임베디드 시스템 설계 및 실험

6주차

7조
201424470 서민영
201424421 김시은
201424533 정종진

목차

1. 실험목표

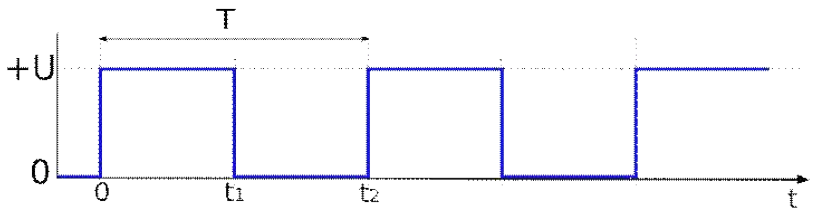
1) Clock Tree의 이해 및 사용자 Clock 설정
2) UART 통신의 원리를 배우고, 실제 설정 방법 파악

2. 배경지식

1) Clock(클럭)

① Clock이란

- 논리상태 1과 0이 주기적으로 나타나는 방형파 신호 – 디지털 회로에서 신호의 처리를 하는 기본 단위



<그림 1 : Clock>

② HSI, HSE

◦ STM32: STMicroelectronics 사의 32비트 마이크로컨트롤러 직접회로

◦ STM32에 클럭을 공급하는 방법

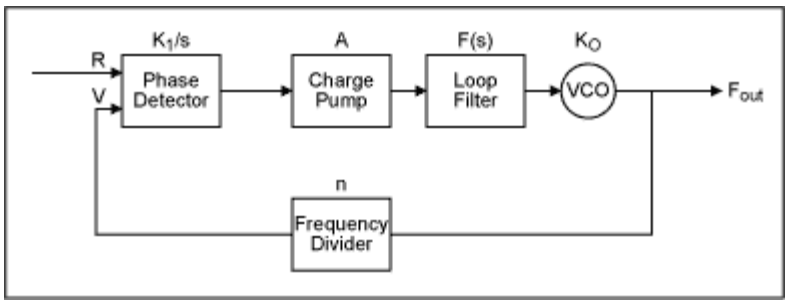
- HIS(High Speed Internal) Clock – 내장되어 있는 RC발진회로의 주파수를 클럭으로 사용
- HSE(High Speed External) Clock – 크리스탈이나 오실레이터 등 외부에 부착한 것에서 입력되는 주파수를 클럭으로 사용

③ PLL(Phase Locked Loop)

◦ 입력신호와 출력신호의 위상차를 검출하여 주파수의 흔들림을 막기위해 VCO를 제어해서 고정된 주파수 신호를 발신하는 회로

◦ VCO(Voltage Controlled Osillator): 전압제어 발진기

- 입력신호의 주파수를 조절해서 출력 신호를 내보낼 수 있다.



<그림 2 : PLL>

④ MCO(Microcontroller Clock Output)

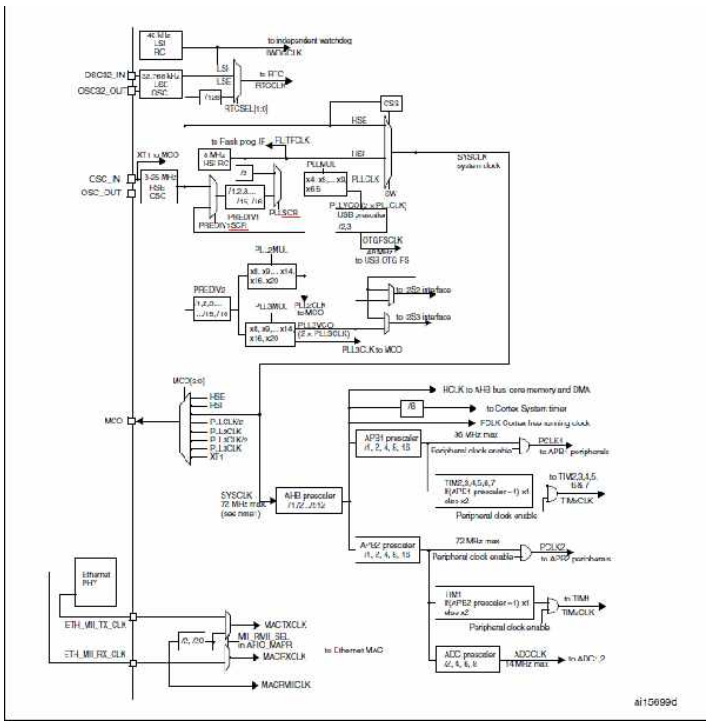
◦ 내부에서 사용되는 클럭을 외부로 출력할 수 있는 핀

◦ HSI/HSE, SYSCLK

◦ MCO핀을 오실로스코프로 연결하면 출력되는 파형을 확인할 수 있다.

⑤ Clock tree

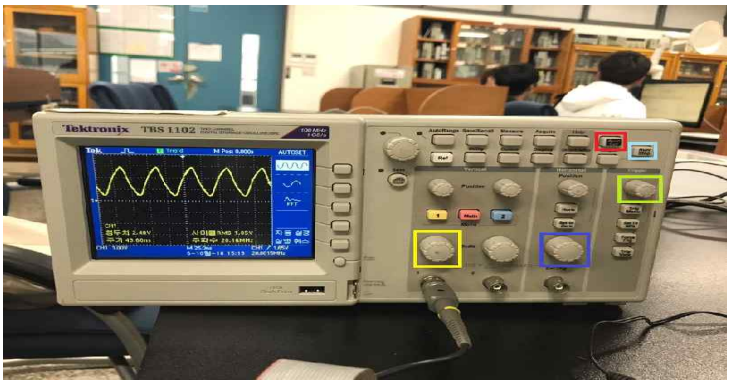
◦ HSE 클럭을 선택
◦ PLL로 주파수 조절(단, GPIO 최대속도를 넘으면 안됨)
◦ System clock이 MCO로 들어감



<그림 3 : Clock tree>
※오타: 빨간줄 SCR→SRC

2) 오실로스코프

◦ 오실로스코프 사용법



<그림 4 : 오실로스코프 사진>

(1) 전원을 켠다.
(2) 프로브를 연결하여 보드의 오실로스코프와 GND 핀에 각각 꽂는다.
(3) 노란색 노브를 조절하여 전압단위를 조절한다.
(4) 파란색 노브를 조절하여 시간단위를 조절한다.
(5) 초록색 노브를 조절하여 트리거지점을 설정한다.
(6) 하늘색 Run/Stop 버튼을 누르면 파형을 정지시켜서 볼 수 있다.

(7) 빨간색 AutoSet 버튼을 누르면 최적화된 파형 그림이 출력된다.

3) UART/USART

① 통신의 종류

◦직렬 통신: 한 번에 하나의 비트 단위로 데이터를 전송

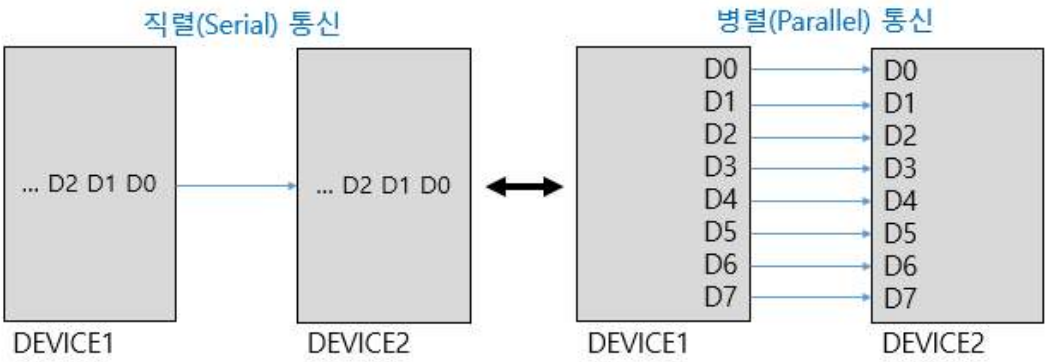
◦병렬 통신: 동시에 여러 개의 비트를 전송

◦비동기 통신: 송/수신 측의 자체 생성 클록에 동기화 된 데이터 송수신

- 각 측의 통신 조건이 초기에 일치해야 정상통신 가능

◦동기 통신: 다른 장비 또는 자체 생성 클럭에 동기된 데이터 송수신

- 데이터 전송 효율이 좋지만 절차가 복잡함



<그림 5 : 직렬통신과 병렬통신>

② UART vs USART

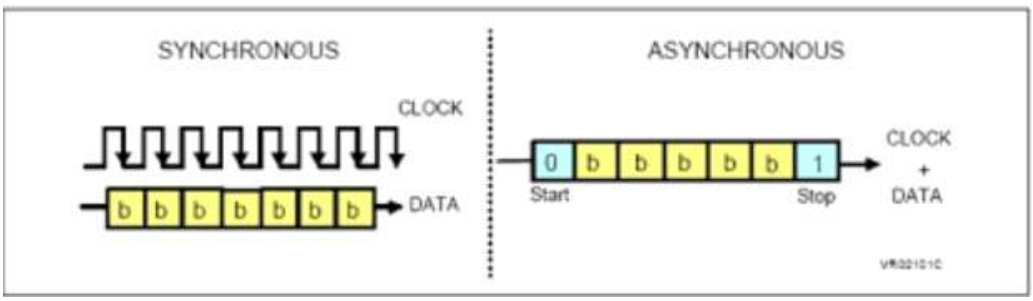
◦UART(Universal asynchronous receiver/transmitter)

◦USART(Universal Synchronous and Asynchronous serial Receiver and Transmitter)

◦UART와 USART의 차이점 - 클럭을 같이 보내느냐 아니냐의 차이

◦USART는 UART이면서 경우에 따라 클럭에 맞추어 데이터를 보내거나 받을 수 있음

◦즉, 동기화된 통신을 지원



<그림 6 : Synchronous와 Asynchronous>

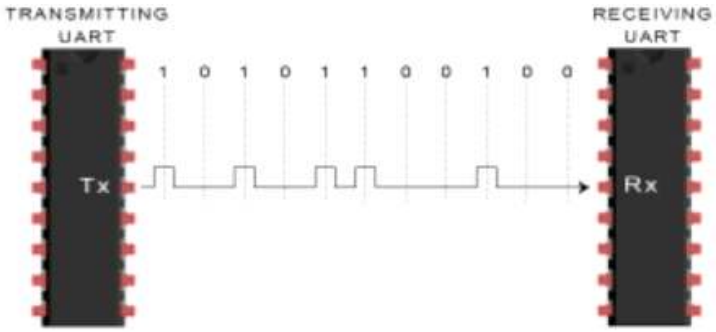
③ UART

◦병렬 데이터의 형태를 직렬 방식으로 전환하여 데이터를 전송하는 컴퓨터 하드웨어의 일종

◦주로 디바이스와 디바이스간에 1:1 통신을 위하여 사용

◦송신라인과 수신라인이 따로 있어 동시에 송수신이 가능

◦수신 쪽에서 동기신호를 찾아내어 데이터의 시작과 끝을 시간적으로 알아 처리할 수 있도록 약속



<그림 7 : UART통신 그림>

④ 전송 절차

◦ 송신 UART가 DATA bus에서 병렬형태로 데이터를 받음

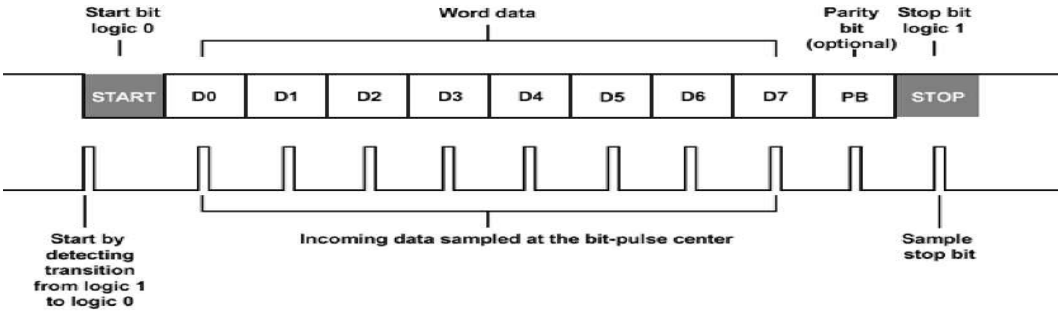
◦ 송신 UART가 start bit, parity bit, stop bit를 data frame에 추가

◦ 송신 UART에서 수신 UART로 전체 packet을 직렬로 보내고, 수신 UART는 사전 협의된 전송 속도로 데이터 라인을 샘플링

◦ 수신 UART에서 start bit, parity bit, stop bit을 data frame에서 제거

◦ 수신 UART는 직렬 데이터를 다시 병렬로 변환하여 수신 측의 데이터 버스로 전송

⑤ DATA FRAME



<그림 8 : DATA FRAME 그림>

◦ Start Bit: UART 데이터 전송 라인은 일반적으로 비 전송 시 고전압 레벨 유지

- 수신 UART가 고전압에서 저전압으로의 변화 감지 시, baud rate의 주파수에서 데이터 프레임의 비트를 읽기 시작

◦ Data Frame: 데이터 프레임은 전송중인 실제 데이터를 포함

- 패리티 비트를 사용 시 5 비트에서 최대 8 비트 길이,
- 비사용 시 9 비트 길이 , 데이터는 최하위 비트부터 먼저 전송

◦ Parity bit: UART가 전송 중 데이터가 변경되었는지 여부를 알 수 있는 방법

- 수신 쪽 데이터 프레임을 읽은 후, 1의 값을 가진 비트 수가 짝수, 홀수인지 패리티와 비교하여 에러를 확인
- 사용 안함, 짝수, 홀수 패리티 등의 세가지 옵션으로 해당 레지스터 설정에 따라 선택

◦ stop bit: 데이터 패킷의 끝을 신호하기 위해 송신 UART는 적어도 2 비트 기간 동안

- 저전압에서 고전압으로 데이터 전송 라인을 구동

◦ baud rate: 1초당 데이터가 변조되는 비율

- 비동기식 통신을 할 경우 송신부와 수신부는 같은 속도로 데이터를 보내거나 받아야 함
- 정확한 데이터 송수신을 위해서 통신 속도를 정해줘야 하는데, 이것이 baud rate 세팅

⑥ RS-232

- UART라는 통신 방식으로 데이터를 주고 받을 수 있게 정한 통신장비규격
- 1969년 미국의 EIA(Electric Industries Association)에 의해 정해진 표준 인터페이스 PC 통신을 위한 하드웨어 규격

3. 실험과정

1) 보드 연결

표 1 : 보드 연결 순서

<그림 1 : Coretex M3/JTAG/DSTREAM을 연결한 모습>

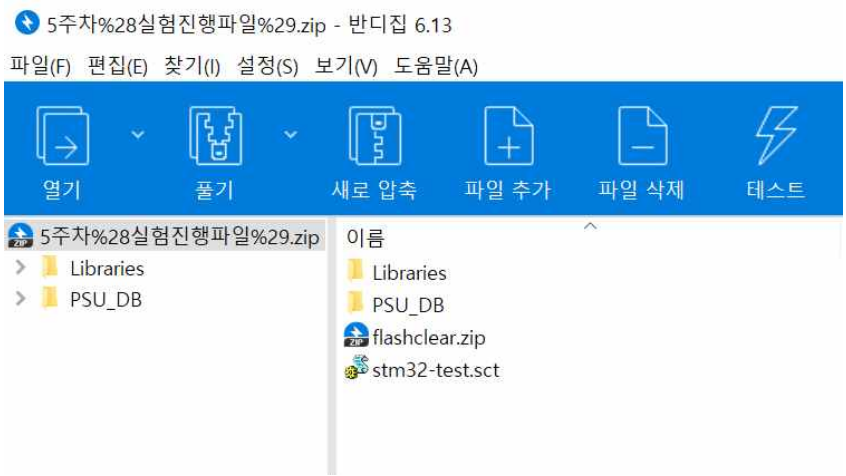
1-1) Coretex M3/JTAG/DSTREAM 연결

다음과 같은 순서로 보드를 연결한다. 이 때 연결 및 분리 순서를 제대로 지키지 않으면 장비가 망가질 수 있으니 주의해야한다.

① 보드와 DSTREAM JTAG 연결
② 보드 전원선만 연결
(보드의 전원은 OFF 상태)
③ DSTREAM 전원 연결 및 ON
④ DSTREAM Status LED 점등 확인
⑤ 보드 전원 ON
⑥ DSTREAM Target LED 점등 확인
⑦ DS-5에서 ‘connect target’

2) DS-5 디바이스 데이터베이스 추가

2-1) 수업게시판에서 실습파일로 제공되는 PSU_DB, LIBRARIES, SCATTER FILE을 다운



<그림 7 : 실습파일로 제공된 파일들>

2-3) Eclipse에 데이터베이스 추가

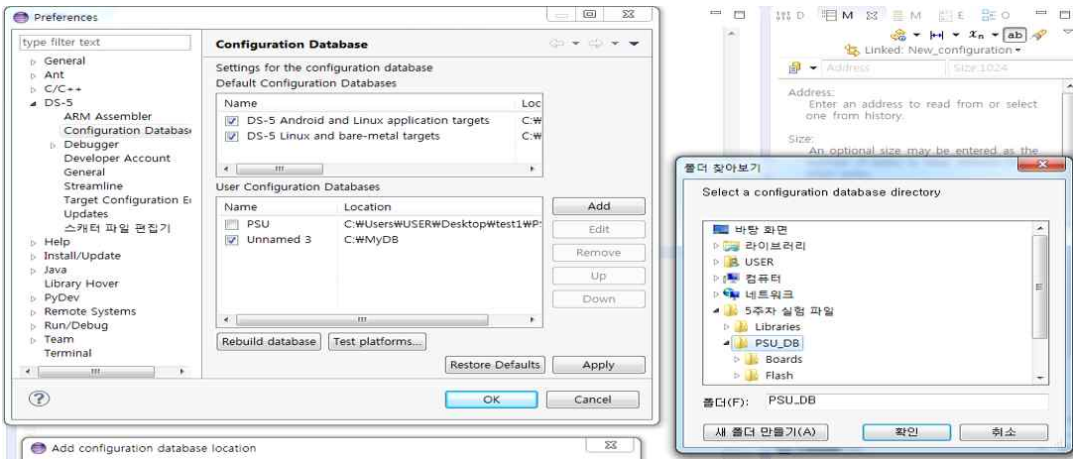
① 시작 → 모든 프로그램 → ARM DS-5 → Eclipse for DS-5 메뉴를 선택

② Windows → Preferences

③ DS-5 → Configuration Database 항목을 선택

④ Add 버튼을 클릭하여 사용자 데이터베이스의 디렉터리를 지정

⑤ Rebuild database 버튼을 클릭하여 데이터베이스 추가를 완료



<그림 8 : Eclipse에 데이터베이스 추가>

3) C Project 생성 및 환경설정

3-1) C Project 생성

① New Project → C project → Excutable → Empty Project를 선택해주고 Toolchains는 ARM Cmpiler 5(DS-5 built in)로 선택 후 프로젝트 생성

3-2) C Project Properties 설정

① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → ARM Linker 5 → Image Layout → Scatter file 설정

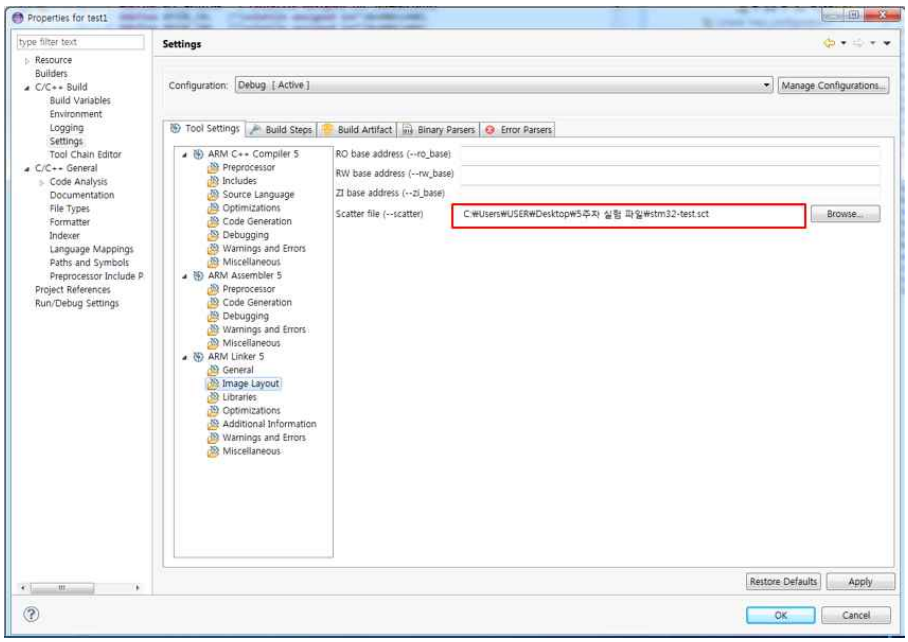
(Scatter file에서 RO/RW base address를 지정해주므로 따로 설정해줄 필요가 없음)

② C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Code Generation과 General의 Target CPU를 Cortex M3로 설정

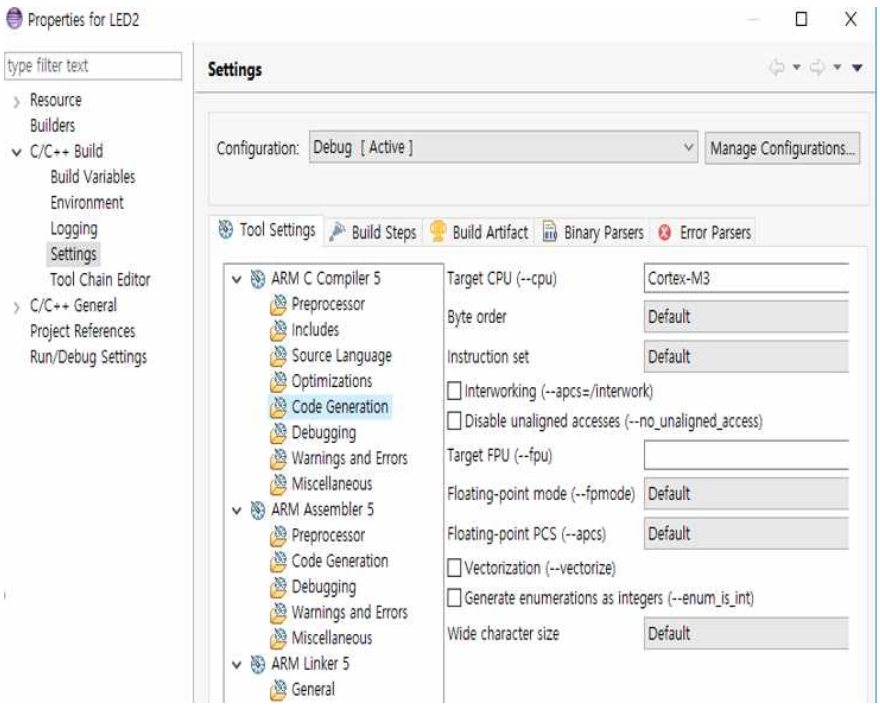
(entry point를 main으로 지정해주지 않아도 됨)

3-3) LIBRARIES 추가

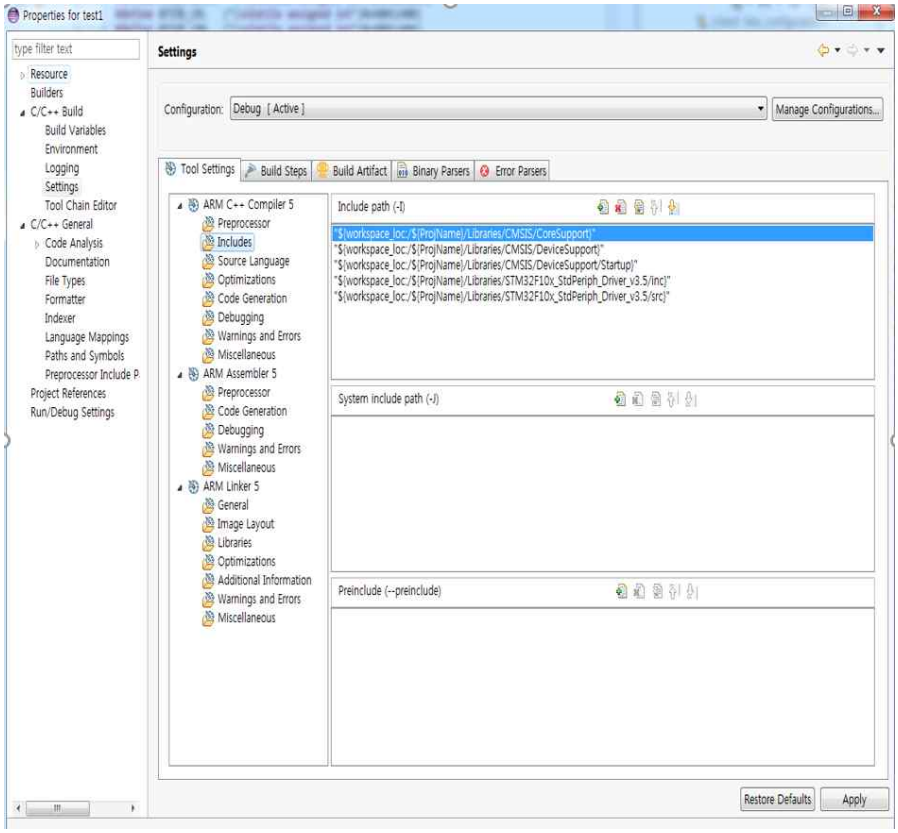
① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Includes에서 제공되는 LIBRARIES 추가



<그림 9 : Scatter file 설정>



<그림 10 : Target CPU 설정>



<그림 11 : 제공되는 LIBRARIES 추가>

4) DS-5 Debugger 연결

4-1) Debug Configuration 설정

① Run → Debug Configuration 메뉴를 선택

② DS-5 Debugger 더블 클릭하여 새로운 하위 오브젝트 생성

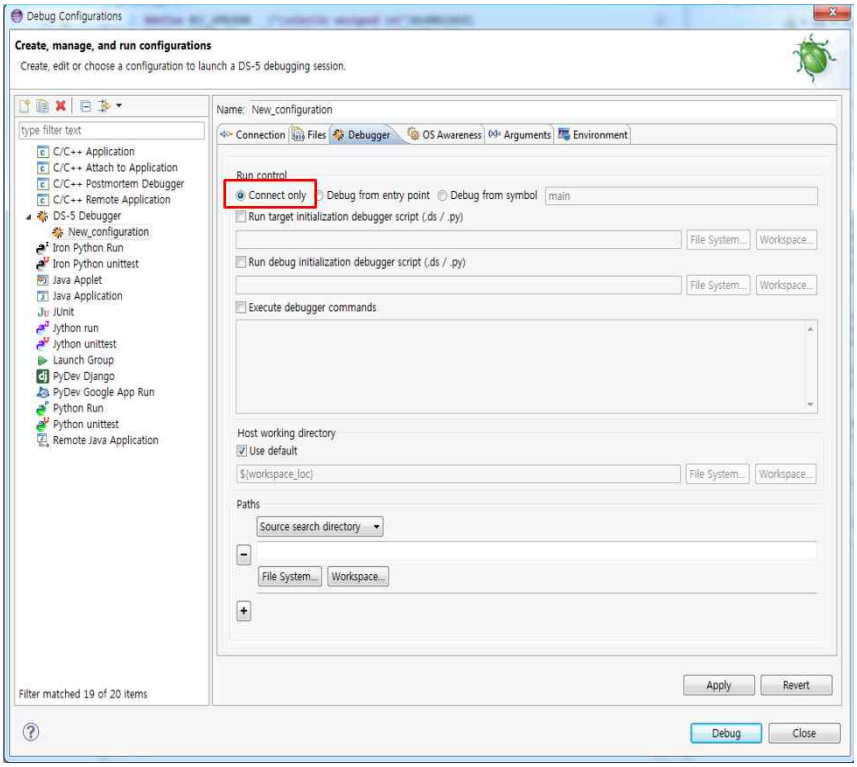
③ Name, Platform 등 Debug 환경설정을 변경

④ Browse 버튼을 클릭하여 DSTREAM 장비를 detection

4 2) Debug

① Debugger 탭에서 Connect only 체크

② Apply 클릭



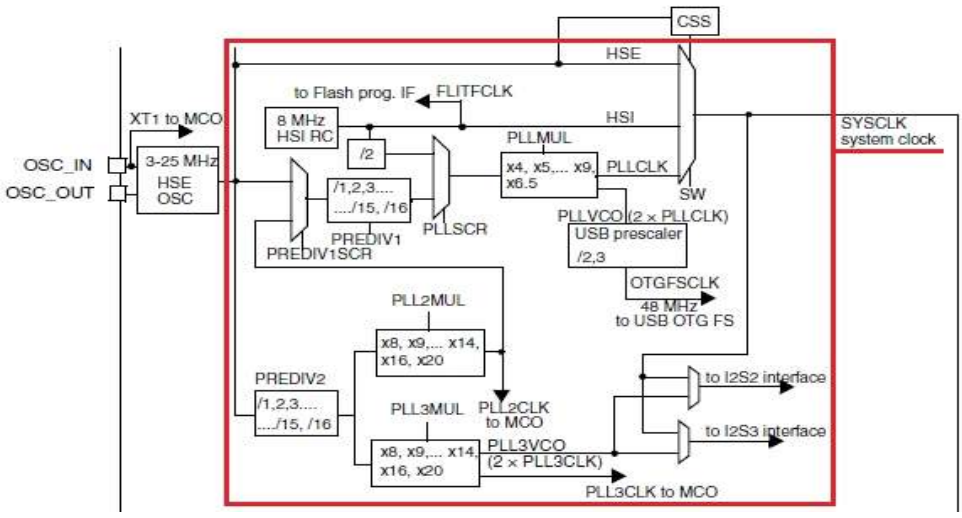
<그림 12 : Debug Configurations에서 Debugger탭의 설정화면>

5) C source code 작성

5-1) 이번 실험에서 주어진 Clock, Baud Rate 값

System Clock : 40MHz, HCLK : 20MHz, PCLK2 : 10MHz, Baud Rate : 115200

5-2) HSE-25MHz 값을 이용하여 System Clock 계산



<그림 13 : Clock 회로도>

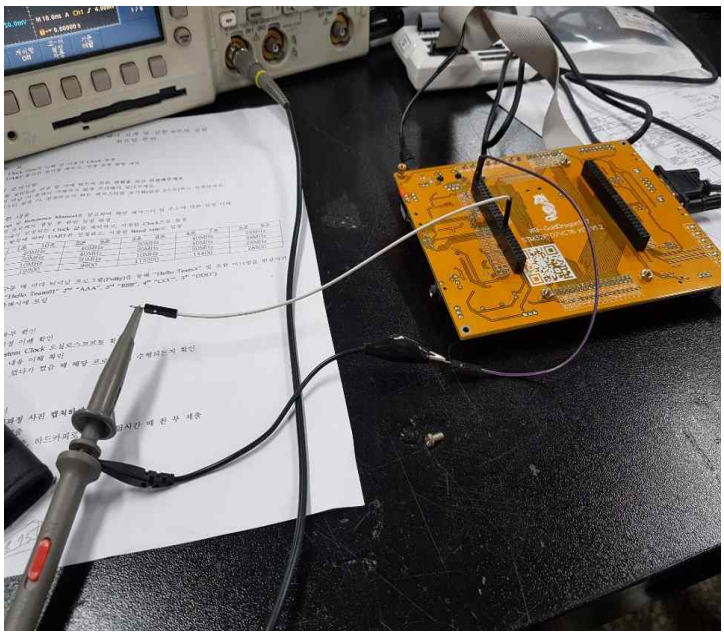
① PREDIV2 에서 /5

② PLL2MUL 에서 *8

③ PREDIV1 에서 /8

④ PLLMUL 에서 *8

6) 오실로스코프 연결



<그림 14 : Cortex-M3와 오실로스코프 연결, UART Cable 과 PC 연결>

7) C project 빌드 및 .axf 업로드

6-1) C project 빌드 및 Debug As

① C Project 빌드

(.axf 파일이 생성된 것을 확인할 수 있음)

② C Project 우클릭 → Debug → Debug As

6-2) flashclear.axf 및 생성된 .axf 업로드

① command 라인에 다음과 같은 명령어를 입력해서 flashclear.axf를 업로드

flash load “flashclear.axf 파일경로”

② disconnect한 다음 보드를 껐다가 켬

③ command 라인에 다음과 같은 명령어를 입력해서 생성된 .axf를 업로드

flash load “생성된 .axf 파일경로”

④ disconnect한 다음 보드를 껐다가 켬

(flash load 후에는 반드시 diconnect를 하고 보드를 껐다가 켜야 함)

6-3) 오실로스코프에 나타나는 파형 확인

① 소스코드에서 작성한 주파수와 일치하는 지 확인한다.

② 제대로 동작하지 않으면 5) C source code 작성으로 돌아감

7) 보드 연결 해체

앞서 보드 연결과 마찬가지로, 보드 연결 해체 시에도 순서를 제대로 지키지 않으면 보드가 망가질 수 있으므로 유의해야한다. 보드 연결 해체 순서는 다음과 같다.

표 2 : 보드 연결 해체 순서

① DS-5에서 ‘disconnect target’ ② 보드 전원 OFF ③ DSTREAM 전원 해제 및 OFF ④ 보드 전원선 분리 ⑤ DSTREAM과 보드 JTAG 분리

4. 작성한 소스코드 및 실험결과

1) 작성한 소스코드

전체 소스코드

```
//flash load "C:\Users\Team07\Desktop\wk6\Team07\Debug\Team07.axf"
//"C:\Users\Team07\Desktop\wk6\Team07\flashclear\flashclear.axf"
#include "stm32f10x.h"

const int SELECT = 0x100;

void SysInit(void) {
   /* Set HSION bit */
   /* Internal Clock Enable */
   RCC->CR |= (uint32_t)0x00000001; //HSION

   /* 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;

   /* Reset PLL2ON and PLL3ON bits */
   RCC->CR &= (uint32_t)0xEBFFFFFF;

   /* Disable all interrupts and clear pending bits  */
   RCC->CIR = 0x00FF0000;

   /* Reset CFGR2 register */
   RCC->CFGR2 = 0x00000000;
}

void SetSysClock(void)
{
   volatile uint32_t StartUpCounter = 0, HSEStatus = 0;

   /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration */
   /* Enable HSE */
   RCC->CR |= ((uint32_t)RCC_CR_HSEON);

   /* Wait till HSE is ready and if Time out is reached exit */
   do
   {
      HSEStatus = RCC->CR & RCC_CR_HSERDY;
      StartUpCounter++;
   } while ((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));
```

```
  if ((RCC->CR & RCC_CR_HSERDY) != RESET)
  {
    HSEStatus = (uint32_t)0x01;
  }
  else
  {
    HSEStatus = (uint32_t)0x00;
  }

  if (HSEStatus == (uint32_t)0x01)
  {
    /* Enable Prefetch Buffer */
    FLASH->ACR |= FLASH_ACR_PRFTBE;

    /* Flash 0 wait state */
    FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
    FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_0;

    /* HCLK = SYSCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;

    /* PCLK2 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;

    /* PCLK1 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;

    /* Configure PLLs  */
    /* PLL configuration: PLLCLK = ???? */
    //
    RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC |
RCC_CFGR_PLLMULL);
    RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1
|RCC_CFGR_PLLMULL8);

    /* PLL2 configuration: PLL2CLK = ???? */
    /* PREDIV1 configuration: PREDIV1CLK = ???? */
    RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL |
      RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
    RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL8
|RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV8);

    /* Enable PLL2 */
    RCC->CR |= RCC_CR_PLL2ON;
    /* Wait till PLL2 is ready */
    while ((RCC->CR & RCC_CR_PLL2RDY) == 0)
    {
    }

    /* Enable PLL */
    RCC->CR |= RCC_CR_PLLON;

    /* Wait till PLL is ready */
    while ((RCC->CR & RCC_CR_PLLRDY) == 0)
    {
    }

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

```
//      RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_2;

      /* Wait till PLL is used as system clock source */
      while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08)
      {
      }
      /* Select System Clock as output of MCO */
      RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
      RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK;
   }
   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 UartInit(void) {
   /* RCC Configuration */
   /* GPIO RCC Enable  */
   //@TODO
   RCC->APB2ENR |= (RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPBEN);
   /* USART RCC Enable */
   //@TODO
   RCC->APB2ENR |= RCC_APB2ENR_USART1EN;

   /* USART Pin Configuration */
   //@TODO
   GPIOA->CRH = 0x00000000;
   GPIOA->CRH   |=   (GPIO_CRH_MODE8   |   GPIO_CRH_CNF8_1   |   GPIO_CRH_MODE9   |
GPIO_CRH_CNF9_1); // PA9 : USART1 TX // PA8 : MCO output

   /* USART CR2 Configuration */
   /* Clear STOP[13:12] bits */
   //@TODO reset bits
   USART1->CR2 = 0x00000000;
   /* Configure the USART Stop Bits, Clock, CPOL, CPHA and LastBit */
   //@TODO reset bits
   USART1->CR2   &=   ~(USART_CR2_STOP   |   USART_CR2_CPOL   |   USART_CR2_CPHA   |
USART_CR2_CLKEN | USART_CR2_LBCL);
   //USART1->CR2 |= (USART_CR2_CLKEN | USART_CR2_LBCL);

   /* Set STOP[13:12] bits according to USART_StopBits value */
   //@TODO : Stop bit : 1bit
   //USART1->CR2 |= USART_CR2_STOP_0;

   /* USART CR1 Configuration */
   /* Clear M, PCE, PS, TE and RE bits */
   //@TODO reset bits
   USART1->CR1 &= ~(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR1_TE |
USART_CR1_RE);
   /* Configure the USART Word Length, Parity and mode */
   /* Set the M bits according to USART_WordLength value */
   //@TODO : WordLength : 8bit

   /* Set PCE and PS bits according to USART_Parity value */
   //@TODO : Parity : None

   /* Set TE and RE bits according to USART_Mode value */
   //@TODO enable TE, RE
   USART1->CR1 |= (USART_CR1_RE | USART_CR1_TE);
   /* USART CR3 Configuration */
   /* Clear CTSE and RTSE bits */
```

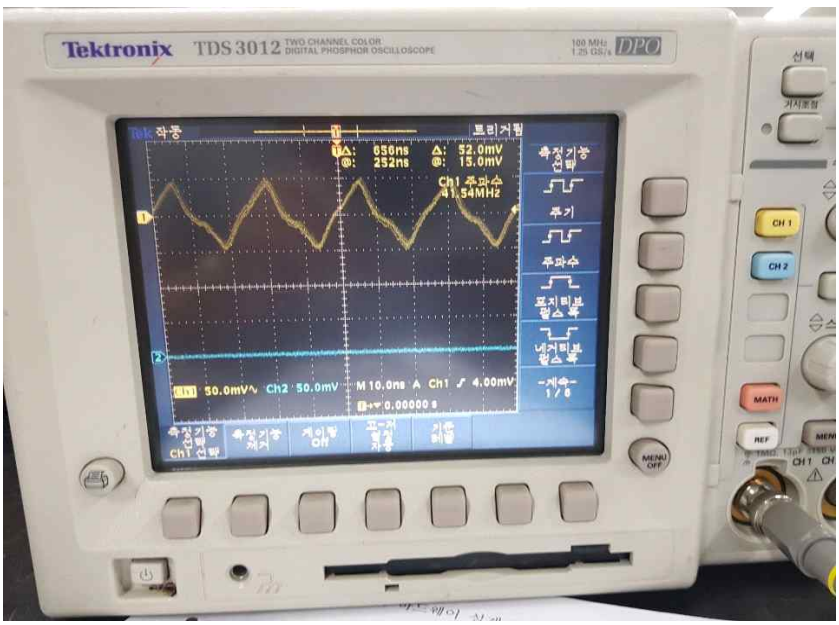
```
    //@TODO reset bits
    USART1->CR3 &= ~(USART_CR3_CTSE | USART_CR3_RTSE);
    /* Configure the USART HFC */
    /* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */
    //@TODO : CTS, RTS : disable
    /* USART BRR Configuration */
    /* Configure the USART Baud Rate */
    /* Determine the integer part */
    /* Determine the fractional part */
    //@TODO reference manual 참조
    USART1->BRR = 0x56D;
    /* USART Enable */
    /* USART Enable Configuration */
    //@TODO USART1 Enable
    USART1->CR1 |= USART_CR1_UE;
    /* USART DATA output */
    /* USART DATA Transmission */
}

void delay(void) {
    int i;
    for(i=0;i<50000;++i)
            ;
}
void delay2(void) {
    int i;
    for(i=0;i<100000;++i)
            ;
}
void SendData(int data) {
    USART1->DR = data & 0xFF;
    delay();
}
int main() {
    GPIOB->CRH = 0x00000000;
    GPIOB->CRH |= (GPIO_CRH_CNF8_1);
    GPIOA->BRR |= 0x100;
    GPIOA->BSRR |= 0x100;
    SysInit();
    SetSysClock();
    UartInit();
    int cnt = 0;
    /* USART DATA output */
    while (1) {
        if((~GPIOB->IDR)&SELECT) {
            if(cnt==0) {
                SendData('H');
                SendData('E');
                SendData('L');
                SendData('L');
                SendData('O');
                SendData(' ');
                SendData('T');
                SendData('e');
                SendData('a');
                SendData('m');
                SendData('7');
                SendData('\r');
                SendData('\n');
            }
            else if(cnt==1) {
```

```
            SendData('S');
            SendData('M');
            SendData('Y');
            SendData('\r');
            SendData('\n');
         }
         else if(cnt==2) {
            SendData('J');
            SendData('J');
            SendData('J');
            SendData('\r');
            SendData('\n');
         }
         else {
            SendData('K');
            SendData('S');
            SendData('E');
            SendData('\r');
            SendData('\n');
         }
         cnt++;
         cnt = cnt%4;
         delay2();
      }
   }
   return 0;
}
```

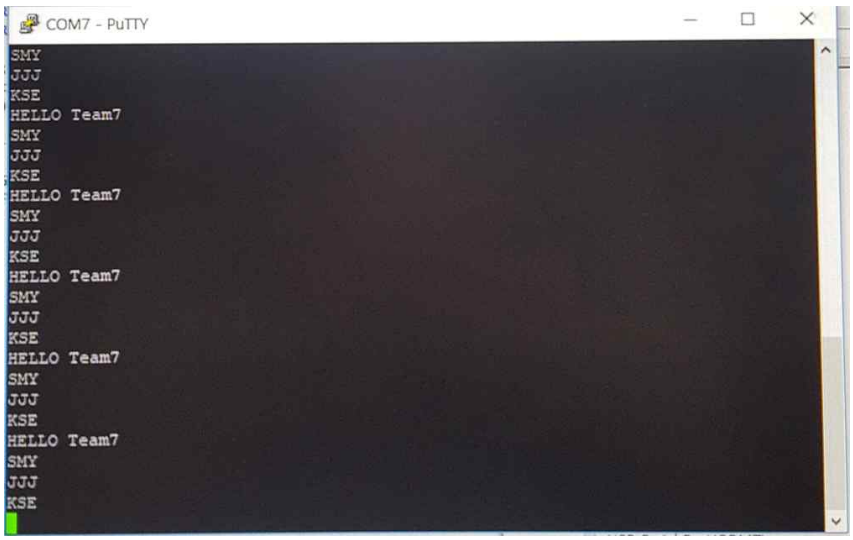
2) 실험결과

2-1) 오실로스코프에서 40Hz에 해당하는 파형 확인



<그림 14 : System Clock : 40Mhz>

2-2) 조이스틱 select를 누를 때 putty에서 출력되는 결과



<그림 9 : putty에 출력된 결과>

5. 결론 및 느낀점

이전 실험들과 이번 실험에서 달랐던 점은, header file에서 #define되어 있는 값들을 찾아서 썼다는 점이다. 즉, 이전 실험에서는 reference manual에서 16진수 형태의 주소 값과 데이터 값을 찾아 썼다면 이번에는 “stm32f10x.h”에서 해당 값의 define 명(예 : GPIO_CRH_CNF8_1)을 찾아 사용했다. header file을 활용하니 어떤 값들인지 좀 더 직관적으로 알 수 있어서 좋았고 잘못된 값을 넣은 부분을 찾고 수정하는 과정도 훨씬 더 쉬워졌다.

HSE을 사용하여 STM32에 클럭을 공급하고 PLL로 주파수 설정한 후 MCO핀을 오실로스코프로 연결해 해당 주파수를 확인하였다. 이 과정에서 클럭 공급방법, 주파수 설정방법 등에 대해 알 수 있었으며, Clock tree에서 system clock이 지나가는 경로에 대해 배울 수 있었다.

UART통신과 USART통신의 차이점에 대해서 배울 수 있었으며, 정확한 데이터 송수신을 위해서는 baud rate 세팅이 필요하다는 사실과 적절한 baud rate를 계산하는 법에 대해서도 알 수 있었다. 또한, USART 통신에서 delay를 먼저 준 후 data를 보내주는 식으로 코드를 작성하는 실수를 했었는데 data를 보내고 delay를 줘야한다는 사실도 알 수 있었다.

6. 참고 문헌 및 사이트

[1] 부산대학교 ENS 연구실 (https://enslab.pusan.ac.kr/)
[2] 2018_6주차_임베디드_시스템_설계_및_실험 예비조 발표자료