임베디드 시스템 설계 및 실험
화요일 분반 5주차 실험 결과 보고서

조 : 6조

조원 : 이주승 김선규 이동현 이지현 최세희

실험 목표

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

세부 목표

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	9600

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

이론적 배경

1. Clock

Cortex-M3 보드에는 다음 3가지 clock을 시스템 클럭으로 사용한다.

① HSE (External High-Speed Clock) : HSE OSC에서 25MHz 클럭이 생성되며 바로 시스템 클럭으로 사용하거나, PLL Clock으로 사용가능하다.

② HSI (Internal High-Speed Clock) : 8MHz RC 오실레이터에서 생성된다. 생성된 clock은 시스템 클럭으로 사용하거나, PLL Clock으로 사용가능하다.

③ PLL(phase-Locked Loop)은 위상 동기 회로로써, 입력신호와 출력신호를 이용해 출력신호를 제어하는 시스템을 말한다. HSI와 HSE를 곱하거나 나누어 원하는 주파수 값을 만들 수 있다.

그림 1의 Clock Tree는 Clock의 이동 경로를 보여준다.

HSI, HSE, PLL 중 SW MUX에 의해 시스템 클럭으로 설정될 수 있다. 시스템 클럭을 이용해 FCLK, HCLK, PCLK값을 구할 수 있다. FCLK는 CPU에서 사용 되는 클락, HCLK는 AHB버스에 사용되는 클락, PCLK는 APB버스에 사용되는 클락으로 GPIO, UART, SPI 등에 사용된다. APB1 prescaler를 이용해 최대 36MHz 클럭을 PLCK1에, APB2 prescaler를 이용해 최대 72MHz클럭을 PCLK2에 제공할 수 있다.

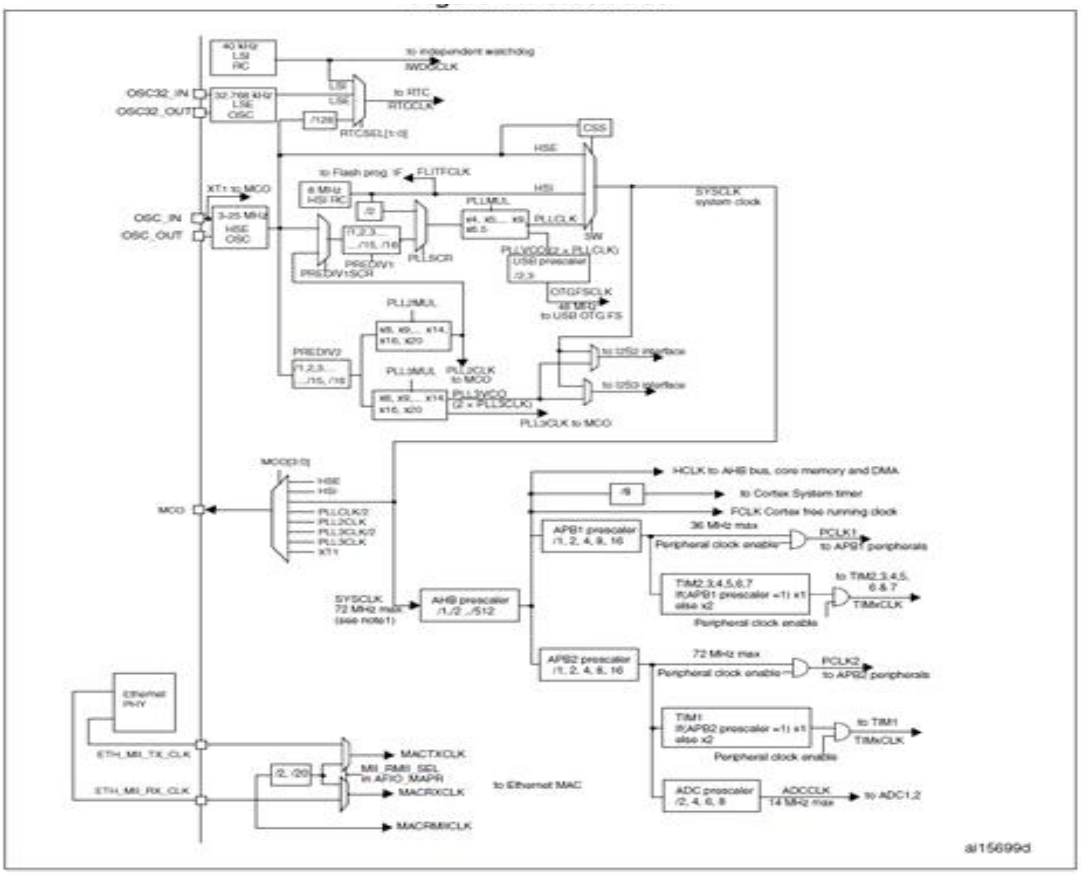


그림 1 clock tree

2. 시리얼 통신

시리얼 통신은 병렬 데이터를 직렬 형식으로 전환하여 데이터를 전송하는 방법으로 아래와 같은 2가지 방법이 있다. 그림2 는 두가지 방법에 대한 그림이다.

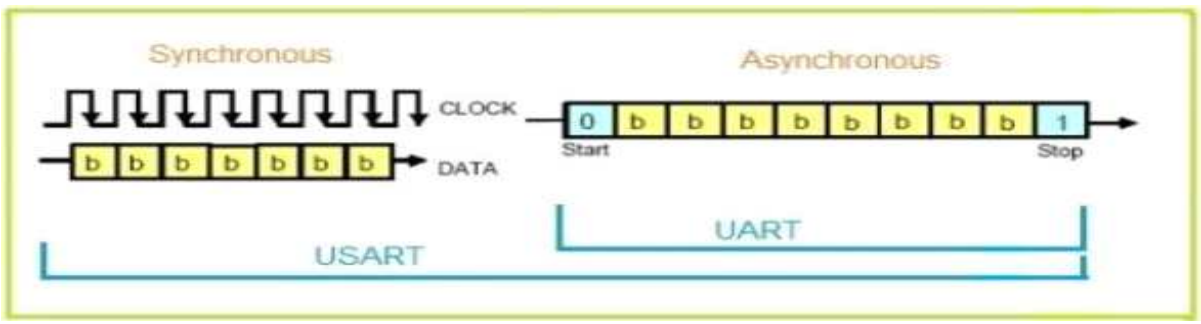


그림 2 UART와 USART

① UART (Universal Asynchronous Receiver/Transmitter)는 비동기 통신 프로토콜이다. 비동기이기 때문에 Baud Rate를 일치시켜야 한다. Rx와 Tx는 교차 연결한다.

② USART (Universal Syn/Asynchronous Receiver/Transmitter)는 동기식 통신을 지원하는 UART이다. 데이터 동기화를 위해 같은 클럭을 사용한다. 따라서 별도의 클럭선을 필요로 한다.

구현 내용

1. 레지스터 초기화

```
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;
}
```

=> SysInit 함수에서는 사용할 Clock 관련 레지스터를 초기화한다. 이전 주차와 달리 직접 주소로 접근하지 않고 라이브러리를 활용해 정의된 주소 값을 사용한다.

① RCC_CR로 내부 Clock을 Enable 시킨다. (HSI On / HSE, PLL Off)

② RCC_CFGR로 HSI가 SYSCLK이 되도록 초기화한다.

③ 각각의 레지스터가 곱하거나 나누는 동작을 하지 않도록 초기화한다.

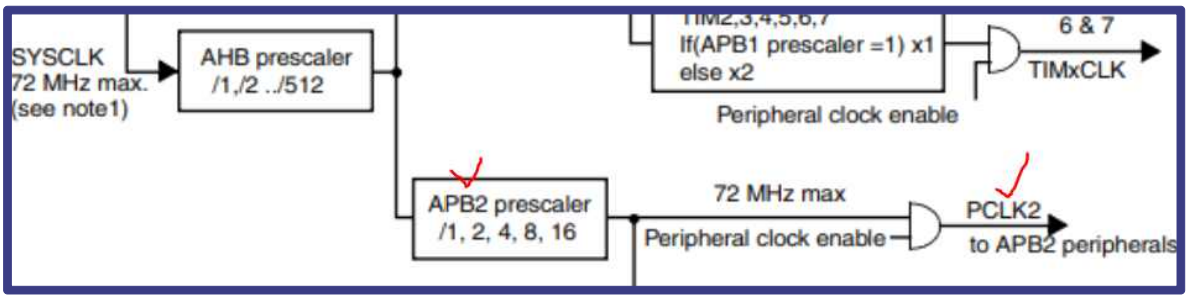
2. Clock 값 설정

SYSCLK	52MHz
PCLK2	26MHz

=> 주어진 Clock 값을 참고하여 PLL 값을 설정한다.

```
//@TODO - 1 Set the clock, (///) 주석 표시를 없애고 틀린 값이 있다면 제대로 된 값으로 수정하시오
    /* HCLK = SYSCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
    /* PCLK2 = HCLK / ?, use PPRE2 */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2; //fixed 26MHz
    /* PCLK1 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;
```

① 먼저 PCLK2는 HCLK를 2로 나눈 값을 사용한다. 수정한 **RCC->CFGR |= RCC_CFGR_PPRE2_DIV2** 에서 PPRE2는 APB2 prescaler이고, DIV2는 2로 나누는 것이다.

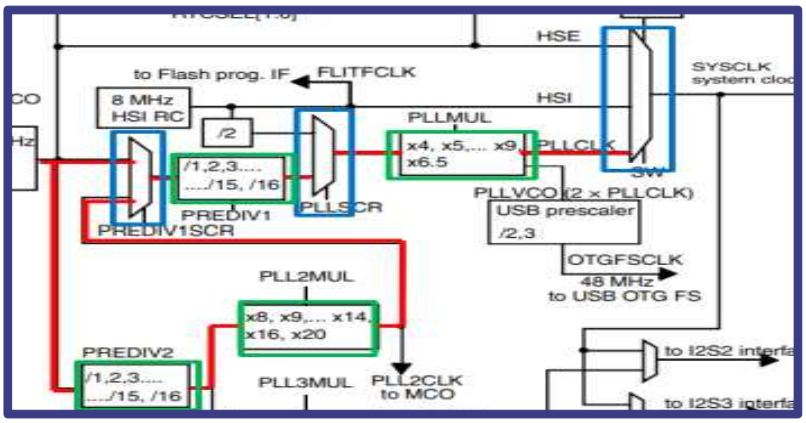


Clock Tree를 참고하면 AHB prescaler에 SYSCLK이 들어가고 주어진 PCLK2 값은 SYSCLK에서 2를 나눈 값이므로, PCLK2는 APB2 prescaler를 거쳐 2로 나눈 값이 들어간다.

```
/* Configure PLLs ------------------------------------------------------*/
RCC->CFGR &= (uint32_t)~( RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
RCC->CFGR |= (uint32_t)( RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL4); //fixed 1-3 bit 4>mul

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 PLL에는 HSE Clock(25MHz) 값을 곱하거나 나누어서 원하는 SYSCLK 값을 만들어준다. 52 = 25 / 5 * 13 / 5 * 4 이므로 아래 사진을 참고해서 값을 넣는다.



PREDIV2 – /5, PLL2MUL – *13, PREDIV1SRC Mux를 거치고, PREDIV1 – /5 이다.
그리고 **PLLSRC Mux를 거쳐, PLLMUL에서는 *4**를 하게 된다.

```
    /* Select System Clock as output of MCO */
//@TODO - 2 Set the MCO port for system clock output
    RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
    RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK; //Fixed SYSCLK
//@End of TODO - 2
```

③ SYSCLK을 MCO MUX를 통해 MCO에 출력해야 하므로, MCO port를 설정한다.

3. RCC 설정

```
//@TODO - 3 RCC Setting
 //Using PA8,PA9,PA10
  /*-------------------------------- RCC Configuration ------------------------------*/
  /* GPIO RCC Enable */
  /* UART Tx, Rx, MCO port */
  RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; //Fixed PA8,PA9,PA10 PA-enable
  /* USART RCC Enable */
  RCC->APB2ENR |= RCC_APB2ENR_USART1EN; //Fixed USART-enable

  RCC->APB2ENR |= RCC_APB2ENR_IOPDEN; //Fixed PD-enable
```

=> MCO 및 URART 사용을 위해 PA를 enable 시키고 USART도 enable 시킨다. 라이브러리에 정의된 값을 사용하였으며 각각 RCC_APB2ENR_IOPAEN, APB2ENR_USART1EN이다.

4. Port 설정

```
//@TODO - 4 GPIO Configuration
  /* 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 |= 0xB; //Fixed PA8 - 10MHz
   /* USART Pin Configuration */
   GPIOA->CRH |= 0x8B0; //Fixed  0x0B0 -> TX
   GPIOA->CRH |= 0x800; //Fixed 0x800 -> RX

  /* Reset(Clear) Port D CRH - User S1 Button */
   GPIOD->CRH &= ~(GPIO_CRH_CNF11 | GPIO_CRH_MODE11); //Fixed PD11 Reset//~0xf000; //Fixed
  /* User S1 Button Configuration */
   GPIOD->CRH |= 0x8000; //Fixed PD11 S1 Button
```

=> MCO, UART1 TX, RX는 각각 PA8, PA9, PA10을 사용하므로 GPIO_CRH를 통해 값을 설정해준다. UART TX는 Alternate function ouput Push-pull, UART RX는 Input with pull-up/pull down으로 설정하였다. Max speed는 10MHz로 설정하였다.

5. UART 통신 설정

```
    /*---------------------------- USART CR1 Configuration -----------------------*/
    /* Clear M, PCE, PS, TE and RE bits */
    USART1->CR1 &= ~(uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR1.
    /* Configure the USART Word Length, Parity and mode ----------------------- */
    /* Set the M bits according to USART_WordLength value */
  //@TODO - 6: WordLength : 8bit
    USART1->CR1 &= ~0x1000; //Fixed

    /* Set PCE and PS bits according to USART_Parity value */
  //@TODO - 7: Parity : None
    USART1->CR1 &= ~0x400; //Fixed Bit10 100 0000 0000 > 011 1111 1111 => 000 0000 0000

    /* Set TE and RE bits according to USART_Mode value */
  //@TODO - 8: Enable Tx and Rx
    USART1->CR1 |= 0xC; //Fixed TE,RE Bit3,Bit2 > 0000 0000 0000 1100
```

=> UART 통신을 사용하기 위해 CR1, 2, 3을 이용해 설정을 변경한다.

27.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

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
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

① Reference를 참고하면, WordLength는 8bit이므로 Bit 12(M)를 0으로, Parity는 disabled이므로 PCE, PS Bit를 0으로, Tx와 Rx는 Enable하므로 TE, RE Bit를 각각 1로 만들어야 한다. 0으로 만들 부분은 & 연산, 1로 만들 부분은 | 연산을 한다.

```
/*---------------------------- USART CR2 Configuration -----------------------*/
/* Clear STOP[13:12] bits */
USART1->CR2 &= ~(uint32_t)(USART_CR2_STOP);
/* Configure the USART Stop Bits, Clock, CPOL, CPHA and LastBit ------------*/
USART1->CR2 &= ~(uint32_t)(USART_CR2_CPHA | USART_CR2_CPOL | USART_CR2_CLKEN);
/* Set STOP[13:12] bits according to USART_StopBits value */
@TODO - 9: Stop bit : 1bit
 USART1->CR2 &= ~0x3000; //Fixed 1100 1111 1111 1111 > 0011 0000 0000 0000

/*---------------------------- USART CR3 Configuration -----------------------*/
/* Clear CTSE and RTSE bits */
USART1->CR3 &= ~(uint32_t)(USART_CR3_CTSE | USART_CR3_RTSE);
/* Configure the USART HFC -------------------------------------------------*/
/* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */
@TODO - 10: CTS, RTS : disable
USART1->CR3 &= ~0x300; //Fixed 1100 1111 1111 > 0011 0000 0000
```

CR2, 3도 동일하게 Reference를 참고해 각각 원하는 설정값에 맞게 비트 연산을 한다.

```
/*---------------------------- USART BRR Configuration -----------------------*/
/* Configure the USART Baud Rate -------------------------------------------*/
/* Determine the integer part */
/* Determine the fractional part */
//@TODO - 11: Calculate & configure BRR
  USART1->BRR &= ~(USART_BRR_DIV_Mantissa | USART_BRR_DIV_Fraction);
  USART1->BRR |= 0xA94 ; //Fixed Baud Rate 9600 > 169-A9, 27->4.3 > 4
```

② USART BRR 설정을 위해서는 Baud Rate 값을 구해야 한다. Baud Rate를 구하는 식을 참고해서 지정된 Baud Rate 값을 넣어 USARTDIV 값을 구한다. 계산 과정은 다음과 같다.


```
/*---------------------------- USART Enable ----------------------------------*/
/* USART Enable Configuration */
//@TODO - 12: Enable UART (UE)
  USART1->CR1 |= 0x2000; //Fixed 13 > 01 > 10 0000 0000 0000
```

③ 마지막으로 USART를 Enable 한다. CR1에서 UE Bit를 1로 만들어준다.

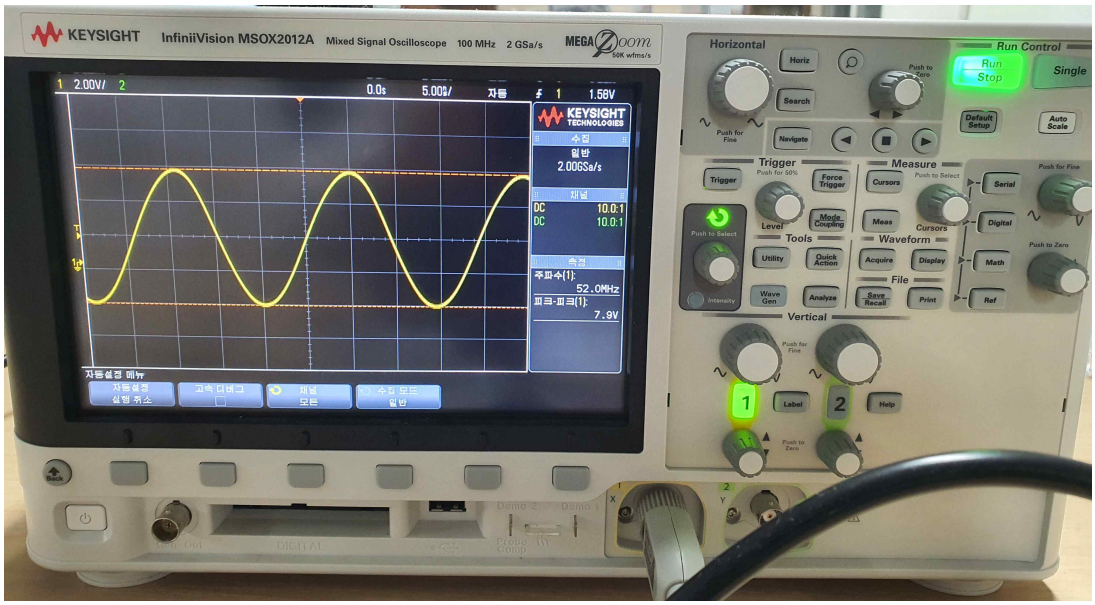
7. main 함수

```
while (1) {
        //@TODO - 13: Send the message when button is pressed
      if (!(*(volatile unsigned int *)0x40011408 & 0x800))//Fixed S1 Button > GPIOD->IDR , PD11
      {
       i = 0;
       char *message = &msg[i];
       while (*message != '₩0'){
         SendData(*message);
         message++;
       }
       delay();
    }
}
```

⑨ main() 내부는 터미널에 띄울 msg를 설정하고, 앞에서 구현한 여러 설정 함수들을 각각 실행한다. 다음으로 반복문을 돌면서 버튼을 누르고 있는 동안 메시지가 UART 통신을 통해 전송되도록 구현한다.

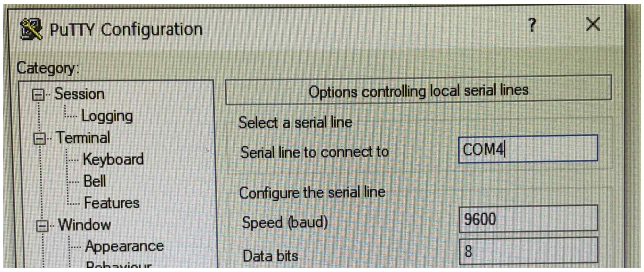
실험 결과

1. 오실로스코프를 통해 MCO에서 나오는 SYSCLK 확인하기

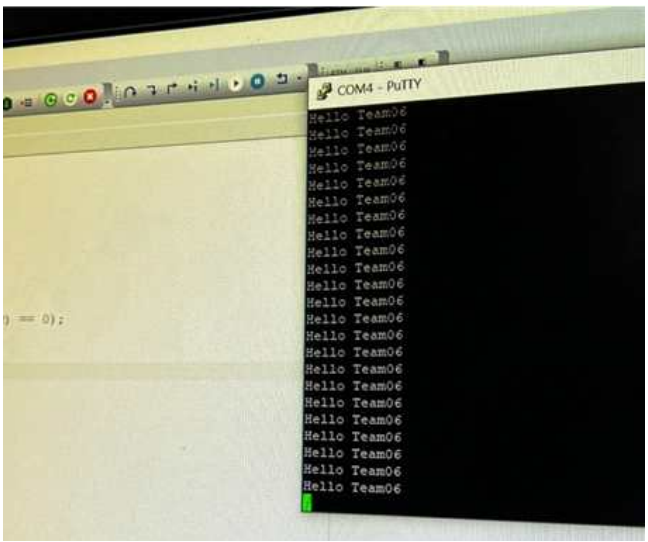


=> 사진과 같이 SYSCLK 52MHz가 잘 나오는 것을 확인할 수 있었다.

2. UART 통신으로 버튼 누르는 동안 Putty로 msg 출력하기



=> PuTTy에서는 다음과 같이 보드와 연결된 Serial Port(COM4), 지정된 Baud Rate(9600) 값을 설정한다.

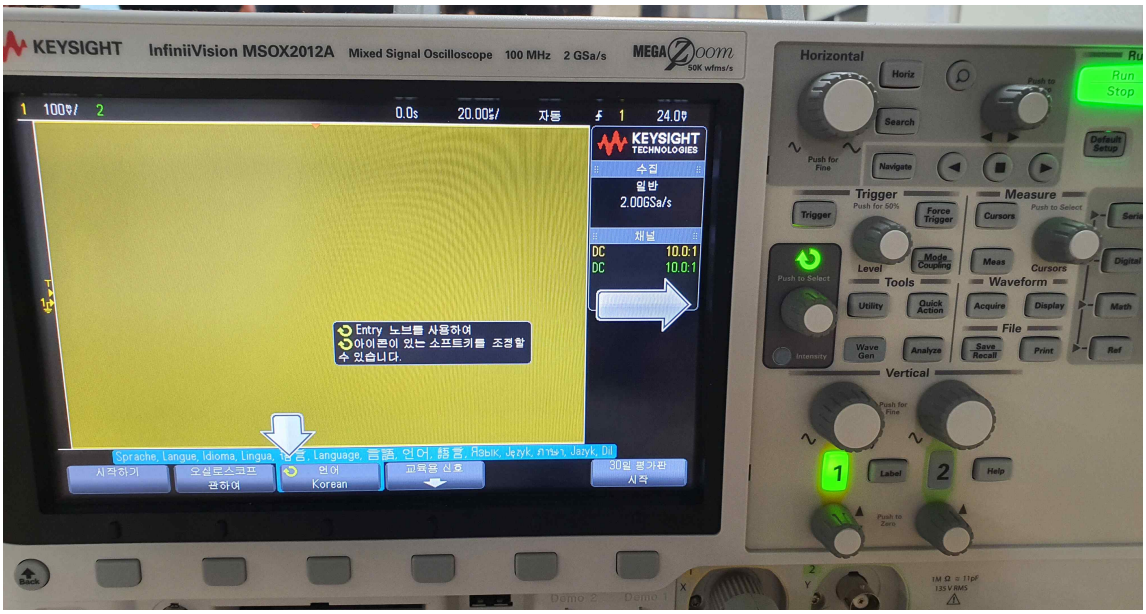


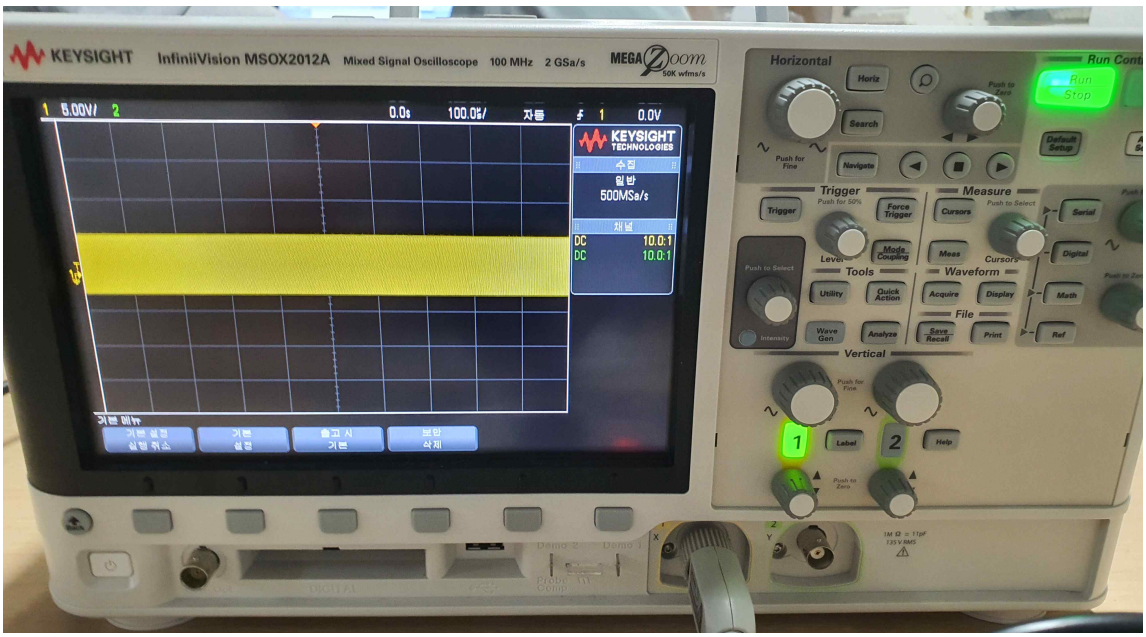
=> 사진과 같이 버튼을 누를 때마다 터미널에 설정한 Hello Team06 메시지가 잘 뜨는 것을 확인할 수 있었다.

어려웠던 점

1. Clock은 자주 접해서 잘 알고 있었지만 Clock Tree를 보고 원하는 SYSCLK을 만드는 과정은 낯설어서 이해하는 데 많은 시간이 걸렸다. 내부에 MUX, DIV을 어떻게 이용해야 하는지, 만든 SYSCLK을 어떻게 MCO에 출력할지 등을 고민해보면서 Clock Tree 구조를 자세히 살펴보았고 Clock의 이동 경로를 잘 이해할 수 있었다.

2. 오실로스코프 설정





오실로스코프를 처음 다뤄봐서 사진과 같이 그래프가 제대로 뜨지 않을 때 어려움을 겪었다. Auto scale 버튼을 눌러 해결할 수 있었다.