2021년 2학기
임베디드 시스템설계 및 실험
12주차 실험보고서

201712159 조현우
201724566 전승윤
201724611 최호진
201924650 박지호

목차

1. 실험 목적

1. DMA 이해 및 실습

2. 실험 과정

1) Datasheet 및 Reference Manual을 참고해 사용할 Port의 RCC, GPIO 설정

2) 조도 센서의 사용을 위해 ADC와 DMA 설정

3) 명시된 조건으로 작동하기 위해 동작 코드 작성

3. 실험 세부 내용

1) 사용할 Port의 RCC, GPIO 설정

먼저 사용할 Port의 RCC enable과 GPIO configuration이 필요하다. 사용하는 Port들은 다음 표와 같다.

이름	종류	포트
조도 센서	Input	PB0
ADC	-	-
DMA	-	-
LCD	Output	PC6, PD13, PD14, PD15

표 1 - Ports

그러므로 RCC configuration에서는 PB0, ADC, DMA를 설정하고, GPIO에서는 값을 받을 PB0 포트를 위해 설정한다.

```
void RCC_Configure(void) // stm32f10x_rcc.h
{
  // TODO: Enable the APB2 peripheral clock using the function 'RCC_APB2PeriphClockCmd'

  /* LCD_CS, LCD_RS, LCD_WR, LCD_RD  */
  /* PC6,    PD13    PD14    PD15*/
  /* lcd.c do RCC Configuration in LCD_Configuration()

  /* ADC12_IN8 */
  /* PB0     */

  /* RCC configuration fo DMA n ADC also need */

  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  /* Alternate Function IO clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}

void GPIO_Configure(void) // stm32f10x_gpio.h
{
  // TODO: Initialize the GPIO pins using the structure 'GPIO_InitTypeDef' and the function 'GPIO_Init'
  GPIO_InitTypeDef GPIO_InitStructure;

  /* ADC PB0 GPIO Configuration */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

그림 1 - RCC and GPIO Configuration

2) ADC 설정

10주차 때 사용했었던 ADC configuration의 내용을 그대로 가져와 사용한다. 그러나 ADC interrupt를 사용하여 value를 받았던 10주차와 달리, DMA를 통해 받아야 하므로 ADC_ITConfig() 함수 대신에 ADC_DMACmd() 함수를 사용한다.

```
void ADC_Configure() {
  ADC_InitTypeDef ADC_InitStructure;

  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = ADC_Channel_8;

  ADC_Init(ADC1, &ADC_InitStructure);

  ADC_Cmd(ADC1 ,ENABLE);
  ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_239Cycles5);
  ADC_DMACmd(ADC1, ENABLE); // Configurate ADC-DMA
  ADC_ResetCalibration(ADC1);

  while(ADC_GetResetCalibrationStatus(ADC1)) ;
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1)) ;
  ADC_SoftwareStartConvCmd(ADC1, ENABLE) ;

}
```

그림 2 - ADC Configuration

3) DMA 설정

DMA는 주변장치가 직접 메모리에 접근해 읽거나 쓸 수 있게 하는 기능이다. 따로 CPU의 개입이 필요없이 접근이 가능하며 Read/Write를 위한 Interrupt 또한 필요 없다.

이 보드에서 사용가능한 DMA는 2종류이며, DMA1은 7개의 channel을 가지고 있으며 DMA2는 5개의 channel을 가지고 있다. 각 사용처에 따라 알맞은 DMA와 채널을 선택하면 된다. 이에 관련된 표는 Reference sheet에 찾아볼 수 있으며 표는 아래 그림과 같다.

Table 78. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I^2S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

그림 3 - DMA1 requests for each channel

Table 79. Summary of DMA2 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC3(1)					ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX			
UART4			UART4_RX		UART4_TX
SDIO(1)				SDIO	
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP		TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1			TIM6_UP/ DAC_Channel 1		
TIM7				TIM7_UP/ DAC_Channel 2	
TIM8	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

그림 4 - DMA2 requests for each channel

그림3을 보면 DMA1의 Channel1이 ADC1로부터 data를 받을 수 있음을 알 수 있다. 그러므로 이번 실험에서 사용해야할 DMA는 DMA1의 channel1이다.

이번 실험은 DMA의 숙지가 목적이므로 stm32f10x_dma.h, stm32f10x_dma.c 파일을 참고하여 코드를 작성했다. 코드에 따르면 DMA는 다른 Configuration과 동일하게 구조체를 통해 DMA_Init을 이용해 설정할 수 있다.

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

  DMA_DeInit(DMA1_Channel1);
  DMA_StructInit(&DMA_InitStructure);
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&value;
  //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_Disable;
  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);
  DMA_Cmd(DMA1_Channel1, ENABLE);
}
```

그림 5 - DMA Configuration

위의 사진은 DMA를 Configuration 하는 코드이다. DMA_DeInit() 은 이 DMA를 Deinitialize하기 위한 함수이다. DMA_StructInit() 은 이 구조체를 모두 default값으로 돌

리기 위한 함수이다.

다음은 꼭 설정해야할 구조체 변수들이다. DMA_PeripheralBaseAddr은 DMA가 값을 받아야할 Source의 base address이다. 이를 지정해주어야 DMA가 원하는 값을 받아올 수 있다. 우리는 ADC로부터 값을 받아와야 하므로 ADC1->DR의 값을 넣어준다. DMA_MemoryBaseAddr는 DMA가 저장해야할 값의 주소를 넣어야 한다. 미리 값을 받을 전역변수를 만든 뒤 이 값의 주소를 여기에 넣어준다.

DMA_Mode는 DMA_Mode_Circular로 설정한다. DMA의 모드는 두 가지로 나뉜다. Normal 모드는 NDT 값 만큼만 Read/Write를 하는 모드이다. 미리 설정해둔 NDT의 값이 0이 되면 DMA는 작동을 중지한다. Circular 모드는 주기적인 값의 전송이 필요할 때 사용하는 모드이다. 이번 실험은 주기적으로 조도 센서의 값을 받아와야 하므로 Circular 모드를 이용한다.

4) 동작 코드 작성

이제 동작을 위한 코드를 작성해야 한다. 조도센서로부터 받아온 값에 따라 LCD 화면을 어둡게 또는 밝게 설정해야 한다. 조도 센서에 플래시로 통해 비출 경우는 어두운 배경을 키며, 플래시를 끌 경우 하얀색을 보여야 한다.

```
void delay(){
  for(int i = 0; i < 1000000; i++){}
}

int main(void)
{
    SystemInit();
    RCC_Configure();
    GPIO_Configure();

    ADC_Configure();
    DMA_Configure();

    LCD_Init();
    LCD_Clear(WHITE);

    while (1) {
      if(value <= 300) {
        LCD_Clear(BLACK);
        LCD_ShowNum(0, 32, value, 10 ,WHITE, BLACK);
      }else {
        LCD_Clear(WHITE);
        LCD_ShowNum(0, 32, value, 10 ,BLACK, WHITE);
      }

      delay();
    }

    return 0;
}
```

그림 6 - delay(), main()

Configuration 작업이 끝난 후 main() 은 while(1) loop에 갇히게 된다. 한번의 loop 동안 조도 센서와 DMA를 통해 받아온 value 값에 따라 if-else 문을 통해 구분해 작동한다. 플래시를 조도센서에 비출 경우 value 값이 300 이하임을 확인했다. 그래서 if-else 기준을 300으로 두어 코드를 작성했다.

또한 delay() 함수를 loop 끝에 두어 너무 주기적으로 값이 변경되는 것을 방지하였다.

4. 실험 결과

그림 7 - Flash Off

LCD 화면에서는 주기적으로 조도 센서의 값을 출력하게 했다. 위의 사진은 조도 센서에 Flash를 껐을 때의 화면이다. value 값이 300 이상이므로 배경이 하얗게 출력되고 있다.

그림 8 - Flash On

위의 사진은 조도 센서에 Flash를 켰을 때의 사진이다. 조도 센서로 부터 받은 값이 300 이하이므로 LCD는 배경을 검정색으로 표시하고, 값을 출력할 글자는 하얀색으로 출력하고 있다.