임베디드 설계 및 실험 12 주차 실험 보고서

목요일 분반 3 조

202011503 강찬미

201606151 김민승

201724425 김민종

201824451 김정호

201724485 배재홍

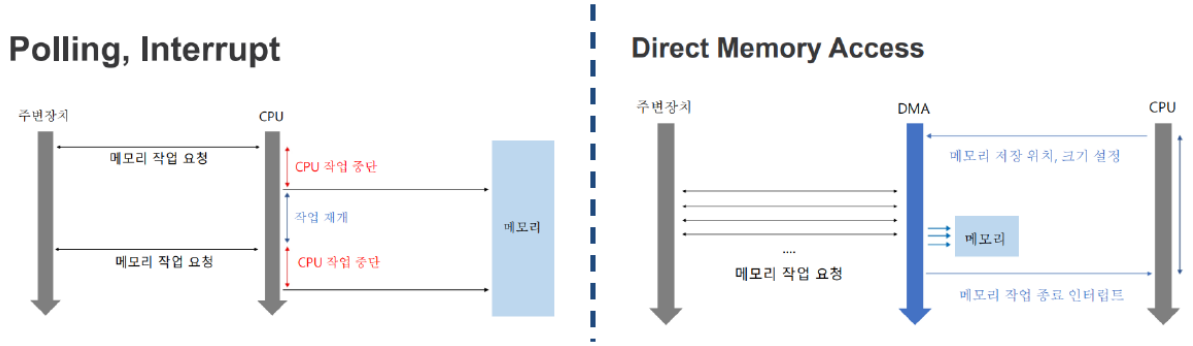
목 표

1. DMA 이해 및 실습

1. 배경 지식

1.1 DMA

DMA 는 Direct Memory Access 의 약자로 주변장치들이 메모리에 직접 접근하여 읽거나 쓸 수 있도록 하는 기능이다. CPU 의 개입 없이 I/O 장치와 기억장치 데이터를 전송할 수 있다. Interrupt 와 달리 별도의 중앙제어장치는 명령을 실행하지 않아도 되며 이에 따라 메모리 처리 Interrupt cycle 만큼 성능이 향상된다.

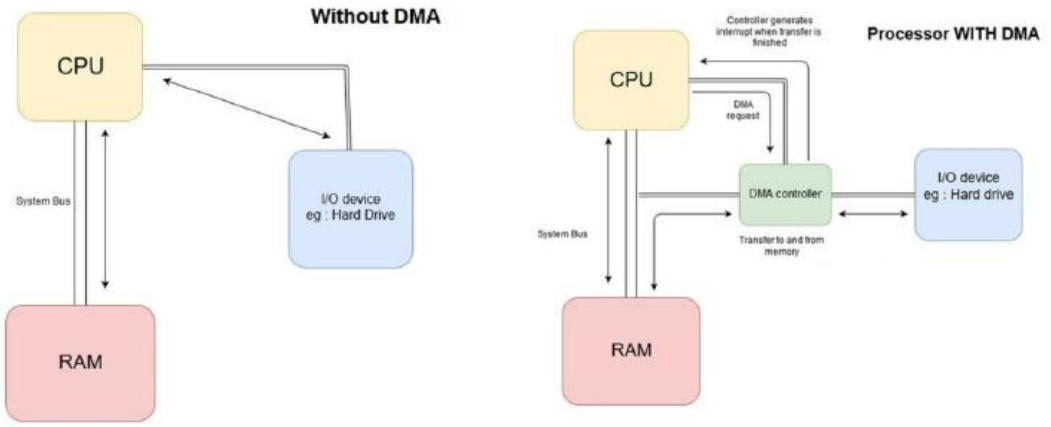


Polling, Interrupt와 DMA cycle 차이

1.2 메모리 접근 방식의 차이

일반적인 메모리 접근 방식은 모든 I/O로의 접근은 CPU를 통해서 수행되고 Data를 전달

할 때마다 CPU가 관여한다. 하지만 DMA 방식은 RAM이 I/O 장치로부터 데이터를 요청하면, CPU는 DMA controller에게 신호(전송 크기, 주소 등)를 보내고 DMA controller가 bus를 통해 RAM 주소로 Data를 주고 받는다. 모든 데이터 전송이 끝나면, DMA controller가 CPU에게 Interrupt 신호를 보내 작업이 끝남을 알린다.



좌: 일반적인 메모리 접근 방식, 우: DMA 방식

1.3 DMA mode

- Normal Mode: DMA controller는 데이터를 전송할 때 마다 NDT(전송할 데이터의 총 용량, 레지스터 값이 0 이되면 데이터 전송 중단)를 감소시킨다. 따라서 데이터를 전송 받고 싶을 때 마다 새롭게 요청이 필요하다.

- Circular Mode: 주기적인 값의 전송(업데이트)이 필요할 때 사용하는 모드이다. NDT 값이 0 이 될 경우 설정한 데이터 최대 크기로 자동 재설정 된다. 따라서 새로 요청하지 않아도 된다.

2. 실험 과정

2.1 세부 실험 내용

1. TFT-LCD 를 보드와 연결

2. 조도센서를 보드와 연결

3. 단 조도 값을 인터럽트 방식이 아닌 DMA 방식으로 얻어와야 함

4. TFT-LCD 에 Team 03, 및 조도 출력

5. 조도가 일정 수치 이상이면 화면에 회색이 출력되고 이하면 흰색이 출력되야 함

2-2. 실험 방법

1. J-Link 와 보드를 컴퓨터에 연결한다.

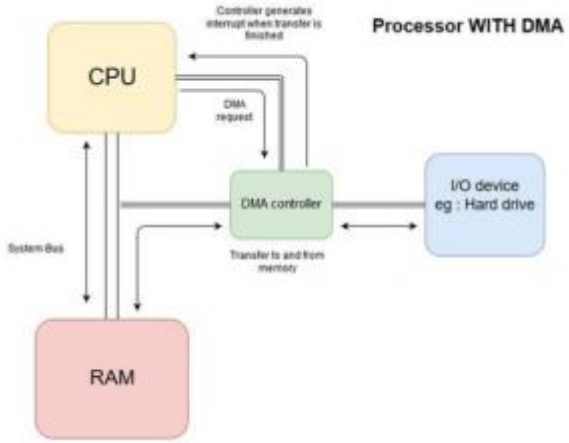
2. IAR EW for Arm 을 실행해 project 생성 및 기본 설정을 한다.

3. 실험에 필요한 라이브러리 파일을 프로젝트에 넣는다.

4. 조도 센서와 lcd 를 보드에 연결한다.

5. 10 주차 실험 코드와 stm32f10x_dma.h(.c) 파일의 내용을 바탕으로 하여 clock 인가, DMA 설정 등 필요한 함수를 정의하고 구현한다.
   - RCC_Configure() : Clock 를 인가해주는 함수
   - Gpio_Configure() : 사용되는 Gpio pin 의 핀번호, In/ouputt Mode, Speed 를 설정해주는 함수
   - ADC_Configure() : 아날로그 값으로 들어오는 조도센서 데이터를 디지털 값으로 바꿔주기 위한 함수
   - DMA_Init2() : 메모리에 직접 접근하여 데이터를 읽는 DMA 방식을 조도센서에 적용시키기 위한 함수
6. 미션 동작을 수행할 수 있도록 main 함수의 while 문을 작성한다.
   - TFT-LCD 에 Team 03, 조도센서 측정 값 출력

   -조도 센서 값이 임계를 지날 시 TFT-LCD 의 바탕색이 하얀색 또는 회색이 되도록설정.


Controller generates interrupt when transfer is finished
Processor WITH DMA
CPU
DMA request
DMA controller
I/O device eg : Hard drive
System Bus
Transfer to and from memory
RAM

- DMA 방식의 예제-

3-2. 실험 결과- 코드

3-1. 전처리 및 RCC/GPIO Configure

```
#include "stm32f10x.h"
#include "core_cm3.h"
#include "misc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_adc.h"
#include "stm32f10x_dma.h"
#include "lcd.h"
#include "touch.h"

volatile uint32_t ADC_Value[1];

void RCC_Configure() {
    //clock Enable
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); // PB0
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE); // ADC1
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE); // DMA1
}

void GPIO_Configure() {
  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; //PIN B0 = ADC_Channel_8
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //AIN = Analog INput
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

읽어온 조도센서 값을 저장할 ADC_Value 를 선언한다.

다음으로, 우리는 ADC1 의 8 번 채널(PIN B0)을 사용하므로 RCC_Configure()에서 GPIOB 와 ADC1 에 clock 을 ENABLE 시켜준다. 또한, 이번 실험에서는 DMA 를 사용하므로 DMA1 의 clock 도 ENABLE 시켜준다.

그리고 GPIO_Configure()를 통해 PIN B0 를 초기화해준다. 조도센서의 값을 읽을 것이므로 Analog Input Mode 로 설정한다.

3-2. ADC Configure

```
void ADC_Configure() {
  ADC_InitTypeDef ADC_InitStructure;

  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //use ADC1 only
  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; //use one channel

  ADC_Init(ADC1, &ADC_InitStructure);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_239Cycles5);
  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);
  }
```

ADC_DMAcmd 를 제외하고는 10 주 차 실험의 코드와 동일하다.

ADC_Init 을 통해 ADC1 을 설정하고, ADC_RegularChannelConfig 를 통해 ADC 채널 8 을 설정해준다.

그 후, ADC_DMACmd 를 통해 ADC1 DMA 요청을 enable 해준다.

ADC_TypeDef* ADCx : ADC1

FunctionalState NewState : ENABLE

ADC_ResetCalibration , ADC_StartCalibration, ADC_SoftwareStartConvCmd 함수를 통해 값을 변환한다.

3-3. DMA_Init2

```
void DMA_Init2() {
  DMA_InitTypeDef DMA_InitStructure;
  DMA_DeInit(DMA1_Channel1); //channel 11
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) &ADC1 ->DR;
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) ADC_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);
}
```

DMA_Init2 를 통해 DMA 를 설정해준다. 우선 DMA_DeInit 을 통해 채널 1 을 초기화한다.
정보를 가져올 곳의 주소이므로 DMA_PeripheralBaseAddr : (uint32_t) &ADC1 ->DR
가져온 정보를 저장할 곳의 주소이므로 DMA_MemoryBaseAddr : (uint32_t) ADC_Value
조도센서 값이 Source 이므로 DMA_DIR : DMA_DIR_PeripheralSRC
Data unit buffer size 를 DMA_BufferSize : 1 로 정한다.
ADC1 에서 조도센서 값 하나만 전달하므로 데이터 전송 후 Peripheral 의 주소를 증가시켜주지 않아도 되기 때문에 DMA_PeripheralInc : DMA_PeripheralInc_Disable
Memory 측의 주소도 증가시켜줄 필요가 없으므로 DMA_MemoryInc : DMA_MemoryInc_Disable
Peripheral 의 Data Width 를 DMA_PeripheralDataSize : DMA_PeripheralDataSize_Word 로 정하고
Memory 의 Data Width 를 DMA_MemoryDataSize : DMA_MemoryDataSize_Word 로 정한다.
정해진 메모리를 모두 사용하면 자동으로 처음으로 초기화하도록 하기 위해 DMA_Mode : DMA_Mode_Circular
SW 우선 순위를 DMA_Priority : DMA_Priority_High 로 정해준다.
Memory to Memory 로 사용되지 않으므로 DMA_M2M : DMA_M2M_Disable
DMA_Init(DMA1_Channel1, &DMA_InitStructure)를 통해 채널 1 을 위의 설정으로 초기화해준다.
DMA_Cmd(DMA1_Channel1, ENABLE)를 통해 DMA1 의 채널 1 을 사용할 수 있도록 해준다.

3-4. main function

3-2. 실험 결과- 코드

3-1. 전처리 및 RCC/GPIO Configure

```
int main() {
  SystemInit();
  RCC_Configure();
  GPIO_Configure();
  ADC_Configure();
  DMA_Init2();

  LCD_Init();
  Touch_Configuration();
  Touch_Adjust();
  LCD_Clear(WHITE);

  uint32_t threshold = 800;
  uint32_t flag = 0;
  uint32_t isGray = 0;

  while(1){
    if (!isGray && ADC_Value[0] > threshold)  flag = 1;
    if (isGray && ADC_Value[0] <= threshold)  flag = 1;

    if (flag) {
        if (isGray) {
            LCD_Clear(WHITE);
            isGray = 0;
        } else {
            LCD_Clear(GRAY);
            isGray = 1;
        }
        flag = 0;
    }

    LCD_ShowNum(20, 100, ADC_Value[0], 4, BLACK, WHITE);
  }
}
```

필요한 함수를 호출해주고 필요한 변수를 정의한다.

threshold : 배경색이 GRAY 가 되는 기준점

flag : 배경색 변경 필요 여부(1 : 필요, 0: 불필요)

isGray : (1 : 현재 배경색이 GRAY, 0 : 현재 배경색이 GRAY 가 아님)

Wihile 문 내에서 미션지에서 요구하는 동작을 수행한다.

현재 배경색이 GRAY 가 아니고(isGray==0), 조도센서 값이 threshold 초과인 경우와

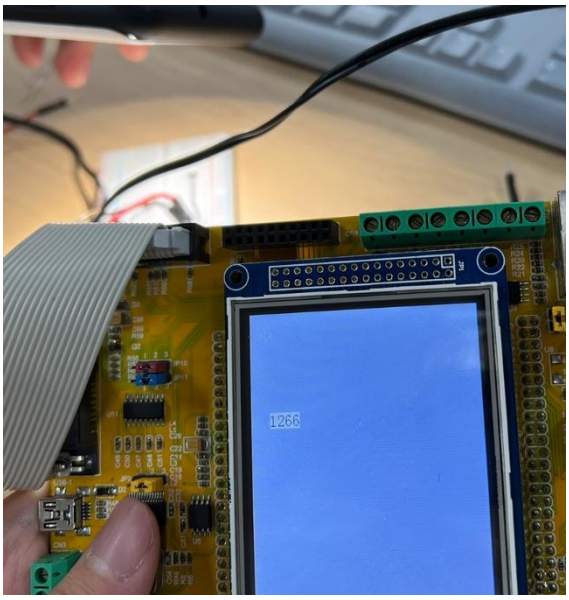
현재 배경색이 GRAY 이고(isGray==1), 조도센서 값이 threshold 이하인 경우 배경색을 바꿔야하므로 flag 를 1 로 둔다.

만약 배경색 변경이 필요하고(flag==1), 현재 배경색이 GRAY 라면(isGray==1) 배경색을 WHITE 로 바꾸고 isGray=0 으로 바꾼다.

반대로 현재 배경색이 GRAY 가 아니라면(isGray==0) 배경색을 GRAY 로 바꾸고 isGray=1 로 바꾼다

그 후 flag=0 으로 설정하고 LCD_ShowNum 을 통해 TFT-LCD 에 조도센서 값을 출력한다.

3-5. 동작결과 및 회로 사진



Threshold 이하인 경우

4. 결론

이번 실험에서는 DMA 동작 방법을 이해하고 직접 구현하는 방법을 배웠다.

기존에는 보드와 연결한 각종 IO 장치들로 보드를 제어하고 입출력을 할 때, 보드의 CPU 를 거치는 인터럽트 방식을 사용해왔다. 그러나 이번 실험에서 사용한 DMA 를 활용하면 이름에서 알 수 있듯,(Direct Memory Access) 모든 과정에서 CPU 가 개입하는 것이 아니라 일부 과정에서만 개입하여 BUS 를 통해 RAM 주소에 직접 접근하여 데이터를 주고 받음으로써 기존 인터럽트 방식에서 CPU 가 메모리 처리에 필요했던 사이클 만큼 성능을 향상시킬 수 있음을 알게 되었다.

DMA 를 사용함에 있어서 각 설정 옵션의 역할과 파라미터들에 대한 이해에 어려움이 있었는데 제공된 헤더와 정의된 함수, 레퍼런스들을 더 꼼꼼히 살펴 어떤 원리, 방식으로 동작하는지, 각 옵션 값들에 어떤 의미가 있는 지 확실히 이해하고 알고 사용해야 함을 다시금 깨달을 수 있었다.