임베디드 시스템 설계 및 실험
6주차 실험 결과 보고서

조 : 8조

조원 : 서진욱, 이승민, 하연지, 하태훈

● 실험목표

1. **Clock Tree의 이해 및 사용자 Clock 설정**
2. **UART 통신 원리 파악 및 실제 설정 방법 파악**

● 개념설명

1. Clock Tree

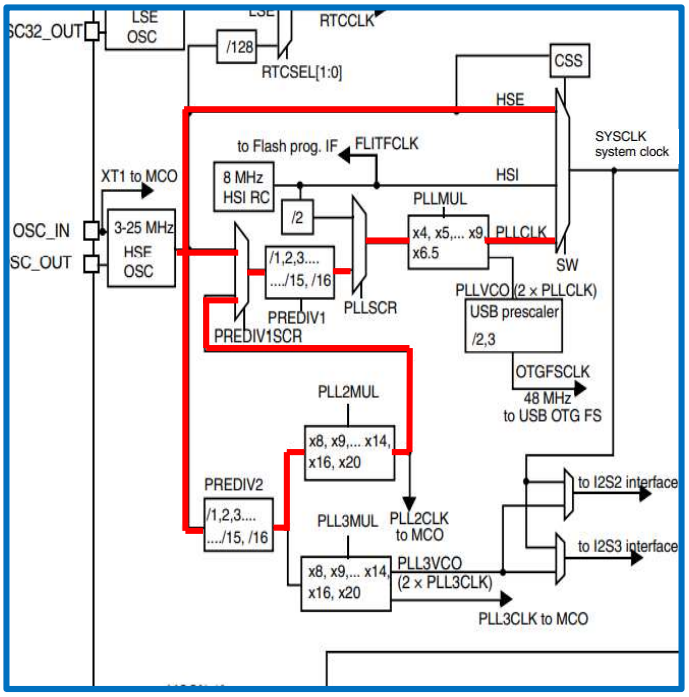


그림 1 SYSCLK

A. HSI(High Speed Internal) Clock : 8MHz의 RC 오실레이터에서 생성되는 Clock
B. HSE(High Speed External) Clock : 25MHz의 RC 오실레이터에서 생성되는 Clock
C. PLL(Phased Locked Loop) : 위상 동기 회로로, 입력된 신호에 맞추어 출력 신호의 주파수를 제어하는 시스템
D. SYSCLK(System Clock)
   i. 시스템 Clock으로, HIS, HSE을 별도의 조정 없이 바로 사용하거나 PLL을 통해 출력 신호의 주파수를 제어하여 원하는 Clock(PLLCLK)으로 설정할 수 있음
E. HCLK : DMA, Core memory, AHB Bus에서 사용되는 Clock
F. FCLK : Cortex System(CPU)에서 사용되는 Clock

G. PCLK : APB Bus에 사용되는 Clock

H. PCLK1은 APB1 Prescaler를 통해 최대 36MHz의 클럭으로 설정될 수 있고, PCLK2는 APB2 Prescaler를 통해 최대 72MHz의 클럭으로 설정될 수 있음

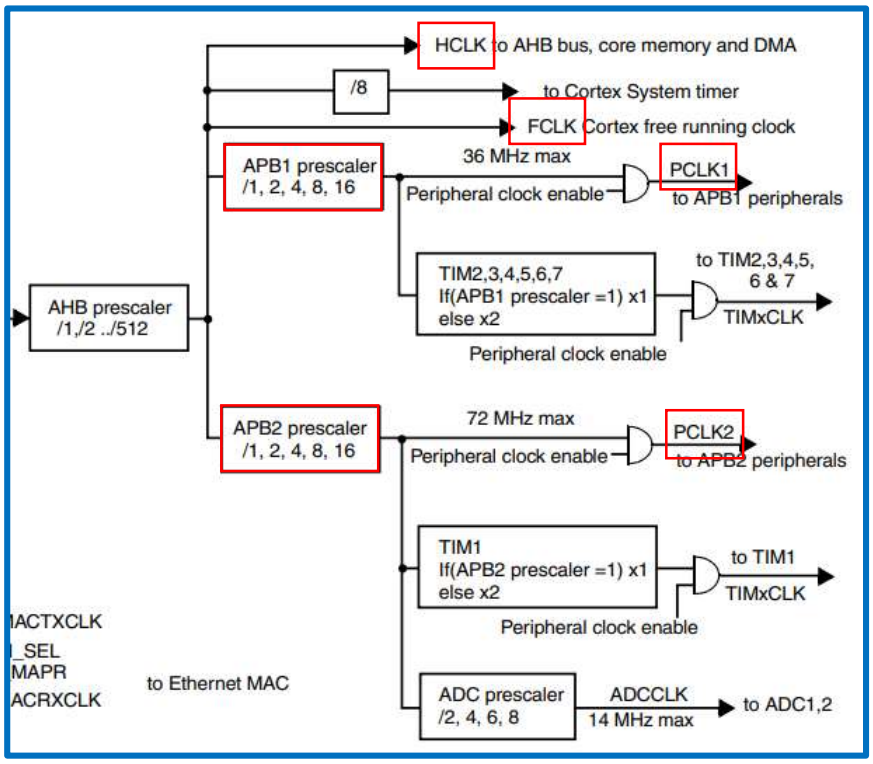


그림 2 HCLK, FCLK, PCLK

2. UART 통신

A. 데이터를 직렬 방식으로 전송하는 형태의 통신으로, 하나의 데이터 선을 이용해 비트를 차례로 보내는 방법. 속도는 느리나 데이터 선을 연장하기 위한 비용이 적은 것이 특징

B. 비동기 통신 방법으로, Clock을 위한 별도의 선은 필요 없지만 데이터의 시작을 알리는 Start bit와 끝을 알리는 Stop bit가 있음. 이때 Stop Bit는 레지스터에 따라 1, 1.5, 2bit로 설정함. Parity Bit는 에러 검사를 위한 것으로, 수신자가 해당 bit를 이용해 에러를 검사함

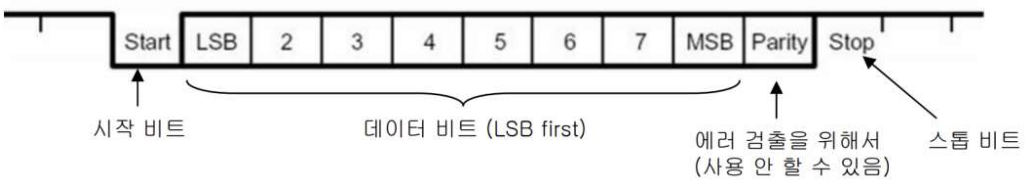


그림 3 UART 통신 시 전송 데이터 구조

C. Baud Rate

i. 초당 보낼 수 있는 데이터의 크기를 의미하는 것으로, 통신 속도를 일정하게 해 통신을 하게 함

● 실험 진행

• 코드 작성

1. Set the clock

이번 실험과제에서는 SYSCLK를 52MHz, PCLK2는 26MHz로 설정을 하고 Baud Rate를 9600으로 설정해야한다.

이 코드는 Clock 도메인 설정을 변경하기위한 코드와 PLL 설정을 위한 코드이다. Clock 도메인 설정에서 HCLK를 SYSCLK으로 설정하고 RCC_CFGR_PPRE2_DIV2를 사용해 PCLK2를 HCLK의 절반으로 설정했으며 PCLK1을 HCLK로 설정하였다.

또한 실험과제에 나와있는 MHz를 맞추기위해 초기값 25에서 어떤 연산을 수행해야하는지를 계산해 PLL 설정을 변경해주었다.

```
RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
/* PCLK2 = HCLK / ?, use PPRE2 */
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;
/* PCLK1 = HCLK */
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;

/* Configure PLLs ------------------------------------------------------*/
RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL4);

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);
```

2. Set the MCO port for system clock output

System clock 신호를 MCO port로 출력

```
RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK;
```

3. RCC setting

GPIO와 UART의 RCC설정을 변경하는 부분으로 RCC_APB2ENR_IOPAEN 와 RCC_APB2ENR_AFIOEN을 OR연산해 GPIOA 및 AFIO의 RCC clock을 활성화시키고 RCC_APB2ENR_USART1EN을 설정해 USART1의 RCC를 활성화시킨다. 마지막 줄 코드는 s1버튼에 대한 RCC설정이므로 GPIOC의 RCC clock을 활성화시킨 것이다.

```
/* GPIO RCC Enable  */
/* UART Tx, Rx, MCO port */
RCC->APB2ENR |= ((RCC_APB2ENR_IOPAEN)| (RCC_APB2ENR_AFIOEN));
/* USART RCC Enable */
RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    /* User S1 Button RCC Enable */
RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;
```

4. GPIO configuration

MCO 및 UART사용에 대한 PORT/PIN의 GPIO설정이 필요해 설정을 해주는 코드로 MCO 및 UART TX는 Alternative Push-pull로 설정을 하고 UART RX는 Input with pull-up / pull-down으로 설정한다. 그리고 Port C의 CRL 레지스터를 초기화하고 s1 버튼 핀에 대한 설정을 입력모드로 하기 위해 CNF4_1로 설정한다.

```
/* Reset(Clear) Port A CRH - MCO, USART1 TX,RX*/
GPIOA->CRH &= ~(
        (GPIO_CRH_CNF8 | GPIO_CRH_MODE8) |
        (GPIO_CRH_CNF9 | GPIO_CRH_MODE9) |
        (GPIO_CRH_CNF10 | GPIO_CRH_MODE10)
    );
/* MCO Pin Configuration */
GPIOA->CRH |= (GPIO_CRH_CNF8_1 | GPIO_CRH_MODE8);
/* USART Pin Configuration */
GPIOA->CRH |= ((GPIO_CRH_CNF9_1 | GPIO_CRH_MODE9) | GPIO_CRH_CNF10_1);

/* Reset(Clear) Port C CRH - User S1 Button */
GPIOC->CRL &= ~((GPIO_CRL_CNF4 | GPIO_CRL_MODE4));
/* User S1 Button Configuration */
GPIOC->CRL |= GPIO_CRL_CNF4_1;
```

5. Enable TX & RX

```
USART1->CR1 |= USART_CR1_TE | USART_CR1_RE;
```

6. Calculate & Configure USART_BRR

```
USART1->BRR &= ~(uint32_t)(USART_BRR_DIV_Fraction | USART_BRR_DIV_Mantissa);
USART1->BRR |= 0xA94;
```

USART1의 BRR값을 계산하고 설정하는 코드로 BRR계산은 ReferenceManual 27.6.3 을 참고하였다.

27.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV)

Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

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

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

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

The baud counters are updated with the new value of the Baud registers after a write to USART_BRR. Hence the Baud rate register value should not be changed during communication.

How to derive USARTDIV from USART_BRR register values

Example 1:

If DIV_Mantissa = 0d27 and DIV_Fraction = 0d12 (USART_BRR = 0x1BC), then

Mantissa (USARTDIV) = 0d27

Fraction (USARTDIV) = 12/16 = 0d0.75

Therefore USARTDIV = 0d27.75

7. Enable UART(UE)

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

8. Delay 함수

```
void delay(void){
    int i = 0;
    for(i=0;i<1000000;i++);
}
```

9. SendData 함수

SendData는 USART1을 사용해 데이터를 전송하는 역할을 한다. 함수의 인자로 전달된 data변수를 USART1의 DR(데이터 레지스터)에 저장하고 TC(Transmission complete)가 설정될 때까지 기다린다.(TC가 설정되면 성공적으로 전송이 된 것)

```
void SendData(uint16_t data) {
    /* Transmit Data */
        USART1->DR = data;

        /* Wait till TC is set */
        while ((USART1->SR & USART_SR_TC) == 0);
}
```

보내려는 데이터인 "Hello Team08"을 선언해준다.

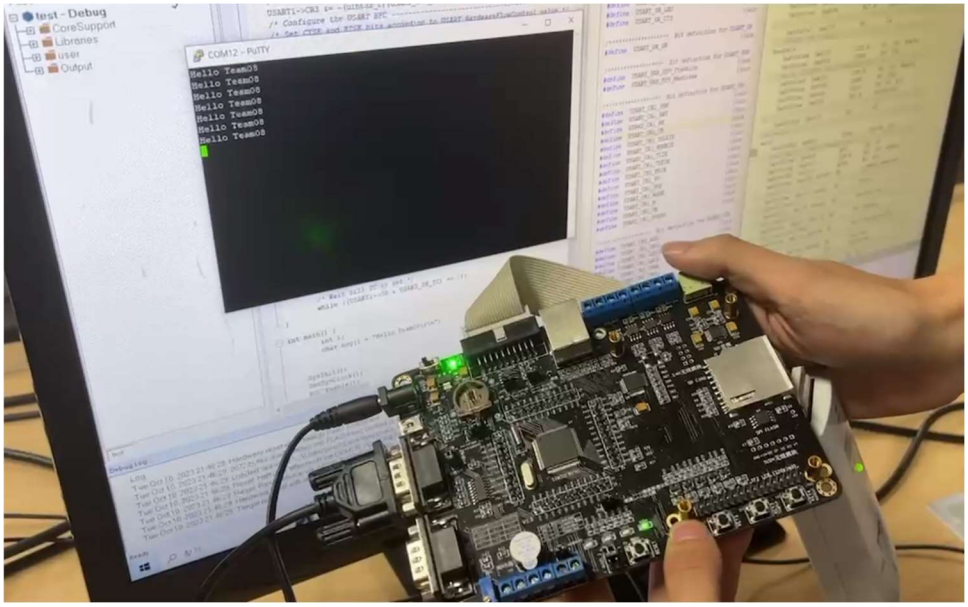
```
int i;
char msg[] = "Hello Team08\r\n";
```

10. Send the message when button is pressed

버튼을 눌렀을 때 "Hello Team08"이 전송되도록 수행하는 무한루프문

```
while (1) {
            //@TODO - 13: Send the message when button is pressed
  if (~GPIOC->IDR & (0x10)){
    for(i = 0; i < 15; i++){
      SendData(msg[i]);
    }
    delay();
  }
}
```

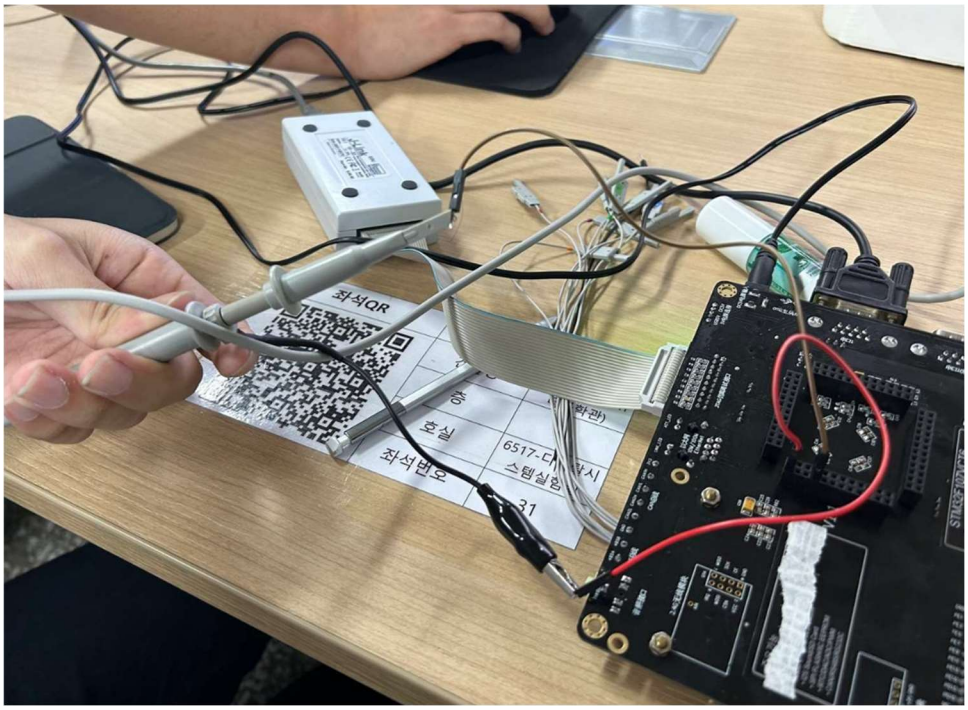
- **코드실행 및 작동 확인**



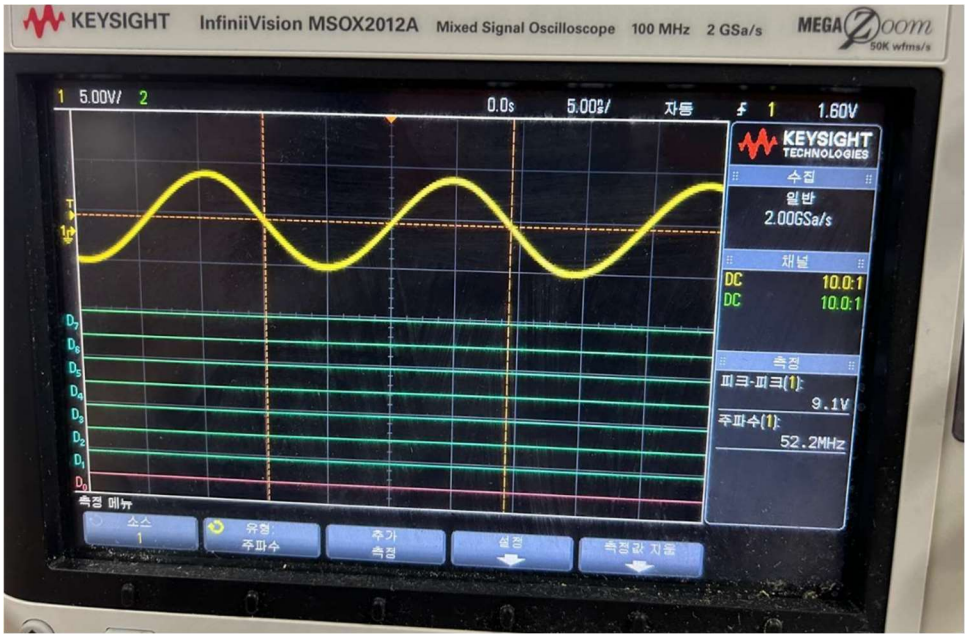
버튼1을 누르면 시리얼 케이블로 연결된 PC의 Putty에 메시지가 출력됨.

- **MCO를 통해 나오는 System Clock을 오실로스코프로 수치 확인**

1. 오실로스코프에 보드의 핀을 연결



2. 오실로스코프에 나온 파형 및 Clock(주파수 확인). SYSCLK로 설정한 52MHz가 정상적으로 확인.



● 결론

- 라이브러리를 활용해 직접 메모리의 주소로 접근하는 것 보다 더 직관적이고 간단하게 코드를 짤 수 있다.

- 시스템 및 사용자 Clock 을 별도로 설정할 수 있으며, 설정하는 방법을 알게 되었고, 시리얼케이블 연결 및 Putty를 통한 UART 통신에 대해 알게 되었다.