PNU, the Premier!

[2조] 5주차 결과보고서

Subject	임베디드 시스템 설계 및 실험
Professor	김원석
Major	정보컴퓨터공학부
Date	2022.10.03
Team Member	202055600 정홍빈
	201924617 끼앗띠퐁 유엔
	201824636 이강우
	201824534 윤상호

부산대학교
PUSAN NATIONAL UNIVERSITY

1. 실험 목적

라이브러리를 활용한 코드의 작성과 Clock Tree의 이해와 설정, 오실로스코프를 이용한 Clock 확인 그리고 UART 통신의 원리를 이해하고 실제로 구동한다.

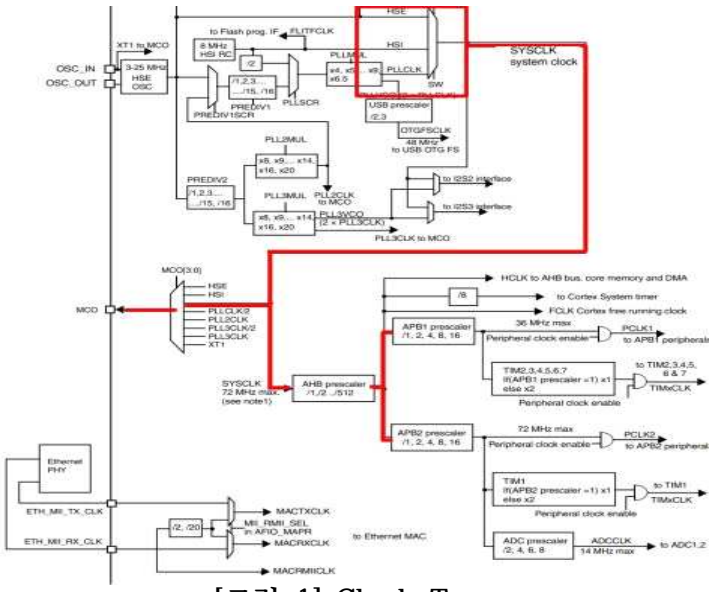
2. 실험 원리 및 이론

1) 라이브러리 활용

기존에 volatile로 타입캐스트해 Configuration이나 enable을 해줬다면 이제는 헤더파일 안에 들어있는 구조체에 접근하거나 정의된 상수를 이용하여 쉽게 설정할 수 있다.

2) Clock의 개념

Clock은 HSI Clock과 Hse Clock이 있다. 기본으로 HSI Clock은 8MHz RC 오실레이터에서 생성되고 HSE Clock은 HSE OSC 25MHz 생성된 Clock은 시스템 클럭이나 PLL 클럭으로 사용할 수 있다.

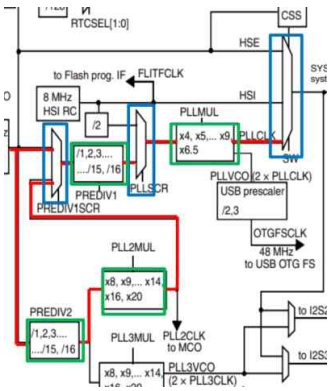


[그림 1] Clock Tree

Clock Tree는 HSE, HSI, PLL 중 하나를 Multiplexer로 선택해 시스템 클럭으로 APB1과 APB2에 전달되고 이때 제대로 출력되는지 MCO Multiplexer를 이용해 MCO에 출력해 오실로스코프로 확인할 수 있다. APB1 prescaler을 이용하면

36MHz, APB2 prescaler은 72MHz까지의 클럭을 각각의 PCLK1과 PCLK2에 출력한다.

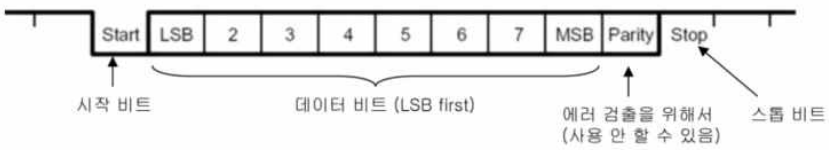
PLL(Phase-Locked Loop)은 위상 동기 회로이고 Multiplexer, DIV, MUL을 거치며 입력 신호와 출력 신호를 통해 사용자가 원하는 주파수의 출력 신호를 제어한다.



[그림 2] PLL(Phase-Locked Loop)

3) 시리얼 통신

시리얼통신은 직렬 통신과 병렬 통신이 있다. 병렬 통신은 여러 개의 데이터 선을 이용해 비트를 하나씩 보내지만 비용이 많이 들고, 직렬 통신은 하나의 데이터 선으로 전송의 시작을 의미하는 Start bit, 정보를 담고 있는 Databit, 오류의 여부를 확인하는 Parity bit와 데이터 통신의 종료를 의미하는 Stop bit로 구분하는 방식이다.

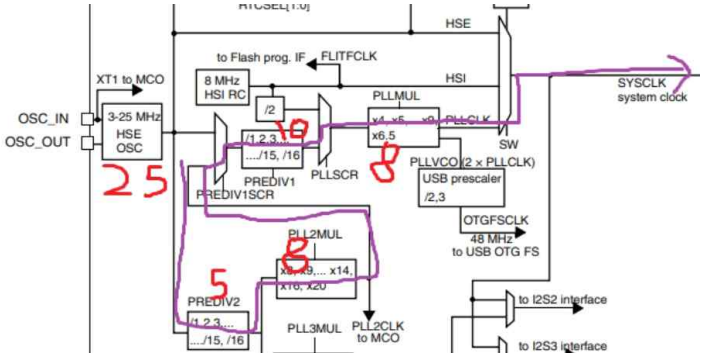


[그림 3] Universal Asynchronous Receiver/Transmitter

3. 실습 진행 과정

미션지의 순서를 따라가며 실습을 진행하였다.

제공된 파일의 TODO 중 수정이 필요한 TODO만 수정해 프로그램을 정상적으로 실행한다. 미션지에 주어진 SYSCLK은 32MHz로 출력되어야 한다. 이를 토대로 회로에서 어떤 식으로 이동해야 좋을지 설계한다.



[그림 4] 회로 설계

[그림 4]를 보면 25 / 5 * 8 / 10 * 8를 거쳐 32의 값으로 SYSCLK에 출력한다.

TODO - 1 Set the clock

stm32f10x.h 헤더 파일을 찾아보면 코드의 의미가 무엇인지 알 수 있다.

```
RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;
```

이 두 코드의 뜻은 SYSCLK과 HCLK를 나누지 않고 그대로 쓴다는 뜻이다.

RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1으로 되어있는데 **[그림 4]**를 거친 SYSCLK을 PCLK2에서 16MHz로 만들기 위해 RCC_CFGR_PPRE2_DIV2로 수정해 입력값을 2로 나눈다.

```
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;
```

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

RCC_CFGR_PLLSRC_PREDIV1의 뜻은 PREDIV1 clock이 MUX에서 PLL을 선택하는 옵션이며 비트 연산자 or로 CC_CFGR_PLLMULL1은 PLL로 들어오는 값을 곱하려는 의도인데 설계한 회로를 보면 PLL 값에 8을 곱해준다.

```
RCC->CFGR |= (uint32_t)( RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL8);
```

이 코드를 분석해보면

RCC_CFGR2_PREDIV2_DIV1 | RCC_CFGR2_PLL2MUL1 | RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV1

RCC_CFGR2_PREDIV2_DIV1 = PREDIV2 input clock not divided
RCC_CFGR2_PLL2MUL1 = 헤더파일에 존재하지 않음
RCC_CFGR2_PREDIV1SRC_PLL2 = PLL2 selected as PREDIV1 entry clock source
RCC_CFGR2_PREDIV1_DIV1 = PREDIV1 input clock not divided

위의 의미를 가진다. **[그림 4]**에 의하면 수정해야 할 점은 PREDIV2를 5로 DIV, PLL2는 8로 MUL, PREDIV1SRC는 PLL2를 선택, PREDIV1은 10으로 DIV해야 한다.

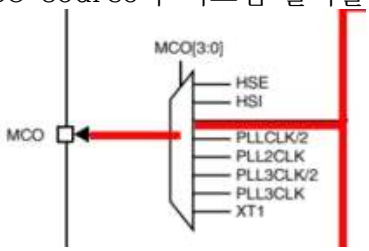
RCC_CFGR2_PREDIV2_DIV5 = PREDIV1 input clock divided by 5
RCC_CFGR2_PLL2MUL8 = PLL2 input clock * 8
RCC_CFGR2_PREDIV1_DIV10 = PREDIV1 input clock divided by 10
RCC_CFGR2_PREDIV1SRC_PLL2 = PLL2 selected as PREDIV1 entry clock source
로 수정하고 각각을 비트 or 연산 시킨다.

TODO - 2 Set the MCO port for system clock output

RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO
RCC 구조체의 MCO의 비트에 접근해 negate 시킨 후 비트 and 연산 시켜 초기화했다. 초기화 후엔 MCO를 설정해주면 되는데 헤더 파일을 찾아보면

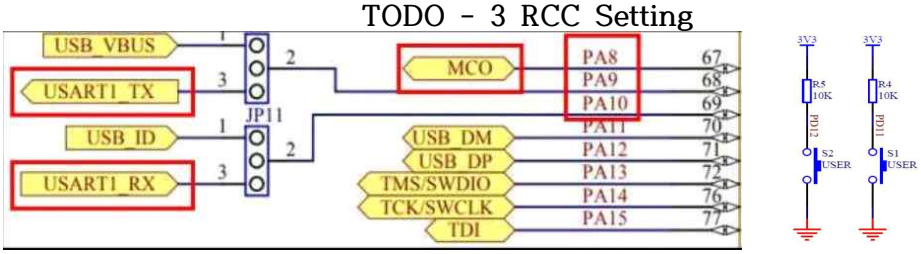
RCC_CFGR_MCO_SYSCLK = System clock selected as MCO source

이를 이용해 그림과 같이 MCO source가 시스템 클락을 선택하게 설정할 수 있다.



[그림5] MCO가 [그림4]의 회로를 거친 SYSCLK을 MUX로 선택

TODO - 3 RCC Setting



[그림 6] GPIO설정에 필요한 MCO, TX, RX, S1 Button 의 PORT/PIN

[그림 6]을 보면 MCO, TX, RX, S1에 접근하기 위해선 PORT A와 D를 Enable 및 GPIO Configuration 해야 한다.

RCC->APB2ENR |=

RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPDEN = Port A와 D Enable
RCC_APB2ENR_USART1EN = USART1 clock enable

주석 처리된 물음표 자리에 bit or 연산해 PORT A와 D를 Enable, USART1을 Enable 하는 비트를 |=로 활성화한다.

TODO - 4 GPIO Configuration

GPIOA->CRH |= ??를 만나게 되는데 MCO PIN과 USART PIN의 Configuration을 해준다.

GPIOA->CRH: GPIOA 구조체의 멤버 변수 CRH를 가리켜 PORT A의 GPIO를 바꿔준다.
문서에 설정된 Congifuration 값을 살펴보자.

MCO 및 UART TX: Alternate function output Push-pull 으로 설정
10(Alternate function output Push-pull) **01**(Output mode, max speed 10 MHz)

UART RX: Input with pull-up / pull- down 으로 설정
10(Input with pull-up / pull-down) **00**(Input mode (reset state)) 로 설정.

GPIO_CRH_CNF8_1 | GPIO_CRH_MODE8_0
8번째, 즉 MCO의 CNF를 10 MODE를 01로 바꿔준다.

GPIO_CRH_CNF9_1 | GPIO_CRH_MODE9_0 | GPIO_CRH_CNF10_1
9번째, TX를 1001 RX를 1000으로 설정한다.

MCO, RX와 TX를 설정한 방법과 동일하게.
GPIOD->CRH: 이제는 S1 Button의 설정을 위해 PORT D의 구조체에 접근해 멤버 변수 CRH를 초기화한다. 그 후에 입력을 받는 S1을 1000으로 설정한다.

GPIOD->CRH &= ~(uint32_t)GPIO_CRH_CNF11
PORT D에 접근해 11번째 CNF 초기화

GPIOD->CRH |= (uint32_t)GPIO_CRH_CNF11_1;

11번째 CNF 10으로 변경

TODO - 8: Enable Tx and Rx

이제 Configuration이 끝나 TE 와 RE bits를 켜야한다.

USART_CR1_TE : Transmit Data Enable
USART_CR1_RE : Receive Data Enable

USART1->CR1 |= (uint32_t)(USART_CR1_TE | USART_CR1_RE) 로 설정한다.

TODO - 11: Calculate & configure BRR

Baud란 일반적으로 데이터의 통신에서 사용하는 BPS(bit per seconds)는 초당 보낼 수 있는 비트의 수라면, Baud는 비트들을 심벌로 묶어 한 번의 신호에 보내는 것을 말한다. 이제 Baud Rate를 계산해 16진수로 대입한다.

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

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

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

[그림 7] Baud Rate 계산식

실험에서 쓰는 Baud Rate = 14400, PCLK2는 16MHz이므로 식에 대입하면

$$\frac{16MHz}{16\times 14400} = \frac{1{,}000{,}000Hz}{14400} = 69.44$$

정수부인 69를 16진수로 변환하면 0x45, 가수부인 44를 대입하면 $0.44\times 16 = 7.04$ 로 가장 가까운 정수인 7로 만들어 BaudRate를 구하면 0x457이 된다.

USART1->BRR = 0x457;

TODO - 12: Enable UART (UE)

USART를 Enable 해야 한다. USART1 구조체의 CR1에 접근하여 USART Enable을 bit or 연산해 준다.

```
USART1->CR1 |= (uint32_t)USART_CR1_UE;
```

TODO - 13: Send the message when button is pressed

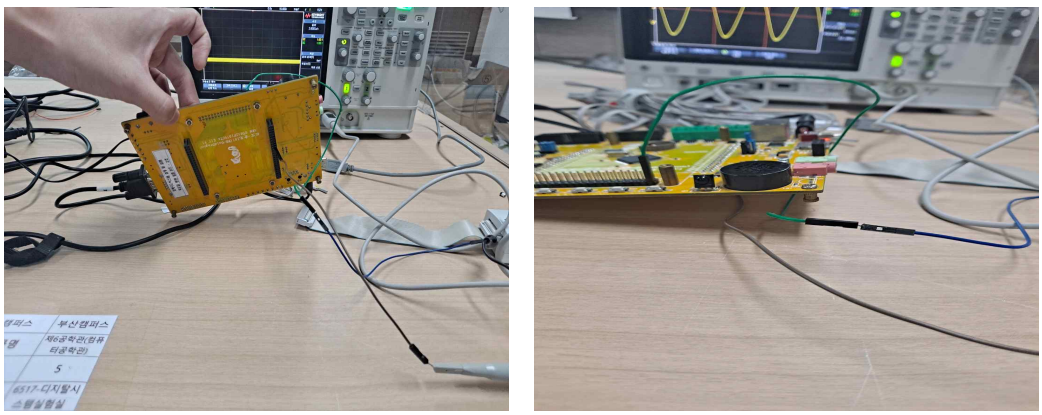
메인 함수로 넘어가서, 무한 루프를 도는 While문 안에 S1 Button이 눌렸을 때, 설정되어 있는 Char배열 “Hello Team02\r\n”이 출력되면 된다. 이미 SendData 함수가 정의되어있어, if문으로 S1 Button이 눌리면 Input Data Register bit and 연산으로 S1이 눌린 SELECT_NUMBER bit와 같다면 delay 함수 실행 후에 for 문으로 msg 배열을 char 하나씩 출력한다.

```
while (1) {
  //@TODO - 13: Send the message when button is pressed
  if((GPIOD->IDR & SELECT_NUMBER) != SELECT_NUMBER){
    for(i = 0; i < 15; i++){
      SendData(msg[i]);
    }
    delay();
  }
}
```

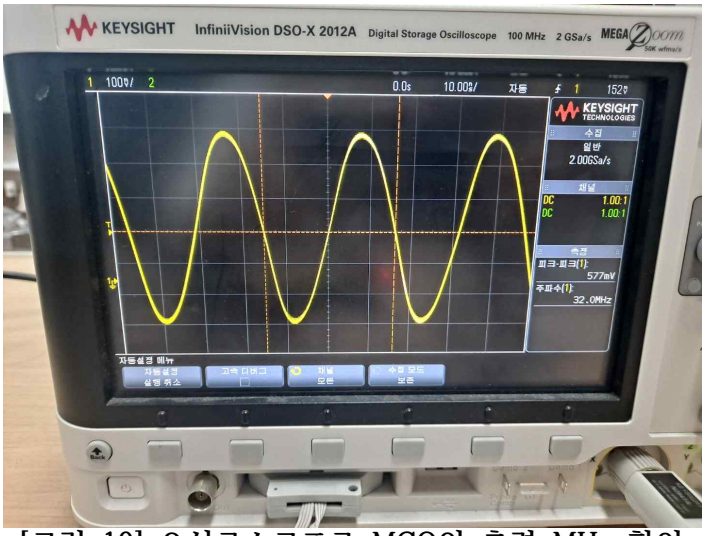
```
#include "stm32f10x.h"
#define SELECT_NUMBER 0x800
```

[그림 8] Main 함수에 선언된 반복문과 매크로상수

[그림 5]를 보면 SYSCLK의 출력이 MCO로 출력되는데, 이 값이 제대로 출력되는지 오실로스코프로 점검한다. **[그림 6]**을 참고하면 MCO는 PA8에 연결되어 있으니 STM32보드의 PA8을 오실로스코프로 측정한다.



[그림 9] 오실로스코프와 STM32보드(Ground, PA8) 연결

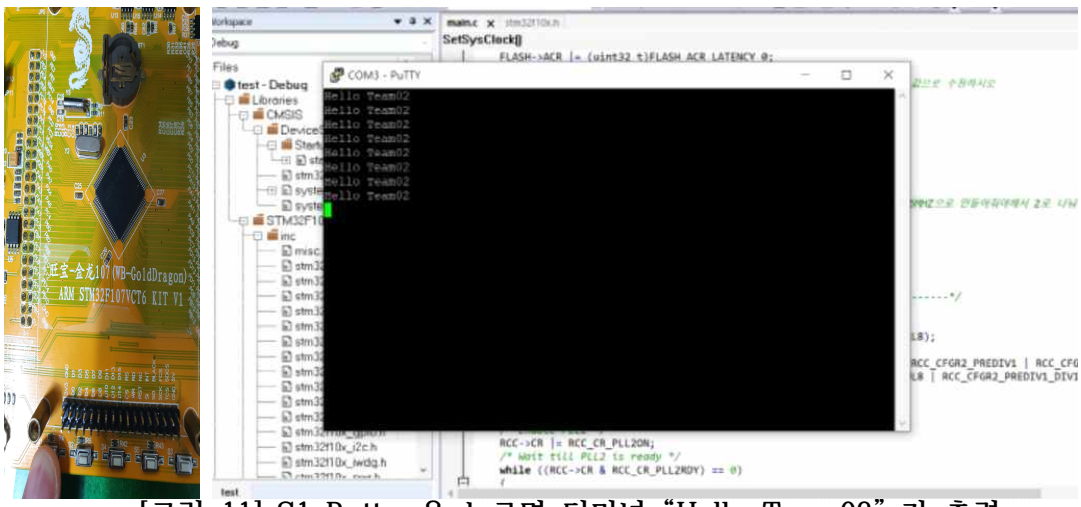


[그림 10] 오실로스코프로 MCO의 출력 MHz 확인

[그림 9] 와 같이 보드를 핀으로 연결 후, **[그림 10]**을 확인하면 MCO의 출력이 32Mhz로 나와, 설계한 대로 회로가 동작했음을 알 수 있다.

4.실험 결과

이제 가상 터미널 PUTTY를 이용해 S1 Button을 눌렀을 때, 터미널에 설정된 문구가 정상적으로 출력되는지 확인해 보자.



[그림 11] S1 Button을 누르면 터미널 “Hello Team02”가 출력

5. 결론

5주 차 실험은 SYSCLK으로 출력할 값을 MUX, DIV, MUL을 이용해 원하는 값으로 만들어주는 과정이 가장 까다로웠고 시간이 많이 들었다. 실험 막바지에 글자가 자꾸 깨져 출력되어서 처음부터 훑어보니. 여기서 SYSCLK에서 출력되는 값을 DIV2 해주는 것을 하지 않아 막히기도 했다.

기존에 비트를 16진수로 계산하고 volatile로 타입 캐스트해 대입하는 귀찮은 과정 대신. 기존 라이브러리의 구조체나 상수를 가져오는 것으로 코드를 작성할 수 있어 한결 수월했다.

저번 실험까지는 STM32보드 안에서 입력과 출력이 일어났는데, 이번 실험은 외부와 시리얼통신을 이용해 출력을 내보냈다. 추후의 실험에서는 외부의 입력을 받아 STM32보드의 출력을 할 수 있을 것이라고 유추해 본다.