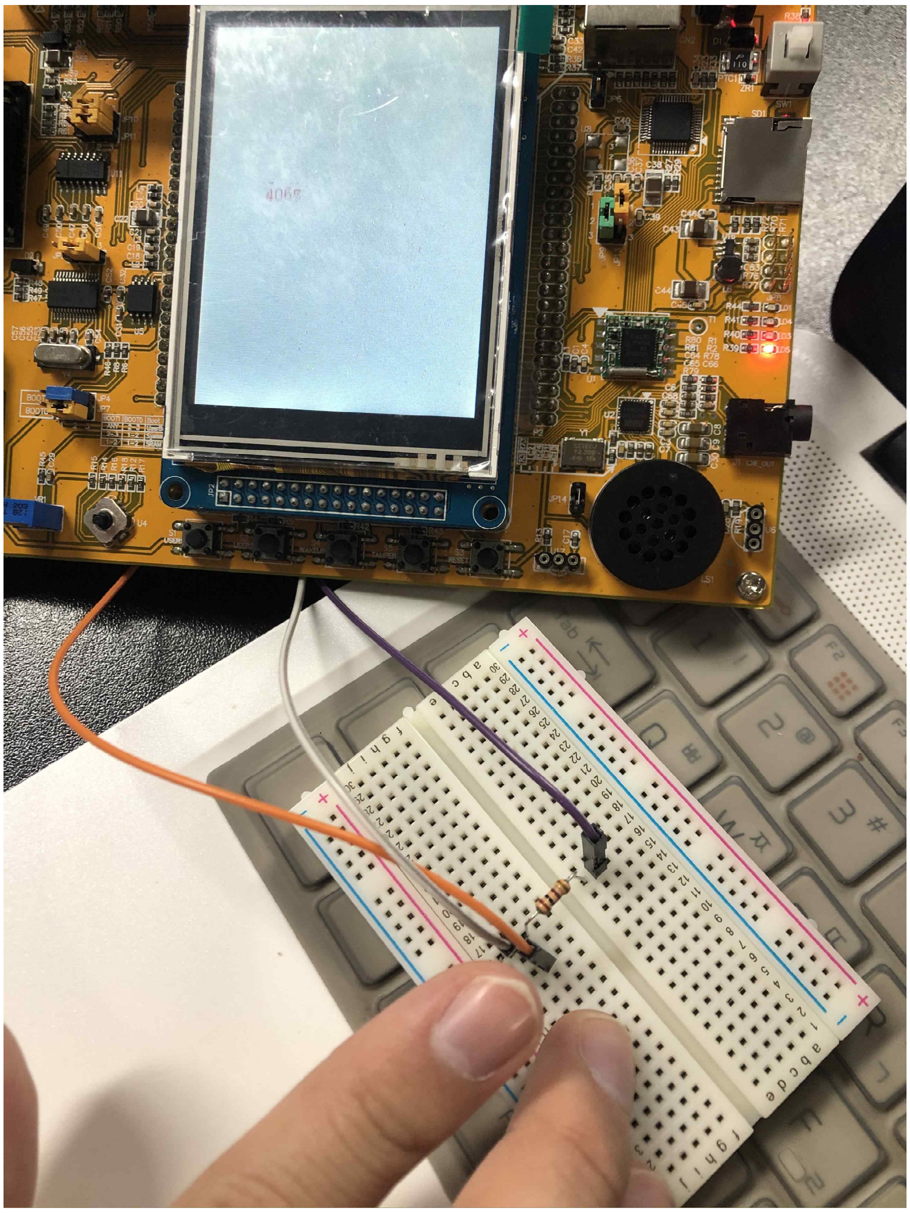
```
while(1){
   adc = adc_value[0]; //조도센서값 받아오기
   i++;
   if(i == 1000){
     i = 0;
     LCD_ShowNum(50, 130, adc, 4, RED, WHITE); //폴링 방식
   }

  if(adc  > 4000){
     GPIO_SetBits(GPIOD, GPIO_Pin_2);
  }
  else{
   GPIO_ResetBits(GPIOD, GPIO_Pin_2);
  }


}
```

- 조도센서 값에 따라 LED 제어

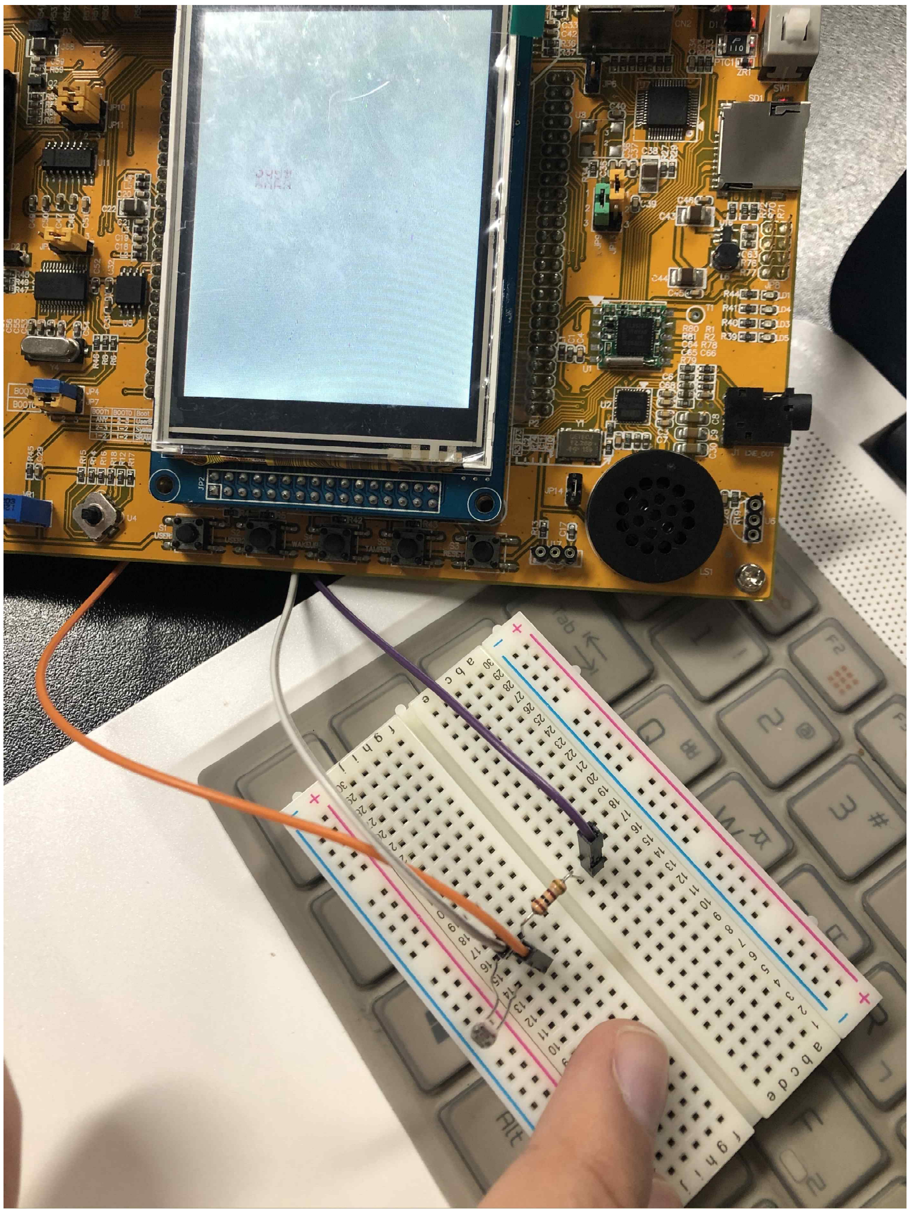


```
if(adc  > 4000){
   GPIO_SetBits(GPIOD, GPIO_Pin_2);
}
else{
 GPIO_ResetBits(GPIOD, GPIO_Pin_2);
}
```

실험결과



```
4068
```



전체 코드

```
#include "stm32f10x.h"
#include "core_cm3.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_adc.h"
#include "misc.h"
#include "lcd.h"
#include "touch.h"

#define ADC1_DR_Address ((u32)0x4001244C)

i                                             n                                             t
color[12]={WHITE,CYAN,BLUE,RED,MAGENTA,LGRAY,GREEN,YELLOW,BROWN,BRRED,G
RAY};
int i = 0;
volatile uint32_t adc_value[1];

void GPIO_Configure() {

   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitTypeDef PA3;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Pin = (GPIO_Pin_2); //LED
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOD, &GPIO_InitStructure);

   PA3.GPIO_Mode = GPIO_Mode_AIN;
   PA3.GPIO_Pin = GPIO_Pin_3; //ADC
   GPIO_Init(GPIOA,&PA3);

}

void RCC_Configure(void){
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    /*TODO : APB2PeriphClockEnable */
     RCC_APB2PeriphClockCmd( // 사용하고자하는 port들의 clock을 enable 한다.
     RCC_APB2Periph_GPIOA  |  RCC_APB2Periph_GPIOD  |  RCC_APB2Periph_ADC1,
ENABLE); //ADC1 레퍼런스 146p에 있음
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}
```

```
void DMA_Configure(void){
    DMA_InitTypeDef DMA_InitStructure;
    DMA_DeInit(DMA1_Channel1);

    DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;

    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&adc_value[0];

    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = 1;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);

    /* Enable DMA1 channel1 */
    DMA_Cmd(DMA1_Channel1, ENABLE);

}
void ADC_Configure(void){

   ADC_InitTypeDef ADCch3;

   ADCch3.ADC_Mode = ADC_Mode_Independent;
   ADCch3.ADC_ScanConvMode = ENABLE;
   ADCch3.ADC_ContinuousConvMode = ENABLE;
   ADCch3.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
   ADCch3.ADC_DataAlign = ADC_DataAlign_Right;
   ADCch3.ADC_NbrOfChannel = 1;
   ADC_RegularChannelConfig(ADC1,ADC_Channel_3,1,ADC_SampleTime_239Cycles5);
   ADC_Init(ADC1,&ADCch3);

   ADC_DMACmd(ADC1, ENABLE);
   ADC_Cmd(ADC1,ENABLE);

   ADC_ResetCalibration(ADC1);
```

```
    while(ADC_GetResetCalibrationStatus(ADC1));

    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}


int main(){
    int adc = 0;
    SystemInit();
   RCC_Configure();
   GPIO_Configure();
   ADC_Configure();
    DMA_Configure();

    LCD_Init();
    LCD_Clear(WHITE);

    while(1){
        adc = adc_value[0]; //조도센서값 받아오기
        i++;
        if(i == 1000){
          i = 0;
          LCD_ShowNum(50, 130, adc, 4, RED, WHITE); //폴링 방식
        }

       if(adc  > 4000){
          GPIO_SetBits(GPIOD, GPIO_Pin_2);
       }
       else{
        GPIO_ResetBits(GPIOD, GPIO_Pin_2);
       }


    }

}
```