2016년 2학기
컴퓨터하드웨어 실험 보고서
7조 12주차

한성희 이호욱 한경수

Table Of Contents

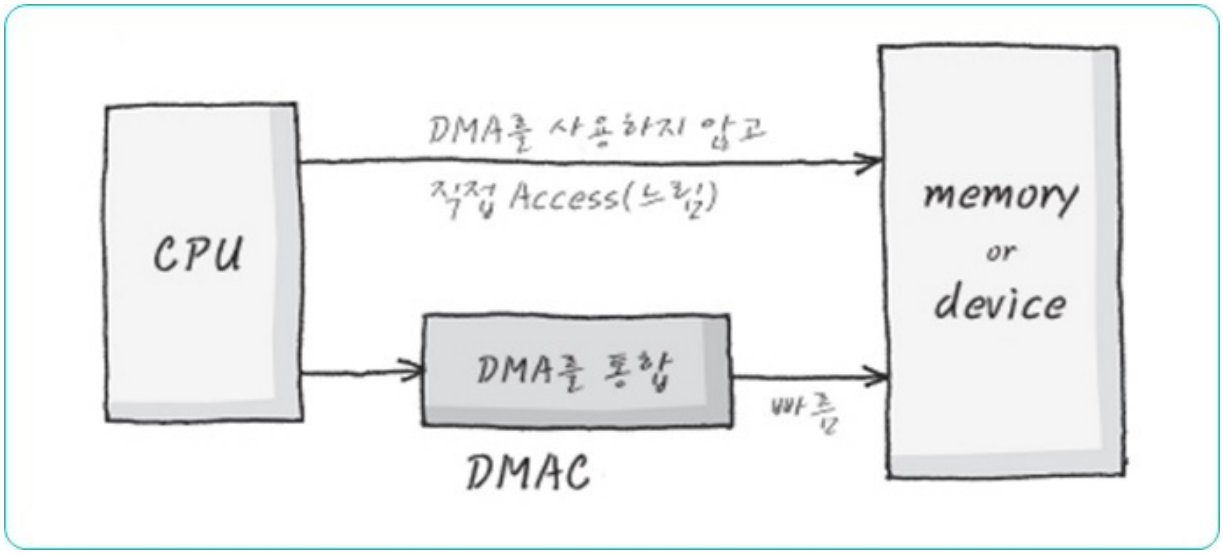
개요

DMA 기능에 대한 이해 및 사용.

기본 개념 (배경 지식)

DMA

DMA (Direct Memory Access)



DMA 란, CPU 개입 없이 I/O device 와 memory device 사이에 data 전송을 가능하게 해주는 것으로 속도가 빠르다.

program 수행 I/O 를 위한 interrupt 를 최소화하여 컴퓨터 효율을 높인다. CPU 와 별개로 분리되어 처리하며 disk, printer, tape-drive 등에 이용된다.

DAM controller 는 system bus 의 권한을 얻어 원하는 data 를 저장하고 bus 의 권한을 반환 후, 개별적으로 처리한다. 또, CPU 는 DMA 의 상태 정보 및 제어 정보만 주고 받는다.

Interrupt 호출이 많은 기능을 이용할 때, DMA 방식을 사용하는 것이 더 효율적이다. (e.g. 조도센서)

DMA 와 interrupt 를 혼합해서 사용 가능하다.

< DMA 동작 방식 >

1. CPU 가 DMA controller 를 초기화. (memory 시작주소, 크기, I/O device 번호, I/O 선택 등)
2. I/O device 가 DMA 를 요청.
3. DMA controller 가 bus 를 CPU 에게 요청.
4. CPU 가 bus 승락. (Grant)
5. DMA controller 가 DMA 승락.
6. I/O device 와 memory 사이의 자료전송.
7. DMA controller 가 DMA 완료 interrupt 를 CPU 에게 보냄.

일반 memory 제어방식 vs DMA

< 일반 memory 제어방식 >

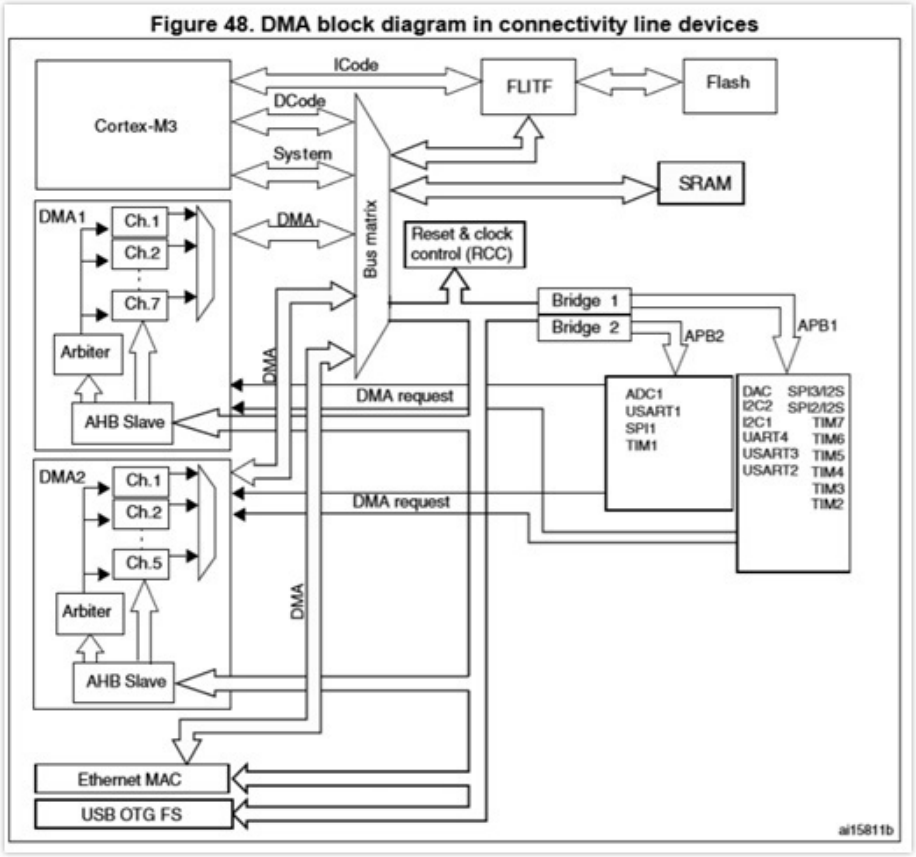
- CPU 가 I/O controller 에 명령을 보내고 CPU 는 다른 작업을 수행.
- controller 는 I/O device 를 제어하여 I/O 명령을 수행.
- I/O 명령 수행이 완료되면 controller 는 CPU 로 interrupt signal 을 보냄.

- CPU 는 interrupt signal 을 받는 즉시 원래의 program 으로 돌아와서 수행을 계속함.

< DMA >

- CPU 가 DMA 에 명령을 보냄.
- DMA 는 CPU 로 BUS REG signal 을 보냄.
- DMA 가 memory 에서 data 를 읽어 disk 에 저장함.
- 전송할 data 가 남아있으면 위의 과정을 반복.
- 모든 data 전송이 끝나면 CPU 에게 INTR signal 을 보냄.

DMA block diagram



다음은 DAM block diagram 으로 원하는 기능의 bus 와 channel 을 선택하여 DMA 를 구현할 수 있다.

- DMA controller : system bus 를 cortex-M3 와 공유하여 직접 memory 전송을 수행한다.
- DMA request 는 CPU 및 DMA 가 동일한 memory 나 peripheral 를 대상으로 할 때, 일부 bus cycle 동안 system bus 에 대한 CPU access 를 중지할 수 있다.
- Bus Matrix 는 Round-robin scheduling 을 구현하므로 CPU 의 system bus 대역폭의 절반 이상을 보장한다.

DMA channel

7개의 DMA1 과 5개의 DMA2 channel 로 구성되어 있다. 각 channel 은 고정 주소에 위치한 peripheral register 와 memory address 사이의 DMA 전송을 처리한다.

동시에 하나의 channel/request 만 동작한다. data 크기는 programming 하고 pointer 를 증가시킨다. 다음 전송 address 는 선택한 data 크기에 따라 이전 address에서 1, 2 또는 4 씩 증가한다.

DMA mode

- circular mode : 순환 buffer 및 연속 data 의 흐름에 대한 handling 이 가능하다. (e.g. ADC scan mode)
- Normal mode : 전송할 data 의 수가 0 이 되면 stream 이 disable 된다.

추가적으로 data 전송시에 필요한 정보들은 뒤에 나오는 register 값으로 set 해주면 된다.

DMA 관련 Standard Peripheral Library 함수

```
DMA_InitTypeDef DMA_InitStructure;
RCC_AHBPeriphClockCmd(                          );

// DMA1 channel1 configuration ----------------------
DMA_DeInit(              );
DMA_InitStructure.DMA_PeripheralBaseAddr =
DMA_InitStructure.DMA_MemoryBaseAddr =
DMA_InitStructure.DMA_DIR =
DMA_InitStructure.DMA_BufferSize =
DMA_InitStructure.DMA_PeripheralInc =
DMA_InitStructure.DMA_MemoryInc =
DMA_InitStructure.DMA_PeripheralDataSize =
DMA_InitStructure.DMA_MemoryDataSize =
DMA_InitStructure.DMA_Mode =
DMA_InitStructure.DMA_Priority =
DMA_InitStructure.DMA_M2M =
DMA_Init(                                    );

// Enable DMA1 Channel1
DMA_Cmd(                      );
```

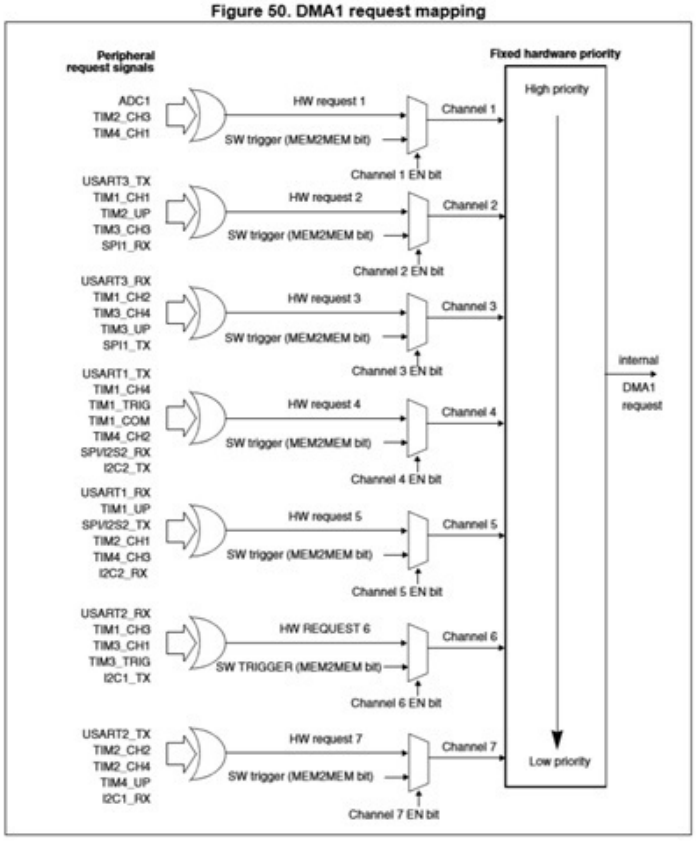
- DMA_PeripheralBaseAddr : DMA 를 사용할 peripheral 와 memory 간의 변수 address 설정.
- DMA_MemoryBaseAddr : 변수를 통해 실제로 저장될 memory address.
- DMA_BufferSize : 변수에 저장할 memory 크기.
- DMA_MemoryDataSize : 변수에 저장될 data 크기.

DMA register

- DMA_CPARx : register address 를 저장.
- DMA_CMARx : memory address 를 저장.
- DMA_CCNDTRx : 전송할 data 개수를 저장.
- DMA_CCRx : 우선순위 지정.

DMA request mapping & 우선순위

Figure 50. DMA1 request mapping



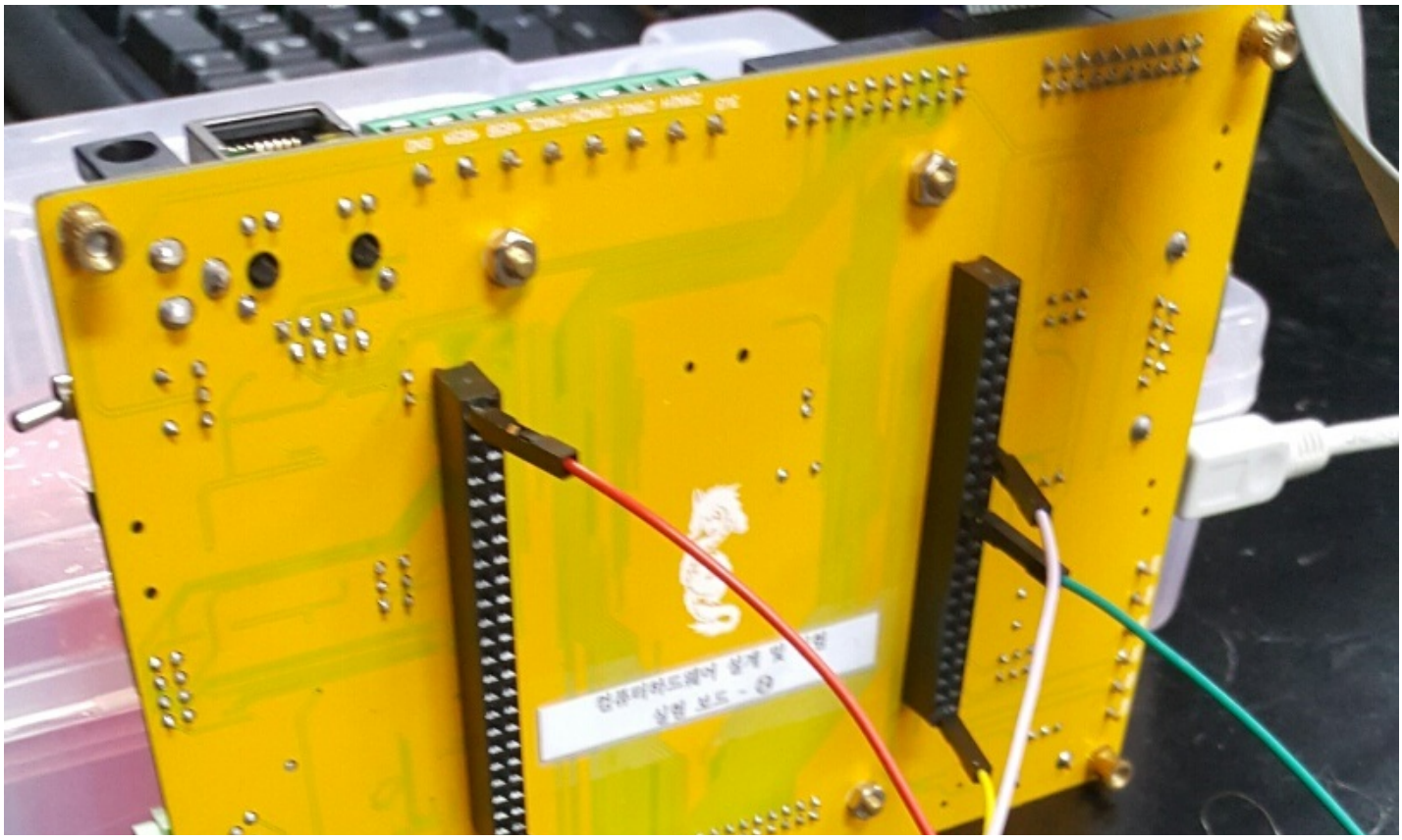
그림과 같이 request mapping 이 되어 있다. peripheral 의 DMA request 는 peripheral 의 register 에서 DMA control bit 를 programming 함으로써 독립적으로 활성화 또는 비활성화 될 수 있다.

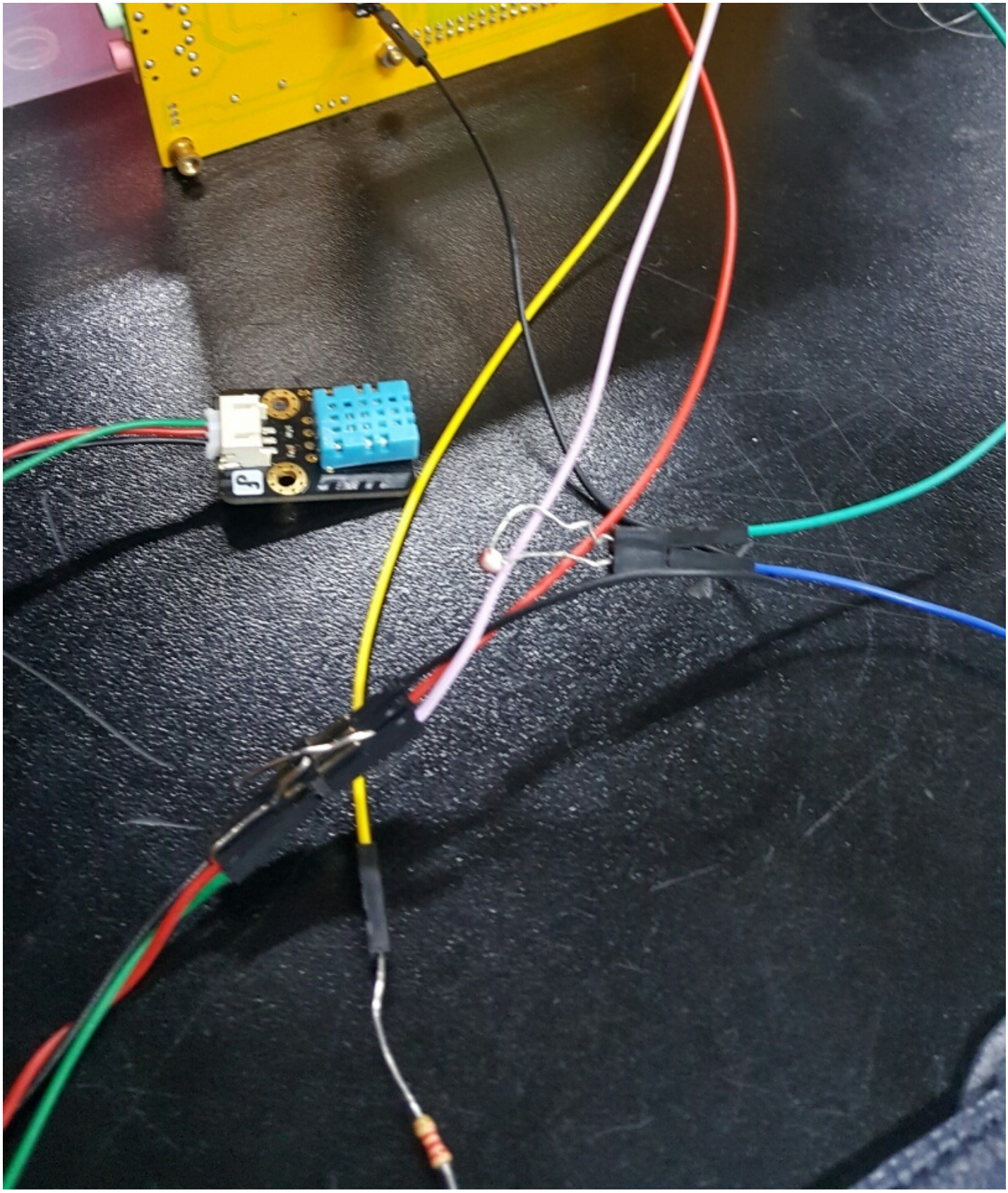
DMA channel 은 very high, high, medium, low 이렇게 4 가지의 우선순위를 가진다.

DMA1 의 1~7번 channel 순서대로 우선순위를 가진다. 우선순위에 따른 한번에 하나의 request 만 가능하다.

원래 설정되어있는 우선순위를 disable 시키고 우선순위를 직접적으로 설정할 수 있다.

실험 방법





다음과 같이 jump 선을 이용하여 board 에 조도센서와 온습도센서를 연결한다.

ADC 와 DMA 를 사용하여 조도센서 1개와 온습도센서 1개의 값을 받아오는 코드를 작성. 이때, ADC 는 interrupt 를 사용하지 않는다.

```
void set_ADC(void) {
        ADC_InitTypeDef ADC_InitStructure;
        ADC_DeInit(ADC1);
        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
        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 = 2;
        ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1,
        ADC_SampleTime_55Cycles5);
        ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 2,
        ADC_SampleTime_55Cycles5);
        ADC_Init(ADC1, &ADC_InitStructure);

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

void DMA_init() {
        DMA_InitTypeDef DMA_InitStructure;
        DMA_DeInit(DMA1_Channel1);
        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 = 2;
        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);
        DMA_Cmd(DMA1_Channel1, ENABLE);
}
```

DMA*InitTypeDef 구조체를 선언하고 초기화 후 Enable 시킨다. ADC1에 Channel 11과 12를 열어주면 DMA*MemoryBaseAddr의 값으로 준 ADC_Value에 조도센서와 온습도센서의 data가 각각 update 된다.

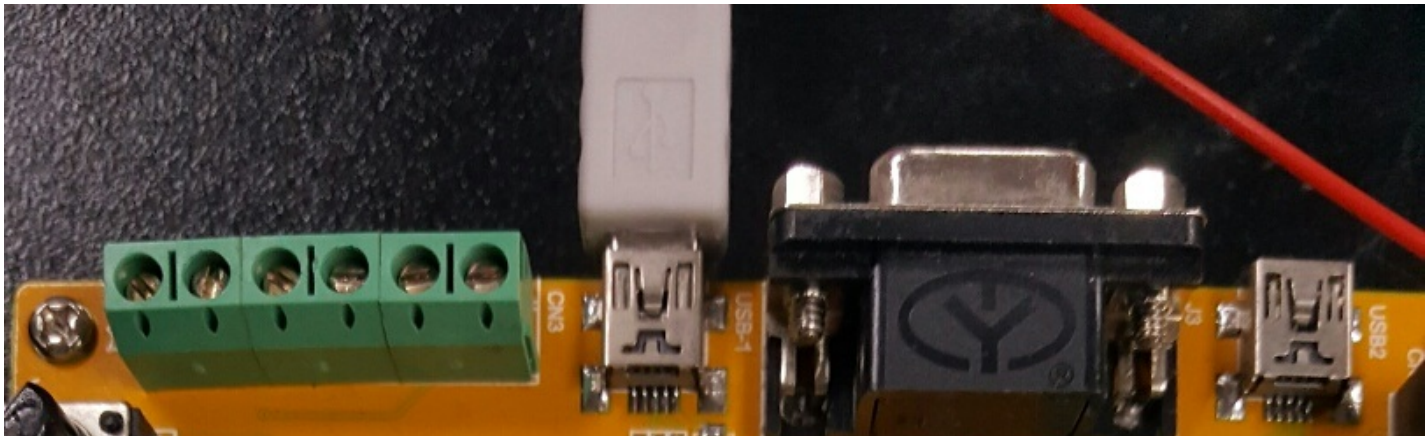
```
void TIM2_IRQHandler(void) {
        t++;
        sprintf(jodo, "%d", ADC_Value[0]);
        sprintf(onsub, "%d", ADC_Value[1]);
        j = (int)ADC_Value[0];
        if (t % 2 == 0) {
                LCD_ShowString(100, 100, jodo, BLACK, WHITE);
                LCD_ShowString(100, 150, onsub, BLACK, WHITE);
        }

        if (j > 3800) {
                GPIO_SetBits(GPIOD, GPIO_Pin_2);
        } else {
                GPIO_ResetBits(GPIOD, GPIO_Pin_2);
        }

        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        //Clears the TIMx's interrupt pending bits.
}
```

Timer 를 사용하여 조도센서 값들을 LCD 에 1초마다 출력하도록 한다.

0.5초마다 센서값을 확인하여 적당한 기준보다 어둡다고 판단되면 LED 를 점등시킨다.

Timer interrupt에서 조도값과 온습도값을 업데이트 및 LED 점등 동작을 수행한다.

실험 결과



3.2 TFT Color LCD Module Rev 2.0
Wed_team07
2886
4088

가운데 부분의 위의 값은 조도센서로부터 받아온 값이고 아래 값은 온습도센서로부터 받아온 값이다.

우리조는 조도값이 3800이상이 되면 LED 점등의 조건을 걸어서 사진에는 LED가 켜지지 않은 상태이다.

결론

DMA를 사용하여 외부 모듈과 STM32보드간의 data 전송을 할 수 있게 되었다.

텀 프로젝트를 진행하게 되면 다양한 센서를 사용하게 될텐데 DMA와 interrupt를 혼합하여 효율적으로 data 전송을 처리할 수 있을 것 같다.

전체 코드

```
#include <misc.h>
#include <stm32f10x.h>
#include <stm32f10x_exti.h>
#include <stm32f10x_gpio.h>
#include <stm32f10x_rcc.h>
#include <stm32f10x_usart.h>
#include <stm32f10x_adc.h>
#include <lcd.h>
#include <Touch.h>
#include <stdio.h>
#include <stdlib.h>

vu32 ADC_Value[2];
char onsub[10];
char jodo[10];
int j;
int t = 1;

void delay(int i) {
        int j;
        for (j = 0; j <= i * 100000; j++)
                ;
}
void TIM2_IRQHandler(void) {
        t++;
        sprintf(jodo, "%d", ADC_Value[0]);
        sprintf(onsub, "%d", ADC_Value[1]);
        j = (int)ADC_Value[0];
        if (t % 2 == 0) {
                LCD_ShowString(100, 100, jodo, BLACK, WHITE);
                LCD_ShowString(100, 150, onsub, BLACK, WHITE);
        }

        if (j > 3800) {
                GPIO_SetBits(GPIOD, GPIO_Pin_2);
        } else {
                GPIO_ResetBits(GPIOD, GPIO_Pin_2);
        }

        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        //Clears the TIMx's interrupt pending bits.
}
void SysInit(void) {
        /* Reset the RCC clock configuration to the default reset state(for de
bug purpose) */
        /* Set HSION bit */
        RCC->CR |= (uint32_t) 0x00000001;

        /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
        RCC->CFGR &= (uint32_t) 0xF0FF0000;

        /* Reset HSEON, CSSON and PLLON bits */
        RCC->CR &= (uint32_t) 0xFEF6FFFF;

        /* Reset HSEBYP bit */
        RCC->CR &= (uint32_t) 0xFFFBFFFF;

        /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
        RCC->CFGR &= (uint32_t) 0xFF80FFFF;
```

```
60
61          /* Reset PLL2ON and PLL3ON bits */
62          RCC->CR &= (uint32_t) 0xEBFFFFFF;
63
64          /* Disable all interrupts and clear pending bits  */
65          RCC->CIR = 0x00FF0000;
66
67          /* Reset CFGR2 register */
68          RCC->CFGR2 = 0x00000000;
69  }
70
71  void SetSysClock(void) {
72          volatile uint32_t StartUpCounter = 0, HSEStatus = 0;
73
74          /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------
75  ----*/
76          /* Enable HSE */
77          RCC->CR |= ((uint32_t) RCC_CR_HSEON);
78
79          /* Wait till HSE is ready and if Time out is reached exit */
80          do {
81                  HSEStatus = RCC->CR & RCC_CR_HSERDY;
82                  StartUpCounter++;
83          } while ((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));
84
85          if ((RCC->CR & RCC_CR_HSERDY) != RESET) {
86                  HSEStatus = (uint32_t) 0x01;
87          } else {
88                  HSEStatus = (uint32_t) 0x00;
89          }
90
91          if (HSEStatus == (uint32_t) 0x01) {
92                  /* Enable Prefetch Buffer */
93                  FLASH->ACR |= FLASH_ACR_PRFTBE;
94
95                  /* Flash 0 wait state */
96                  FLASH->ACR &= (uint32_t) ((uint32_t) ~FLASH_ACR_LATENCY);
97                  FLASH->ACR |= (uint32_t) FLASH_ACR_LATENCY_0;
98
99                  /* HCLK = SYSCLK = 48MHz */
100                 RCC->CFGR |= (uint32_t) RCC_CFGR_HPRE_DIV1;
101
102                 /* PCLK2 = HCLK = 48MHz */
103                 RCC->CFGR |= (uint32_t) RCC_CFGR_PPRE2_DIV1;
104
105                 /* PCLK1 = HCLK  = 24MHz  */
106                 RCC->CFGR |= (uint32_t) RCC_CFGR_PPRE1_DIV1;
107
108                 /* Configure PLLs ------------------------------------------
109 ----------*/
110                 /* PLL configuration: PLLCLK = PREDIV1 * 6 = 48MHz */
111                 RCC->CFGR &= (uint32_t) ~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC
112                                 | RCC_CFGR_PLLMULL);
113                 RCC->CFGR |= (uint32_t) (RCC_CFGR_PLLXTPRE_PREDIV1
114                                 | RCC_CFGR_PLLSRC_PREDIV1 |
115                                 RCC_CFGR_PLLMULL6);
116
117                 /* PLL2 configuration: PLL2CLK = HSE/5 * 8 = 40MHz  */
118                 /* PREDIV1 configuration: PREDIV1CLK = PLL2 / 5 = 8MHz */
119                 RCC->CFGR2 &= (uint32_t) ~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2M
120 UL |
121                 RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
122                 RCC->CFGR2 |= (uint32_t) (RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_P
123 LL2MUL8 |
124                 RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV5);
125
126                 /* Enable PLL2 */
127                 RCC->CR |= RCC_CR_PLL2ON;
128                 /* Wait till PLL2 is ready */
129                 while ((RCC->CR & RCC_CR_PLL2RDY) == 0) {
130                 }
131
132                 /* Enable PLL */
133                 RCC->CR |= RCC_CR_PLLON;
134
135                 /* Wait till PLL is ready */
136                 while ((RCC->CR & RCC_CR_PLLRDY) == 0) {
137                 }
138
```

```
                /* Select PLL as system clock source */
                RCC->CFGR &= (uint32_t) ((uint32_t) ~(RCC_CFGR_SW));
                RCC->CFGR |= (uint32_t) RCC_CFGR_SW_PLL;

                /* Wait till PLL is used as system clock source */
                while ((RCC->CFGR & (uint32_t) RCC_CFGR_SWS) != (uint32_t) 0x0
8) {
                }
        } else { /* If HSE fails to start-up, the application will have wrong
clock
         configuration. User can add here some code to deal with this error */
        }
}
void init_Timer2() {
        NVIC_InitTypeDef NVIC_InitStructure; // for interreupt
        TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; // timerbase...

        /* TIM2 Clock Enable */
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

        /* Enable TIM2 Global Interrupt */
        NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);

        /* TIM2 Initialize */
        TIM_TimeBaseStructure.TIM_Period = 600;
        TIM_TimeBaseStructure.TIM_Prescaler = 60000;
        //계산방법 : 1/72mhz * 1200 * 60000
        TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
        TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
        TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

        /* TIM2 Enale */
        TIM_ARRPreloadConfig(TIM2, ENABLE);
        TIM_Cmd(TIM2, ENABLE);
        TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); // interrupt enable
}

void set_ENABLE(void) {
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);         // interr
upt
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);     // RCC GPIO
E
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);     // RCC GPIO
C
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);         // ADC1
        RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);                  //
DMA1
}
void set_LED(){
        GPIO_InitTypeDef LED;
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);     // RCC GPIO
D
        LED.GPIO_Mode = GPIO_Mode_Out_PP;
        LED.GPIO_Pin = GPIO_Pin_2;
        LED.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOD, &LED);
}

void set_PC1(void) {
        GPIO_InitTypeDef GPIO_InitStructure;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
}
void set_PC2(void) {
        GPIO_InitTypeDef GPIO_InitStructure;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
}

void set_ADC(void) {
        ADC_InitTypeDef ADC_InitStructure;
```

```
        ADC_DeInit(ADC1);
        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
        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 = 2;
        ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1,
        ADC_SampleTime_55Cycles5);
        ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 2,
        ADC_SampleTime_55Cycles5);
        ADC_Init(ADC1, &ADC_InitStructure);

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

}

void ADC_start(void) {
        ADC_ResetCalibration(ADC1);
        while (ADC_GetResetCalibrationStatus(ADC1))
                ;
        ADC_StartCalibration(ADC1);
        while (ADC_GetCalibrationStatus(ADC1))
                ;
        ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

void DMA_init() {
        DMA_InitTypeDef DMA_InitStructure;
        DMA_DeInit(DMA1_Channel1);
        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 = 2;
        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);
        DMA_Cmd(DMA1_Channel1, ENABLE);
}

int main() {
        SystemInit();
        set_ENABLE();
        set_PC1();
        set_PC2();
        set_ADC();
        set_LED();
        LCD_Init();
        LCD_Clear(WHITE);
        DMA_init();
        init_Timer2();
        ADC_start();
//        GPIOD->CRL = (GPIO_CRL_MODE2_0 | GPIO_CRL_MODE3_0 | GPIO_CRL_MODE4_0
//                        | GPIO_CRL_MODE7_0);

        while (1) {
                LCD_ShowString(1, 1, "Wed_team07", BLACK, WHITE);
        }

}
```