실험 결과 보고서
11주차 DMA

실험 일자 : 2024. 11. 12.

실험 참여 : 3조 강태진, 김기윤, 김도완, 임성표

차례

I. 실험 목적

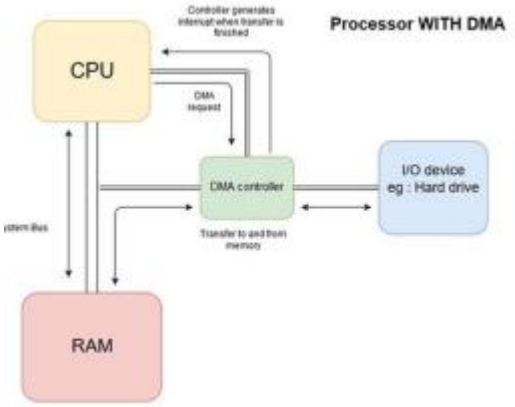
i) DMA 동작 방법의 이해

ii) DMA 구현

II. 세부 목표

i) DMA

- Direct Memory Access (DMA)
- CPU의개입없이 I/O장치와 기억장치 데이터를 전송하는 접근 방식
- Interrupt 와 달리 별도의 중앙제어장치는 명령을 실행할 필요가 없음
- 메모리 처리 Interrupt의 사이클만큼 성능의 향상
- 주변 장치들이 메모리에 직접 접근하여 읽거나 쓸 수 있도록 하는 기능



< DMA의 작동 방식 >

ii) DMA 접근 방식

- RAM이 I/O장치로부터 데이터가 필요해지면, CPU는 DMA 컨트롤러에게 신호(전송 크기, 주소 등등)를 보냄
- DMA 컨트롤러가 RAM 주소로 데이터를 bus를 통해 주고받음
- 모든 데이터 전송이 끝나면, DMA Controller가 CPU에게 Interrupt 신호를 보냄
-

iii) DMA Channel

- 모듈은 DMA Controller의 DMA 채널을 통해 메모리 R/W
- STM32보드 DMA 채널은 총 12개 (DMA1 채널 7개, DMA2 채널 5개)
- 한 DMA의 여러 채널 사이 요청은 Priority에 따라 동작(4 level: very high, high, medium, low)
- Peripheral-to-memory, memory-to-peripheral, and peripheral-to-peripheral 전송

iv) DMA Mode

- Normal Mode
  - ■ DMA Controller는 데이터를 전송할 때 마다 NDT 값을 감소시킴
  - ■ NDT는 DMA를 통해 전송할 데이터의 총 용량을 의미하며 레지스터의 값이 0이 되면 데이터 전송 중단
  - ■ 데이터 전송을 받고 싶을 때 마다 새롭게 요청이 필요
- Circular Mode
  - ■ 주기적인 값의 전송(업데이트)이 필요할 때 사용하는 모드
  - ■ NDT값이 0이 될 경우 설정한 데이터 최대 크기로 재설정됨

v) DMA Controller

- 주변 장치의 Request Signal의 발생
  - ■ DMA Controller 에서 우선순위 설정 및 요청에 대한 서비스 제공
  - ■ Request / ACK 방식을 통한 주변 장치와 DMA Controller 간 통신

III. 실험 과정

i) DMA 및 ADC 를 사용하여 1개의 조도센서 값을 받아오도록 구현

- 인터럽트 및 ADC value 가져오는 함수 사용 금지
- DMA 이용한 변수 값만 사용

ii) 읽은 조도센서 갑을 TFT-LCD에 출력

iii) 평상시 TFT-LCD 배경색 흰색, 조도센서에 스마트폰 플래시로 비출 때 TFT-LCD 배경색 회색으로 설정

- 배경색 바꾸면 글자도 사라지므로 배경 바꾸고 조도 값 띄우기

iv) 조도센서 값 threshold 는 각자 실험적으로 결정

IV. 실험 시행

i) RCC_Configure

```
void RCC_Configure(void)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
}
```

<RCC_Configure 내부 코드>

- RCC_APB2PeriphClockCmd 함수를 사용해서, 사용할 포트의 클럭을 활성화

ii) GPIO_Configure

```
void GPIO_Configure(void)
{
  // PA5
  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

<GPIO_Configure 내부 코드>

- GPIO_InitTypeDef GPIO_InitStructure; 코드를 통해서, GPIO핀을 설정하는 구조체를 호출

iii) ADC_Configure

```
void ADC_Configure(void)
{
  ADC_InitTypeDef ADC_InitStruct;

  ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStruct.ADC_ScanConvMode = DISABLE;
  ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStruct.ADC_NbrOfChannel = 1;

  ADC_Init(ADC1, &ADC_InitStruct);

  //chanel 5 setting
  ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 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_Configure 내부 코드>

- ADC
- 아날로그 값을 입력 받기 위해 ADC 관련 설정하는 부분
- - ADC_InitTypeDef 구조체 선언하고 초기화
- - ADC_RegularChannelConfig 함수사용하여 ADC1의 Regular 그룹에서 사용할 채널 5 설정
- - ADC_DMACmd 함수로 DMA 활성화

iv) DMA Configuration

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

  DMA_Instructure.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
  DMA_Instructure.DMA_MemoryBaseAddr = (uint32_t)&ADC_Value[0];

  DMA_Instructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_Instructure.DMA_BufferSize = 1;
  DMA_Instructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_Instructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
  DMA_Instructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
  DMA_Instructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
  DMA_Instructure.DMA_Mode = DMA_Mode_Circular;
  DMA_Instructure.DMA_Priority = DMA_Priority_High;

  DMA_Instructure.DMA_M2M = DMA_M2M_Disable;

  DMA_Init(DMA1_Channel1, &DMA_Instructure);

  DMA_Cmd(DMA1_Channel1, ENABLE);
}
```

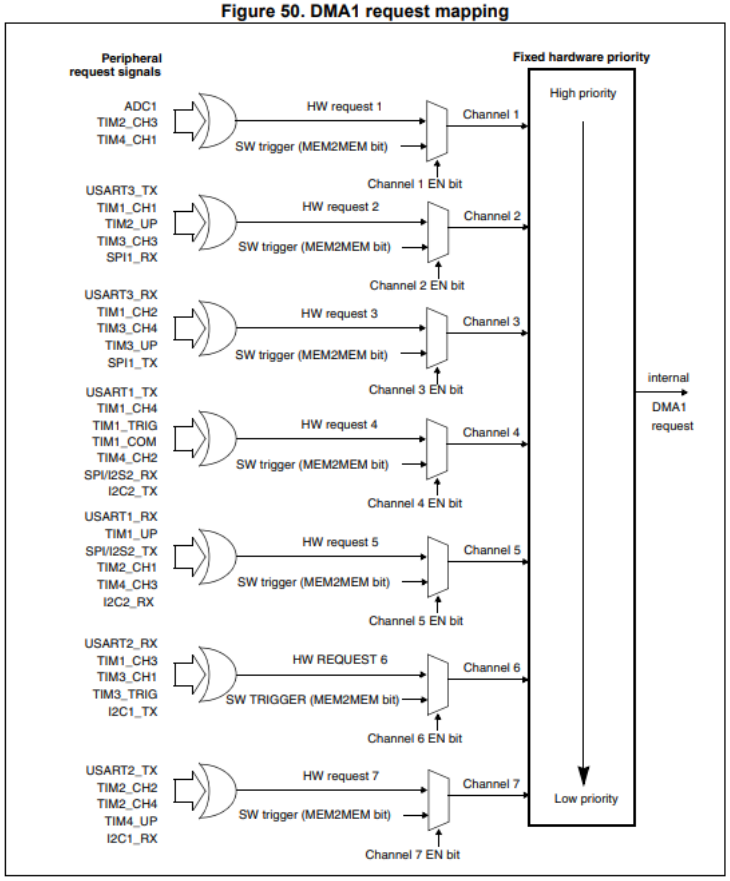
<DMA_Configure 내부 코드>

<stm32f10x_usart.c에서 USART_Init 함수를 정의>

- DMA 관련 설정하는 부분
- 주소 설정:
  - DMA_PeripheralBaseAddr: ADC1의 데이터 레지스터 주소(&ADC1->DR)를 지정
  - DMA_MemoryBaseAddr: DMA가 전송할 데이터가 저장된 메모리의 주소로 ADC_Value 배열의 첫 번째 원소 주소(&ADC_Value[0])를 사용
- 전송 크기 및 주소 증가 설정:
  - DMA_BufferSize: 한 번의 전송에서 처리할 데이터의 개수로 1로 설정
  - DMA_PeripheralInc 및 DMA_MemoryInc: 주소 증가를 비활성화하여 주소를 고정
- 모드 및 우선순위 설정:
  - DMA_Mode: 순환 모드로 설정하여 전송이 계속해서 반복
  - DMA_Priority: 높은 우선순위로 DMA 전송을 설정

- 초기화 및 활성화:
  - DMA_Init 함수를 호출하여 DMA1 채널 1에 대한 초기 설정을 적용
  - DMA_Cmd 함수를 사용하여 DMA1 채널 1을 활성화
  - ADC1이 channel 1을 통해서 DMA1 request로 output
  - 따라서 DMA1을 사용

Figure 50. DMA1 request mapping



<그림6 stm32_ReferenceManual에서 DMA1 resqust mapping 회로도>

v) Main.c

```
int main() {

  SystemInit();
  RCC_Configure();
  GPIO_Configure();
  ADC_Configure();
  DMA_Configure();

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

  while(1){
    delay();

    flag = (ADC_Value[0] < THRESHOLD) ? 1: 0;
    if (flag) {
      LCD_Clear(GRAY);
      LCD_ShowNum(50, 150, ADC_Value[0], 10, WHITE, GRAY);
    } else {
      LCD_Clear(WHITE);
      LCD_ShowNum(50, 150, ADC_Value[0], 10, GRAY, WHITE);
    }
  }
  return 0;
}
```

<main 함수 내부 코드>

- Flag를 ADC로 읽은 값을 기준으로 THRESHOLD(3750)값과 비교해서 값보다 작으면 1, 값보다 크면 0으로 설정
- Flag가 1이면 회색 배경에 흰색으로 ADC 값 출력
- Flag가 0이면 흰색 배경에 회색으로 ADC 값 출력

v) 실험 결과

- **조도 센서에 빛을 강하게 비추지 않은 경우**

조도 센서에 빛을 비추지 않으면 저항이 작아져서 ADC 값이 3995 가 되어서 설정한 THRESHOLD 값 보다 크기 때문에, Flag 가 0 으로 설정 흰색 배경에 회색으로 ADC 값 3995 출력

- **조도 센서에 빛을 강하게 비춘 경우**

- 조도센서에 빛을 비추면, ADC값이 THRESHOLD값보다 더 작아져서 Flag가 1로 설정, 회색 배경에 흰색으로 ADC값 3707 출력

V. 참고문헌

- stm32_Datasheet.pdf
- stm32_ReferenceManual.pdf
- STM32107VCT6_schematic.pdf
- 11주차 DMA.pdf
- 11주차 DMA 미션지.pdf