REPORT

임베디드 실습 및 실험 (002)

10조 11주차 실험 결과보고서

전기컴퓨터공학부	201824446	김윤재
정보컴퓨터공학부	202055558	송세연
정보컴퓨터공학부	2020555889	임연후
바이오소재과학과	201845626	최이한

2022.11.09

◆ 실험 목적

- DMA 동작 방법을 이해하고 구현한다

◆ 실험 목표

- DMA 및 ADC 를 사용하여 1개의 조도센서 값을 받아오도록 구현
- 인터럽트 및 ADC value 가져오는 함수 사용 금지
- DMA 이용한 변수 값만 사용하기
- 읽은 조도센서 값을 TFT-LCD에 출력
- 평상시 TFT-LCD 배경색 WHITE, 조도센서에 스마트폰 플래시로 비출 때 TFT-LCD 배경색 GRAY
- 배경색 바꾸면 글자도 사라지므로 배경 바꾸고 조도 값 띄우기
- 조도센서 값 threshold는 각자 실험적으로 결정

◆ 실험 개념

- **Direct Memory Access (DMA)** : 주변장치들이 메모리에 직접 접근하여 읽거나 쓸 수 있도록 하는 기능으로 CPU 의 개입 없이 I/O 장치와 기억장치 데이터를 전송하는 접근 방식이다. 이는 Interrupt 와 달리 별도의 중앙제어장치 명령을 실행할 필요가 없고, 메모리 처리 Interrupt 의 사이클 만큼 성능이 향상된다.

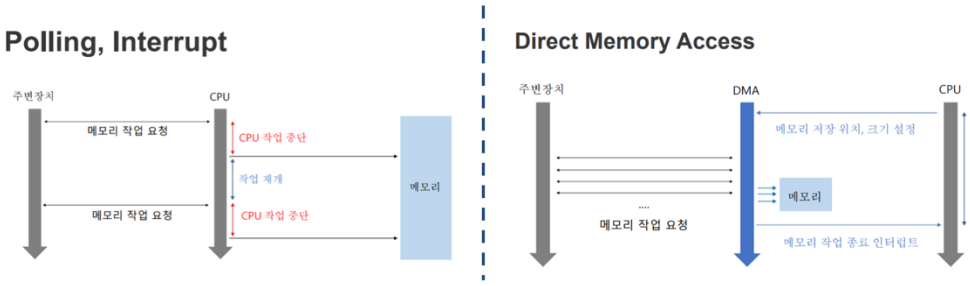

Figure 1 Interrupt 와 DMA 방식의 차이점

- STM32 보드에서 DMA 채널은 총 12개로 구성되어있다. (DMA1 채널 7개, DMA2 채널 5개)
- 한 DMA 의 여러 채널 사이 요청은 Priority 에 따라 동작한다. (4 level : very high, high, medium, low)

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/I²S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I²C	-	-	-	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

Figure 2 DMA1 채널 Summary

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

Figure 3 DMA2 채널 Summary

◆ 실험 과정

- 먼저 8주차 실험 내용과 비슷하게 조도 센서를 이용하여 진행하는 실험으로, 8주차의 프로젝트를 복제하여 실험을 진행하였다.

```
void RCC_Configure(void)
{
   /* Alternate Function IO clock enable */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC, ENABLE); // ADC1, port C RCC ENABLE
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}
```

Figure 2 RCC_Configure(void) 코드

- 조도센서의 아날로그 값을 디지털로 바꾸기 위해 ADC1 과 조도센서를 연결하기로 설정한 APB2

의 GPIOC 를 ENABLE 시킨다.

- DMA1 채널을 사용하기 위해 AHB 의 DMA1 을 ENABLE 하고, AFIO 를 사용하기 위해 APB2 의 AFIO 또한 ENABLE 시킨다.

```
void GPIO_Configure(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; // 아날로그 입력 설정
   GPIO_Init(GPIOC, &GPIO_InitStructure);        // C0포트 활성화, 조도센서를 PC0에 연결할 계획
}
```

Figure 3 GPIO_Configure(void) 코드

- 8주차와 동일하게 GPIO_Configure 을 설정해주었다. 아날로그 입력을 사용하므로 GPIO_Mode 를 GPIO_Mode_AIN 으로 설정해주었다.

- 또한 8주차와 동일하게 GPIO_Speed 는 max speed 인 50MHz 로 설정하였다.

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Figure 6 Port configuration register low (GPIOx_CRL)

```
typedef enum
{ GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;

#define IS_GPIO_MODE(MODE) (((MODE) == GPIO_Mode_AIN) || ((MODE) == GPIO_Mode_IN_FLOATING) || \
                            ((MODE) == GPIO_Mode_IPD) || ((MODE) == GPIO_Mode_IPU) || \
                            ((MODE) == GPIO_Mode_Out_OD) || ((MODE) == GPIO_Mode_Out_PP) || \
                            ((MODE) == GPIO_Mode_AF_OD) || ((MODE) == GPIO_Mode_AF_PP))
```

Figure 7 GPIO_Mode_AIN

```c
void DMA_Configure(void)
{
    DMA_InitTypeDef DMA_InitStruct;
    /*-------------- Reset DMA init structure parameters values ------------------*/
    /* Initialize the DMA_PeripheralBaseAddr member */
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
    /* Initialize the DMA_MemoryBaseAddr member */
    DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)&ADC_Value[0];
    /* Initialize the DMA_DIR member */
    DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC;
    /* Initialize the DMA_BufferSize member */
    DMA_InitStruct.DMA_BufferSize = 1;
    /* Initialize the DMA_PeripheralInc member */
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    /* Initialize the DMA_MemoryInc member */
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Disable;
    /* Initialize the DMA_PeripheralDataSize member */
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
    /* Initialize the DMA_MemoryDataSize member */
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
    /* Initialize the DMA_Mode member */
    DMA_InitStruct.DMA_Mode = DMA_Mode_Circular;
    /* Initialize the DMA_Priority member */
    DMA_InitStruct.DMA_Priority = DMA_Priority_High;
    /* Initialize the DMA_M2M member */
    DMA_InitStruct.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStruct);
    DMA_Cmd(DMA1_Channel1, ENABLE);
}
```

Figure 8 DMA_Configure(void)

- 조도센서의 값을 읽어들이는 ADC1의 Data Register 의 주소값을 Peripheral Base Address 로 설정하고 조도센서의 값을 저장하는 전역변수 ADC_Value 의 주소값을 Memory Base Address 로 설정하였다.

```c
volatile uint32_t ADC_Value[1];
```

Figure 9 volatile 키워드 이용 코드

- Volatile : 전역 변수로 선언한 ADC 값을 저장할 공간을 항상 참조할 수 있도록 volatile 키워드를 이용하였다.

- DIR 은 주변장치에서 값을 읽어 오기에 DMA_DIR_PeripheralSRC 로 설정하였고, 입력받는 아날로그 값이 한 개이므로, 버퍼의 사이즈를 1로 설정하였다. Peripheral increment mode와 Memory increment mode는 Disable 하였다. Peripheral 과 Memory 의 버퍼의 크기는 ADC_Value[0] 의 메모리 크기인 Word Size(4bytes) 로 설정하였다.

- DMA_Mode 는 ADC 값을 주기적으로 업데이트 하기 위해 Circular Mode 로 설정하였다.

- DMA 의 Priority 는 채널이 하나이므로 임의로 High 로 설정하였고, Memory to Memory 는 Disable 하였다.

- DMA_InitTypeDef 구조체에 설정값을 넣은 뒤 DMA_Init 을 이용해 DMA1_Channel1 에 설정을 하였고, DMA_cmd 로 DMA1_Channel1 을 ENABLE 하였다.

```
    while (ADC_GetCalibrationStatus(ADC1));
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    ADC_DMACmd(ADC1, ENABLE); // DMA ENABLE 처리
}
```

Figure 10 DMA ENABLE

- ADC mode는 ADC_Mode_Independent 로 선언하고 단일 채널을 사용할 것이므로 ScanConvMode 는 DISABLE 하였다. 한 번의 트리거로 한 채널의 샘플링을 하기 위해서 ContinuousConvMode를 enable하였으며, 외부 입력핀을 사용하지 않으므로 ExternalTrigConv를 none으로 설정해주었다. DataAlign은 default 값인 ADC_DataAlign_Right로, 채널은 하나이므로 NbrOfChannel 을 1로 선언하였다.

- ADC_RegularChannelConfig 함수를 이용해 채널 우선순위를 10채널 단독사용으로 설정해주었다.

- ADC_ResetCalibration함수로 ADC1 reset calibaration 레지스터를 활성화 한 뒤 레지스터 상태를 확인한 후 StartCalibration 함수를 이용해 calibration을 진행한다. Calibration이 종료되면 Software conversion을 실행한후 DMA를 연결한다.

- ADC 설정 : Interrupt 를 사용하지 않아야 한다는 조건이 있고, DMA 를 이용해야 하므로 ADC_ITConfig 함수 대신 ADC_DMACmd 함수를 사용해야한다. 따라서 해당 함수를 사용해 DMA 를 ENABLE 해주었다.

```
int main(void)
{

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

    LCD_Init();        // LCD 초기화 작업 시행
    LCD_Clear(WHITE); // LCD 배경 WHITE 초기화 작업 시행
```

Figure 11 main 코드

- 8주차와 마찬가지로 LCD 초기화 및 배경 초기화를 진행하여 화면을 WHITE 로 출력하였다.

- DMA 를 사용하였기에 DMA_Configure() 함수를 호출해주었다. 나머지 RCC, GPIO, ADC 함수도 호출하였다.

```
while (1)
{
   if (flag && ADC_Value[0] > 3000)
   {
      LCD_Clear(WHITE); // LCD 배경 초기화 WHITE
      flag = 0;
   }
   else if (!flag && ADC_Value[0] <= 3000)
   {
      LCD_Clear(GRAY); // LCD 배경 초기화 GRAY
      flag = 1;
   }
   LCD_ShowNum(40, 100, ADC_Value[0], 4, BLUE, WHITE); // 조도센서 값 출력
}
return 0;
```

Figure 12 조도센서 값 구간에 따른 배경색 변경 코드

- 코드의 가장 상단에 flag 값을 1로 선언하였는데, while 문에서 배경색이 초기화 되는 것을 방지하기 위해 flag 를 사용하였다.
- 핸드폰 플래시를 비추지 않는 초기 상태에서는 초기에 선언한 int flag = 1 로 flag 가 if문에서 !flag && ADC_Value[0] > 3000 에서 true 라면 LCD 화면을 WHITE 로 출력하고, 조도센서 값이 BLUE 로 출력되게 한다. 이후 flag 값을 0으로 변경하였다.
- 핸드폰 플래시를 비추어서 !flag && ADC_Value[0] <= 3000 에서 true 라면 배경색을 GRAY 로 변경하여 출력하고, 조도센서 값이 BLUE 로 출력되게 한다. 이후 플래시를 껐을 때 배경색이 WHITE 로 바뀌도록 flag 값을 1로 변경해주는 작업을 하였다.
- a value of type "uint32_t volatile *" cannot be assigned to an entity of type "uint32_t" 와 같은 에러가 뜰 시 변수명 앞에 (uint32_t) 로 형 변환하여 에러 처리를 하라는 실험자료가 주어졌는데, 본 실험을 진행할 때는 위의 에러가 발생하지 않았다.

◆ 실험 결과

- 프로그램을 실행시키고 플래시를 비추면 조도센서의 값이 지정한 Threshold 인 3000 이하로 출력되며, 배경색이 GRAY로 출력된다.

Figure 13 조도센서 3000 이하의 범위에서 배경색이 GRAY 로 출력

- 플래시를 끄고 나면 조도센서의 값이 3000 이상으로 올라가는 것을 확인할 수 있고, WHITE 배경 위에 조도센서 값이 출력됨을 확인할 수 있다.

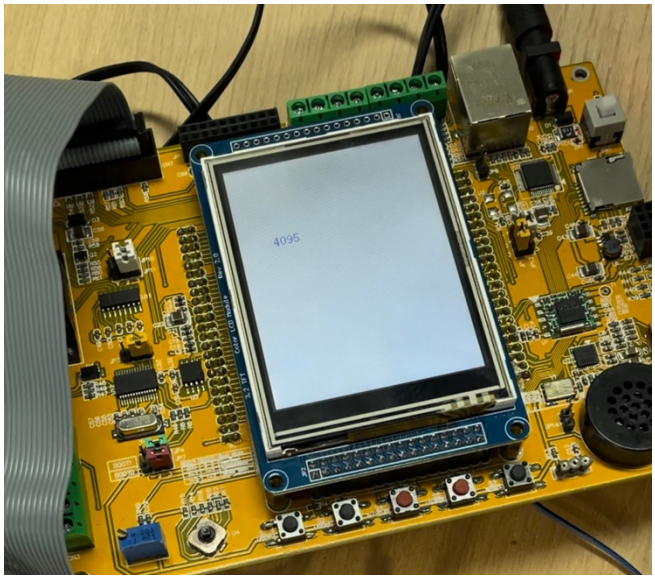

Figure 14 조도센서 3000 초과의 범위에서 배경색이 WHITE 로 출력