2021년 2학기

임베디드 실험

3주차 실험보고서

201712159 조현우

201724566 전승윤

201745611 최호진

201924650 박지호

목차

1. 실험목적

1. Clock Tree의 이해 및 사용자 Clock 설정

2. UART 통신의 원리를 배우고, 실제 설정 방법 파악

2. 실험과정

1. Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해

2. 예제 코드에서 설정되는 Clock 값을 파악하고, 지정된 Clock으로 설정

3. 예제 설정 항목에 따라 UART를 설정하고, 지정된 Baud rate로 설정

SYSCLK	52MHz
PCLK2	26MHz
Baud Rate	14400

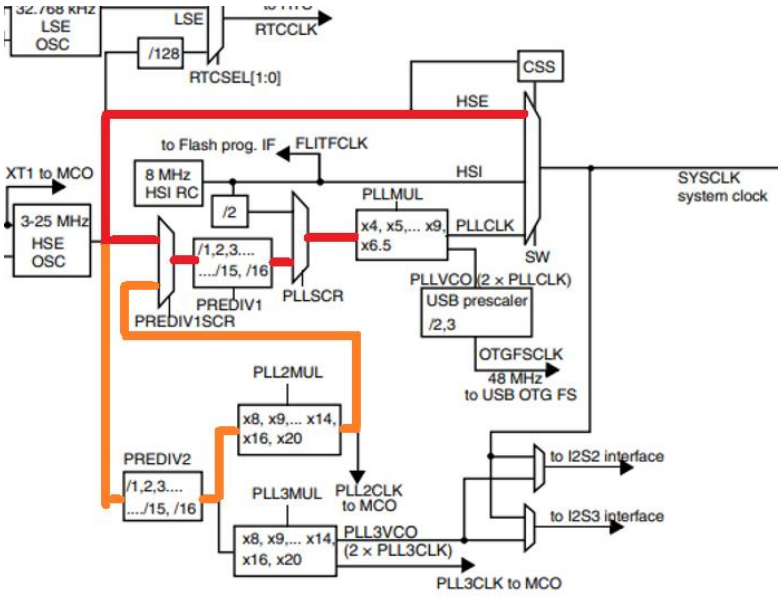
4. User S1 버튼을 누르는 동안 터미널 프로그램(Putty)을 통해 "Hello TeamXX"을 출력 후 줄 바꿈 (다음 "Hello TeamXX"는 다음 줄에서 출력될 수 있도록)

5. MCO를 통해 나오는 System Clock을 오실로스코프로 수치 확인

3. 실험 세부내용

TODO - 1 Set the clock

HSE OSC에서 25 MHz 클럭을 생성하여 원하는 SYSCLK으로 만들기 위해 레지스터를 활용하는 과정이다. HSE에서 생성된 클럭은 PLL클럭을 통해서 SYSCLK인 52MKZ를 맞춰준다.



```
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_PLLMULL4);
//RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL1);

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_PLL2MUL13 | RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV5);
//RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV1 | RCC_CFGR2_PLL2MUL1 | RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV1);
```

RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL4

PLL의 input clock으로 사용하기 위해 사용. PREDIV1와 PLLMULL을 사용하기위해 PLLXTPRE, PLLSRC에 인가. PLLMULL4를 통해 PLL input clock에 *4를 적용. (25 * 4=100MHz)

RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL13 | RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV5

PREDIV2_DIV5를 통해 clock에 /5를 적용. PLL2MUL13를 통해 clock에 *13 적용. PREDIV1SRC_PLL2를 통해 PREDIV1을 사용할것을 암시. PREDIV1_DIV5를 통해 /5 적용. (100 / 5 * 13 / 5 = 52MHz)

TODO - 2 Set the MCO port for system clock output

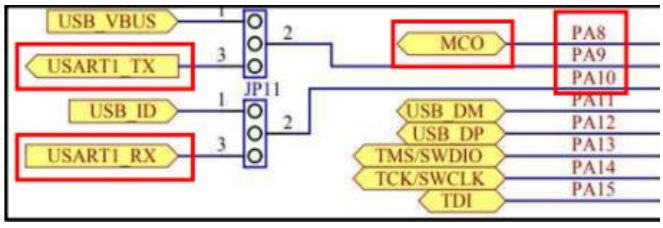
```
//@TODO - 2 Set the MCO port for system clock output
        RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
        RCC->CFGR |= RCC_CFGR_MCO_SYSCLK;
        // RCC->CFGR |= ??
//@End of TODO - 2
```

MCO[3:0]: Microcontroller clock output
Set and cleared by software.
00xx: No clock
0100: System clock (SYSCLK) selected

```
#define  RCC_CFGR_MCO_SYSCLK                ((uint32_t)0x04000000)
```

MCO포트에서 SKSCLK을 사용하기 위해 SYSCLK을 찾아 선택해주는 코드이다. 해당 레지스터에 0100인 MCO_SYSCLK을 할당해준다.

TODO - 3 RCC Setting




```
RCC->APB2ENR |= (uint32_t)RCC_APB2ENR_IOPAEN;
// RCC->APB2ENR |= ??

/* USART RCC Enable */
RCC->APB2ENR |= (uint32_t)RCC_APB2ENR_USART1EN;
// RCC->APB2ENR |= ??

RCC->APB2ENR |= (uint32_t)RCC_APB2ENR_IOPDEN;
 // RCC->APB2ENR |= ??
```

MCO 및 USART 사용을 위한 PORT/PIN 의 GPIO 설정이 필요하다. 사용하기위한 MCO와 USART TX와 RX는 모두 Port A의 8,9,10으로 설정해야한다. PortA를 사용하기위한 APB2ENR_IOPAEN와 USART clock을 사용하기위한 APB2ENR_USART1EN을 작성해주었다. 또한 User S1 버튼을 이용해 Putty통하여 문장을 출력해야 한다. User S1 버튼은 Port D의 11이므로 APB2ENR_IOPDEN으로

Port D까지 사용해준다.

TODO - 4 GPIO Configuration

```
/* MCO Pin Configuration */
/* MCO(PA8) : Alternate function output Push-pull(10) */
GPIOA->CRH |= (uint32_t)(GPIO_CRH_CNF8_1 | GPIO_CRH_MODE8);
// GPIOA->CRH |= ??

/* USART Pin Configuration */
/* TX(PA9) : Alternate function output Push-pull */
/* RX(PA10) : Input with pull-up / pull-down */
GPIOA->CRH |= (uint32_t)(GPIO_CRH_CNF9_1 | GPIO_CRH_MODE9 | GPIO_CRH_CNF10_1 );
// GPIOA->CRH |= ??
```

TODO - 3에서 MCO와 TX, RX 모두 사용할 수 있기에 GPIO설정을 해줘야 한다. MCO 와 TX는 Alternate function output Push-pull(10)로 RX는 Input with pull-up / pull-down(10)로 설정한다. 각각 8, 9 10에서 사용하기에 CNF8_1/ MODE8을 사용해 MCO에 bit (10), CNF9_1/ MODE9 / CNF10_1을 사용해 TX, Rx에 각각 bit(10)을 인가해준다.

```
GPIOD->CRH &= (uint32_t)~(GPIO_CRH_CNF11 | GPIO_CRH_MODE11);
// GPIOD->CRH &= ??

/* User S1 Button Configuration */
/* User S1(PD11) : Input with pull-up / pull-down*/
GPIOD->CRH |= (uint32_t)(GPIO_CRH_CNF11_1);
// GPIOD->CRH |= ??
```

TODO - 3에서 Port D 11에 있는 User s1 버튼을 사용하기로 했다. 이때 버튼 또한 Input with pull-up / pull-down(10)로 설정한다. CNF11_1을 사용해 s1 버튼에 bit(10)을 인가해준다.

TODO – 6: WordLength : 8bits

TODO – 7: Parity bit

```
    /* Clear M, PCE, PS, TE and RE bits */
    USART1->CR1 &= ~(uint32_t)(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 - 6: WordLength : 8bits

    /* Set PCE and PS bits according to USART_Parity value */
//@TODO - 7: Parity : None
```

USART1->CR1 &= ~(uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR1_TE | USART_CR1_RE);

```
#define  USART_CR1_M                         ((uint16_t)0x1000)            /*!< Word length */
```

위 CR1 초기화 부분에서 USART_CR1_M 부분은 기본적으로 8비트로 설정되고, Parity bit는 사용하지 않음으로 초기화 된다. 따라서 WordLength bit는 추가적으로 설정할 필요가 없었고, Parity bit를 사용하지 않아야 전송 문자열이 깨지지 않아서 disable상태도 그대로 두었다.

TODO - 8: Enable Tx and Rx

```
    /* Set TE and RE bits according to USART_Mode value */
//@TODO - 8: Enable Tx and Rx
    USART1->CR1 |= (uint32_t)(USART_CR1_TE | USART_CR1_RE);
    // USART1->CR1 |= ??
```

USART에서 Transmitter와 Receiver를 사용해주기 위해 해당 bit를 할당해주어야 한다. CR1_TE와 CR1_RE를 사용하여 Transmitter와 Receiver를 사용할 수 있게 설정하였다.

TODO - 9: Stop bit: 1bit

```
//@TODO - 9: Stop bit : 1bit


    /*---------------------------- USART CR3 Configuration -----------------------*/
    /* Clear CTSE and RTSE bits */
    USART1->CR3 &= ~(uint32_t)(USART_CR3_CTSE | USART_CR3_RTSE);
```

Stop bit는 00으로 초기화 된 상태에서 추가적인 설정을 하지않았다.

Bits 13:12 **STOP**: STOP bits
These bits are used for programming the stop bits.
00: 1 Stop bit
01: 0.5 Stop bit
10: 2 Stop bits
11: 1.5 Stop bit
The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.

위 레퍼런스를 보면 CR2의 STOP bits 부분이 00으로 설정되어 있을 때 1 Stop bit를 가지기 때문이다.

TODO – 10: CTS, RTS : disable

```
//@TODO - 10: CTS, RTS : disable
```

USART1->CR3 &= ~(uint32_t) (USART_CR3_CTSE | USART_CR3_RTSE);

위 코드에서 CTS와 RTS를 disable로 초기화 시켰기 때문에 따로 설정할 필요가 없다.

TODO – 11: Calculate & configure BRR

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

legend: f_{CK} - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

Example 2:

To program USARTDIV = 0d25.62

This leads to:

DIV_Fraction = 16*0d0.62 = 0d9.92

The nearest real number is 0d10 = 0xA

DIV_Mantissa = mantissa (0d25.620) = 0d25 = 0x19

Then, USART_BRR = 0x19A hence USARTDIV = 0d25.625

putty와의 통신에 필요한 Tx/Rx baud를 14400으로 설정하는게 목적이다.

fck(PLCK2의 clock)는 26Mhz 이므로 **USARTDIV**는 26*(10^6) / (16 * 14400) = 112.85이다.

정수 부분이 112, 소수 부분이 0.85이다.

정수 부분 112를 16진수로 변환하면 0x70이 된다.

소수 부분 0.85에 16을 곱하면 13.6이 되고 이를 반올림하면 14가 된다. 14는 16진수로 바꾸면 0xE가 된다.

정수와 소수 부분의 수를 합치면 **USART_BRR = 0x70E**가 된다.

```
//@TODO - 11: Calculate & configure BRR
    USART1->BRR |= 0x70E;
    // USART1->BRR |= ??
```

TODO – 12: Enable UART

```
    /*---------------------------- USART Enable ----------------------------------*/
    /* USART Enable Configuration */
//@TODO - 12: Enable UART (UE)
    USART1->CR1 |= (uint32_t)USART_CR1_UE;
    // USART1->CR1 |= ??
```

Control Register 1의 USRAT 비트를 활성화 시켰다.

TODO – 13: Send the message when button is pressed

```
while (1) {
//@TODO - 13: Send the message when button is pressed
volatile unsigned int arrow = (uint32_t)GPIOD->IDR;
arrow &= GPIO_IDR_IDR11;
arrow >> 11;

if(arrow == 0) {
  for(int i = 0; i < 14; ++i) {
    SendData((uint16_t)msg[i]);
    delay();
  }
}
```

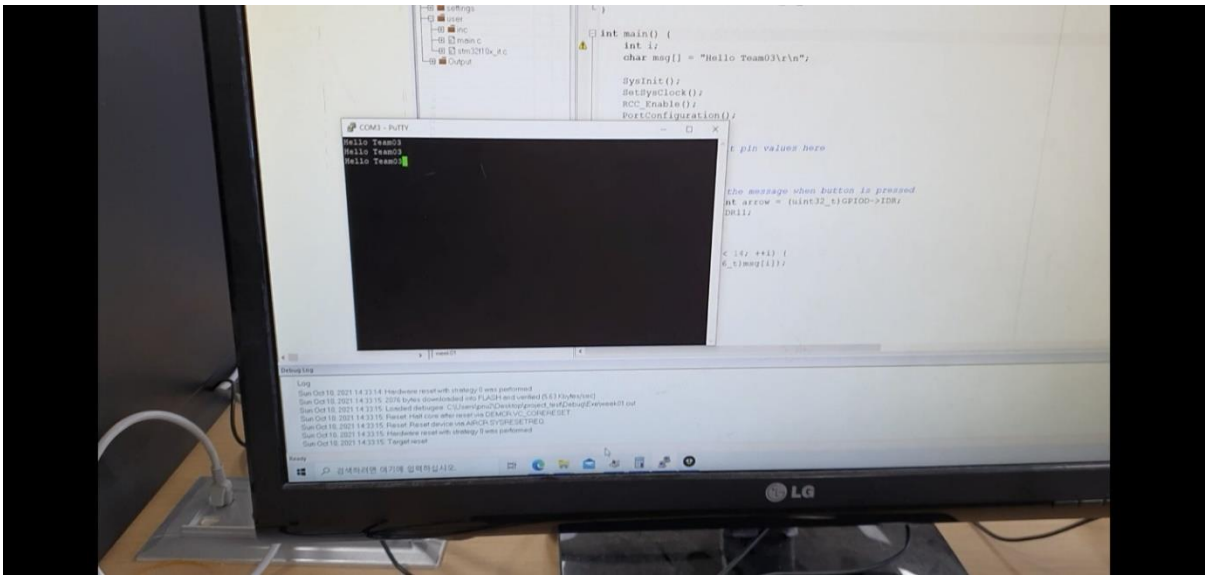
arrow에 GPIOD의 입력을 받도록 한다.

PORT D의 11번 bit를 사용하는 user1 버튼을 사용하기 때문에 11번 IDR값만 남기고 나머지는 초기화 한다.

11만큼 right shift 연산을 통해 버튼의 입력을 0또는 1로 나타낼 수 있다.

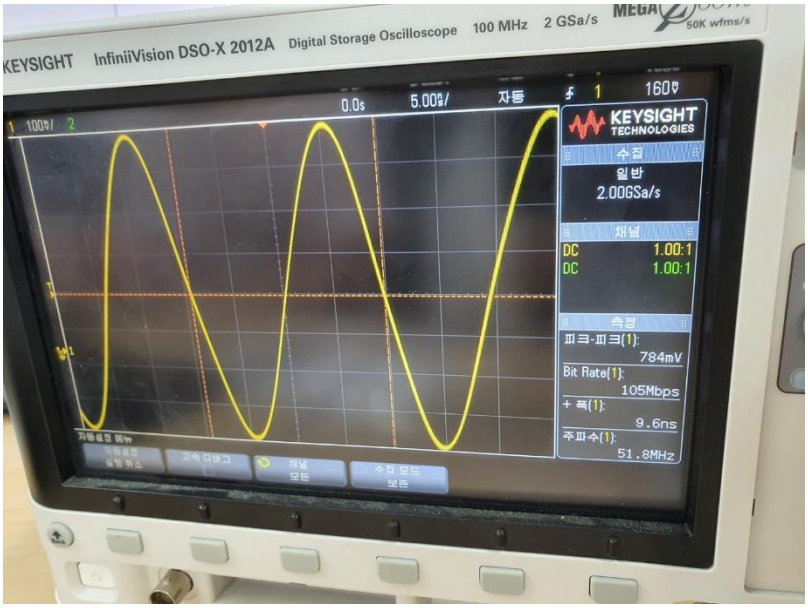
버튼은 pull-up 방식이기 때문에 arrow가 0일 때 누른 것이다. 따라서 arrow==0이면 HelloTeam03을 문자 하나씩 전송한다. Delay가 for문 내부에 있는 이유는 전송되는 도중에 버튼의 입력을 방지하기 위해서이다.

4. 실험결과



(터미널 프로그램(Putty)을 통해 출력)

User s1 버튼을 누를 때 "Hello Team03"이 한 줄씩 출력이 되는 걸 확인할 수 있다.



(오실로스코프 수치 확인)

MCO를 통해 나오는 SYSCLK을 오실로스코프를 통해 확인한다. 주파수가 51.8MHz인 것을 확인할 수 있다.