임베디드시스템 설계 및 실험

6 주차 결과보고서

2023.10.18

003 (수) 분반

8 조

201924437 김윤하

202055604 조찬우

202155565 성가빈

202012142 이은진

목차

1. 실험 목적

1. 라이브러리를 활용하여 코드 작성
2. Clock Tree 의 이해 및 사용자 Clock 설정
3. 오실로스코프를 이용한 Clock 확인
4. UART 통신의 원리를 배우고 실제 설정 방법 파악

2. 세부 실험 내용

1. DataSheet 및 Reference Manual 을 참고하여 해당 레지스터 및 주소에 대한 설정 이해
2. 예제 코드에서 설정되는 Clock 값을 파악하고, 지정된 Clock 으로 설정
3. 예제 설정 항목에 따라 UART 를 설정하고, 지정된 Baud rate 로 설정

	1 조,3 조,5 조,7 조,9 조,11 조	**2 조,4 조,6 조,8 조,10 조,12 조**
SYSCLK	28MHz	52MHz
PCLK2	14MHz	26MHz
Baud Rate	28800	9600

4. User S1 버튼을 누르는 동안 터미널 프로그램(Putty)을 통해 "Hello TeamXX"을 출력 후 줄바꿈 (다음 "hello TeamXX"는 다음 줄에서 출력될 수 있도록)
5. MCO 를 통해 나오는 System Clock 을 오실로스코프로 수치 확인

3. 실험 과정

1. 프로젝트 생성 및 설정

3 주차, 4 주차 실험과 마찬가지로 프로젝트를 생성하고 설정한다.

2. Main 함수에 주어진 문제(@TODO) 해결

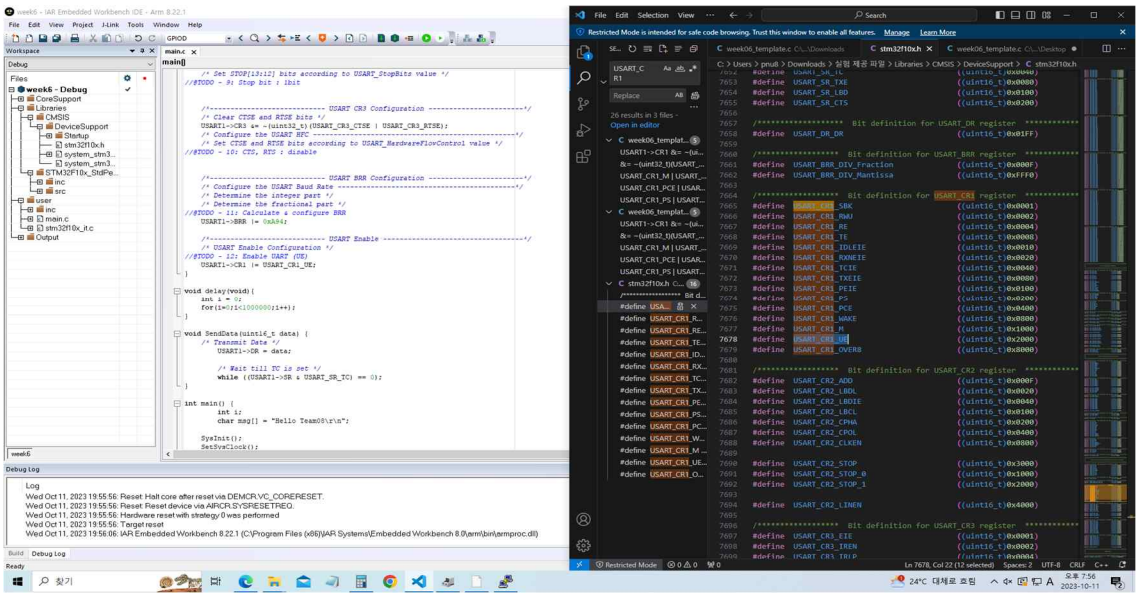
본격적인 Main 함수 작성에 앞서 라이브러리를 이용하여 코드 작성법에 대해 익힌다. 앞선 실험들에서는 아래와 같이 주소와 값을 일일이 입력했다.

```
(*(volatile unsigned int *) 0x40021018) &= ~0x20;
(*(volatile unsigned int *) 0x40021018) |= 0x20;
(*(volatile unsigned int *) 0x40011000) &= ~0x00000F00;
```

```
(*(volatile unsigned int *) 0x40011000) |= 0x00000400;
```

직접 주소로 접근하는 대신 라이브러리를 이용하면 아래와 같이 쉽게 표현할 수 있다.

```
RCC->APB2ENR &= ~(RCC_APB2ENR_IOPDEN);
RCC->APB2ENR |= RCC_APB2ENR_IOPDEN;
GPIOD->CRL &= ~(GPIO_CRL_CNF2 | GPIO_CRL_MODE2);
GPIOD->CRL |= GPIO_CRL_MODE2_0;
```



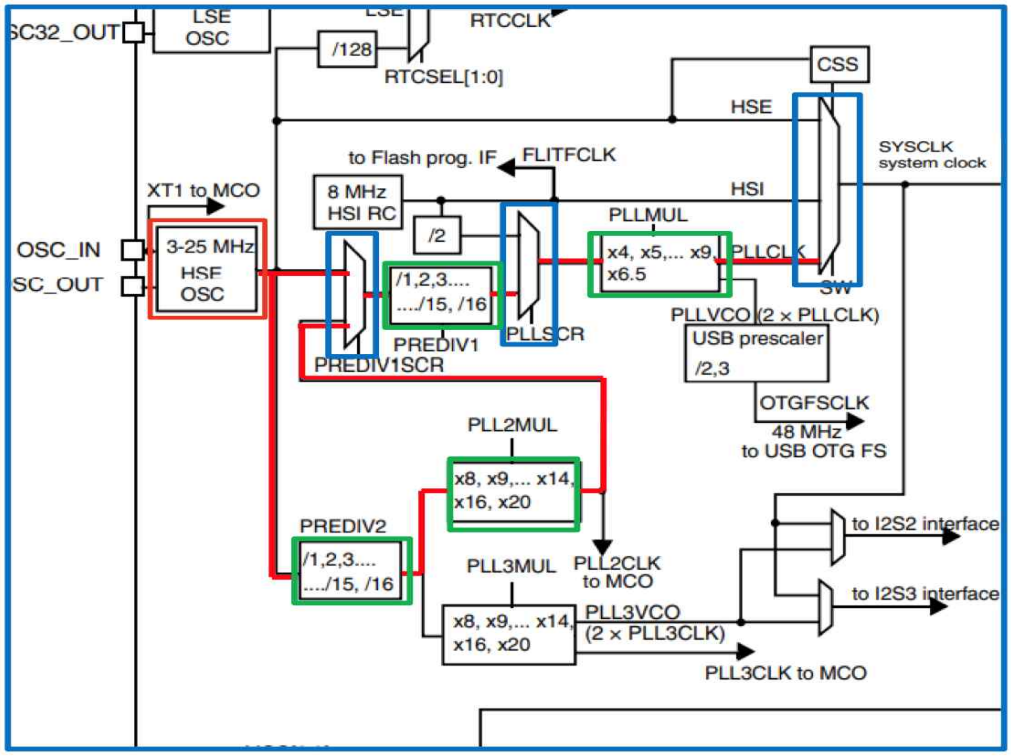
[그림 1] 라이브러리 이용한 코드 작성

[그림 1]은 "stm32f10x.h"에서 라이브러리에 정의된 주소와 값을 찾는 모습이다. 코드에서 라이브러리에 선언된 이름을 어느 정도 입력한 후 ctrl+spaceber 또는 ctrl+alt+spacebar 를 누르면 정의된 이름으로 바로 변경되기도 한다. 이를 이용하면 직접 주소에 접근하는 방식보다 쉽게 코드를 작성할 수 있다.

TODO - 1 Set the clock

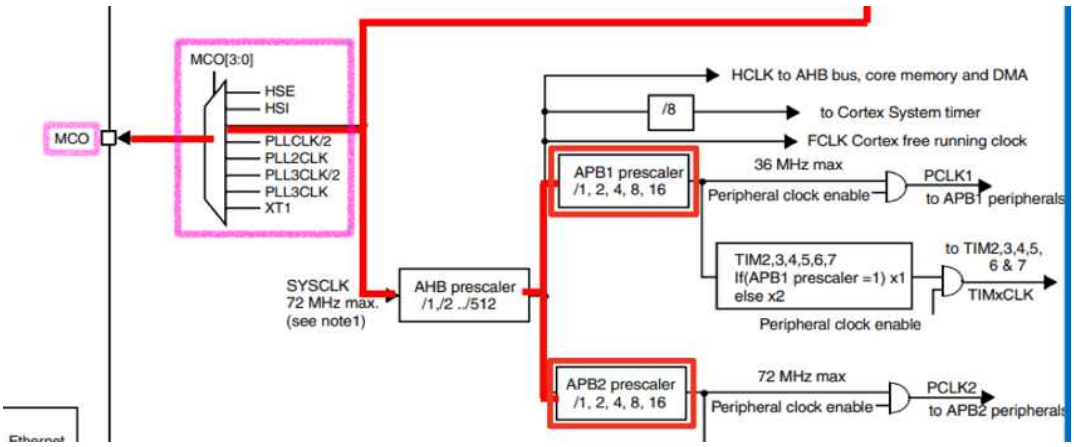
```
//@TODO - 1 Set the clock
      /* HCLK = SYSCLK */
      RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
      /* PCLK2 = HCLK / 2, use PPRE2 */
      RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;
      /* PCLK1 = HCLK */
      RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;

      /* 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);
//@End of TODO - 1
```

[그림 2] TODO1 코드



[그림 3] Clock Tree 구조 1

[그림 3]은 우리가 실험에서 사용하는 보드의 내부 Clock Tree 구조를 나타낸다. 그림에서 볼 수 있듯이 HSE OSC 에서 생성된 25MHz 의 클럭은 바로 시스템 클럭 SYSCLK 로 사용되거나, 주어진 MUX, DIV 그리고 MUL 를 이용해 원하는 값의 위상 동기 신호 PLLCK 를 만들고 이를 SYSCLK 로 사용할 수 있다. **[그림 3]**의 빨간 선을 따라가면 알 수 있듯이 PLLCLK = HSE OSC/PREDIV1 * PLLMUL 또는 PLLCLK = HSE OSC / PREDIV2 * PLL2MUL/PREDIV1*PLLMUL 의 값을 가질 수 있다. 실험에서 구현해야 하는 SYSCLK 값은 52MHz 이므로 25MHz HSE OSC 를 PREDIV2 에서 5 로 나누고, PLL2MUL 에서 13 을 곱하고, PREDIV1 에서 5 로 나누고, PLLMUL 에서 4 를 곱하여 구현할 수 있다.



[그림 4] Clock Tree 구조 2

[그림 4]는 **[그림 3]**에서 이어지는 Clock Tree 구조이다. HIS, HSE, PLL 클럭 중에서 SW MUX 에 의해 시스템 클럭 SYSCLK 으로 설정될 수 있으며, 이는 최대 72MHz 클럭 출력이 가능하다. SYSCLK 는 APB1, APB2 에 전달되며, 전달된 SYSCLK 는 prescaler 에 의해 APB1, APB2 에 각각 최대 36MHz, 72MHZ 의 클럭을 제공한다. 실험에서 구현할 PCLK2 값은 26MHz 이므로 52MHz 로 들어오는 SYSCLK 를 APB2 prescaler 에서 2 로 나누어 구현할 수 있다.

8.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				MCO[3:0]				Res.	OTGFS PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

[그림 5] Clock configuration register (RCC_CFGR)

8.3.12 Clock configuration register2 (RCC_CFGR2)

Address offset: 0x2C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													I2S3SRC	I2S2SRC	PREDIV1SRC
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLL3MUL[3:0]				PLL2MUL[3:0]				PREDIV2[3:0]				PREDIV1[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

[그림 6] Clock configuration register2 (RCC_CFGR2)

따라서, **[그림 5]**와 **[그림 6]**의 레퍼런스를 참고해 TODO1 에 추가한 코드는 아래와 같다.

```
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;
RCC->CFGR |= (uint32_t)( RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL4);
RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL13 |
RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV5);
```

RCC_CFGR_PLLSRC_PREDIV1 는 PLLCLK 로 HIS OSC 클럭이 아닌 PREDIV1 에서 나온 클럭을 택하기 위함이다.

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 |= (uint32_t)RCC_CFGR_MCO_SYSCLK;
//@End of TODO - 2
```

[그림 7] TODO2 코드

[그림 4]에서 볼 수 있듯이 SYSCLK 는 MCO MUX 를 통해서 MCO 로 클럭 출력이 가능하고, 이 출력을 오실로스코프로 파형의 주파수를 측정해 실험에서 요구한 52MHz SYSCLK 가 제대로 구현되었는지 확인할 수 있다. 그러기 위해서 우선 MCO MUX 로 SYSCLK 를 출력하기 위한 설정이 필요하다.

8.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				MCO[3:0]				Res.	OTGFS PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 26:24 **MCO[3:0]**: Microcontroller clock output
Set and cleared by software.
00xx: No clock
0100: System clock (SYSCLK) selected
0101: HSI clock selected
0110: HSE clock selected
0111: PLL clock divided by 2 selected
1000: PLL2 clock selected
1001: PLL3 clock divided by 2 selected
1010: XT1 external 3-25 MHz oscillator clock selected (for Ethernet)
1011: PLL3 clock selected (for Ethernet)

[그림 8] Clock configuration register (Rcc_CFGR)

[그림 8]의 레퍼런스를 참고해 TODO2 에 추가한 코드는 아래와 같다.

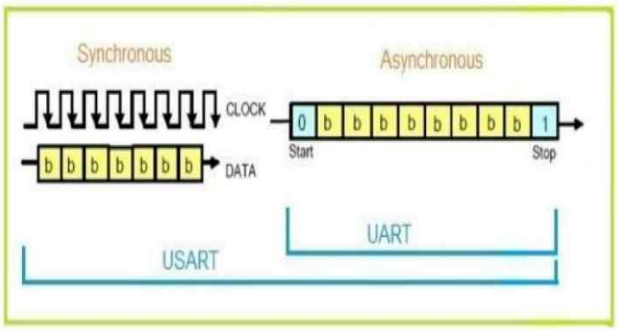
```
RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK;
```

TODO – 3 Rcc setting

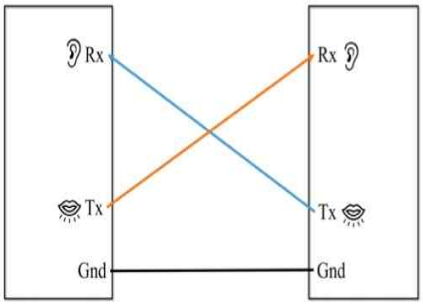
```
void RCC_Enable(void) {
//@TODO - 3 RCC Setting
    /*---------------------------- RCC Configuration -----------------------------*/
    /* GPIO RCC Enable  */
    /* UART Tx, Rx, MCO port */
    RCC->APB2ENR |= (uint32_t)(RCC_APB2ENR_IOPAEN);
    /* USART RCC Enable */
    RCC->APB2ENR |= (uint32_t)(RCC_APB2ENR_USART1EN);
}
```

[그림 9] TODO3 코드

UART 설정을 하기 전에 우선 Serial 통신에 대해 이해해야 한다. Serial 통신이란 하나의 데이터 선을 이용해 여러 비트를 차례로 보내는 통신 방법이다. 여러 개의 데이터 선을 이용해 각각의 선에 비트를 하나씩 보내는 Parallel 통신에 비해 속도는 느리나, 데이터 선을 연장하기 위한 비용이 적다는 장점이 있다.

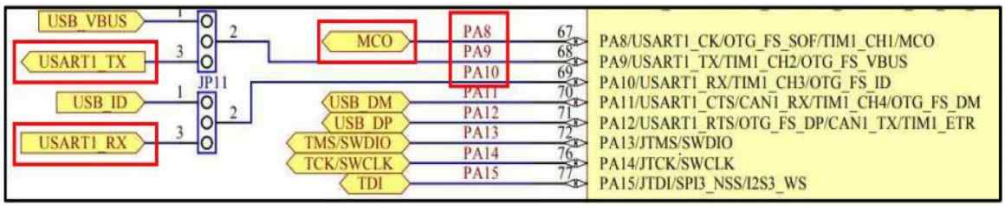


[그림 10] USART 통신과 UART 통신



[그림 11]Rx 와 Tx 교차연결

[그림 10]의 UART 통신은 비동기 통신 프로토콜이다. **[그림 11]**처럼 데이터 수신을 위한 Rx 단자와 데이터 송신을 위한 Tx 단자를 교차연결 해야한다. 클럭이 없기 때문에 양쪽의 Baud Rate 를 일치시켜야 통신이 가능하다. 이때, Baud Rate 란 초당 얼마나 많은 데이터를 전송하는지를 나타내는 값이다. USART 통신은 동기식 통신을 지원하는 UART 통신으로, 별도로 클럭 전송을 위한 클럭선을 사용해 동기화한다. UART 통신과 달리 Start bit 와 Stop bit 가 필요 없다는 장점이 있다.



[그림 12] MCO, UART Tx, Rx pin

[그림 12]에서 볼 수 있듯이 UART 통신을 위해 Tx, Rx pin 을 사용하기 위해서는 USART1 의 클럭을 enable 해줘야 한다. Tx, Rx, Mco pin 은 각각 PA8, PA9, PA10 을 사용하므로 I/O port A 도 클럭을 enable 해줘야 한다.

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

[그림 13] APB2 peripheral clock enable register (RCC_APB2ENR)

[그림 13]의 레퍼런스를 참고해 추가한 코드는 다음과 같다.

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

```
RCC->APB2ENR |= (uint32_t)RCC_APB2ENR_USART1EN;
```

TODO – 4 GPIO Configuration

```
void PortConfiguration(void) {
//@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 |= (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 A CRH - User 4 Button (PA0) */
    GPIOA->CRL &= ~(GPIO_CRL_CNF0 | GPIO_CRL_MODE0);
    /* User S1 Button Configuration */
    GPIOA->CRL |= (GPIO_CRL_CNF0_1);
}
```

[그림 14] TODO4 코드

MCO 인 PA8 과 UART Tx 인 Pa9 는 Alternate function output Push-pull 모드로, UART Rx 인 PA10 은 Input with pull-up/pull-down 모드로 GPIO 설정해야 한다. 버튼을 누르면 문자열을 전송하도록 구현하기 위해서는 사용할 버튼에 대한 GPIO 설정도 필요하다. TODO3 에서 I/O port A 의 클럭을 enable 해주었으므로, port A 에 속하는 PA0 버튼 핀을 사용하기로 했다.

9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

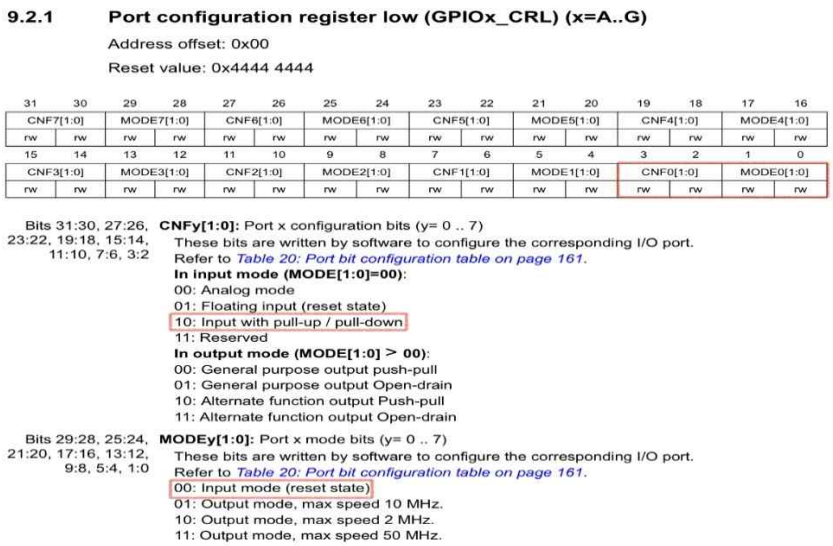
[그림 15] Port configuration register high (GPIOx_CRH) (x=A..G)

```
#define  GPIO_CRH_MODE                    ((uint32_t)0x33333333)

#define  GPIO_CRH_MODE8                   ((uint32_t)0x00000003)
#define  GPIO_CRH_MODE8_0                 ((uint32_t)0x00000001)
#define  GPIO_CRH_MODE8_1                 ((uint32_t)0x00000002)

#define  GPIO_CRH_MODE9                   ((uint32_t)0x00000030)
#define  GPIO_CRH_MODE9_0                 ((uint32_t)0x00000010)
#define  GPIO_CRH_MODE9_1                 ((uint32_t)0x00000020)

#define  GPIO_CRH_MODE10                  ((uint32_t)0x00000300)
#define  GPIO_CRH_MODE10_0                ((uint32_t)0x00000100)
#define  GPIO_CRH_MODE10_1                ((uint32_t)0x00000200)
```

[그림 16] "stm32f10.h" 라이브러리

[그림 15]의 레퍼런스와 **[그림 16]**의 라이브러리를 참고해 추가한 코드는 다음과 같다.

```
GPIOA->CRH |= (GPIO_CRH_CNF8_1 | GPIO_CRH_MODE8);
GPIOA->CRH |= (GPIO_CRH_CNF9_1 | GPIO_CRH_MODE9) | GPIO_CRH_CNF10_1;
```



9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

[그림 17] Port configuration register Low (GPIOx_CRL) (x=A..G)

```
#define  GPIO_CRL_MODE                    ((uint32_t)0x33333333)

#define  GPIO_CRL_MODE0                   ((uint32_t)0x00000003)
#define  GPIO_CRL_MODE0_0                 ((uint32_t)0x00000001)
#define  GPIO_CRL_MODE0_1                 ((uint32_t)0x00000002)
```

[그림 18] "stm32f10.h" 라이브러리

[그림 17]의 레퍼런스와 **[그림 18]**의 라이브러리를 참고해 추가한 코드는 다음과 같다.

```
GPIOA->CRL &= ~(GPIO_CRL_CNF0 | GPIO_CRL_MODE0);
GPIOA->CRL |= (GPIO_CRL_CNF0_1);
```

TODO – 6 WordLength : 8bit, TODO - 7: Parity : None

TODO - 8: Enable Tx and Rx, TODO - 12: Enable UART (UE)

```
void UartInit(void) {
    /*---------------------------- 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_TE | USART_CR1_RE);
    /* Configure the USART Word Length, Parity and mode ----------------------- */
    /* Set the M bits according to USART_WordLength value */
//@TODO - 6: WordLength : 8bit

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

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

[그림 19] TODO6, TODO7, TODO8 코드

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

[그림 20] Control register 1 (USART_CR1)

USART CR1 설정이 필요한 TODO6, 7, 8, 12 는 한번에 해결한다**. [그림 20]**에서 볼 수 있듯이 레지스터의 초기 값은 0X0000 이다. 따라서 1 로 설정이 필요한 bit 에만 값을 주고 아닐 경우 디폴트 값을 그대로 유지하면 된다.

Bit 12 **M**: Word length

This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, n Stop bit

Note: The M bit must not be modified during a data transfer (both transmission and reception)

[그림 21] Control register 1 (USART_CR1) M

[그림 21]에서 볼 수 있듯이 문자열의 길이를 지정하는 12 번 bit M 은 디폴트 값인 0 일 때 8bit 길이로 설정함을 알 수 있다. 따라서 TODO6 에서 추가할 코드는 없다.

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

[그림 22] Control register 1 (USART_CR1) PCE

[그림 22]에서 볼 수 있듯이 페리티 비트를 주는 10 번 bit PCE 는 디폴트 값인 0 일 때 페리티 비트를 disable 함을 알 수 있다. 따라서 TODO7 에도 추가할 코드는 없다. 페리티 비트가 disable 이므로 페리티 비트로 사용할 값을 설정하는 PS bit 는 고려하지 않는다.

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

Note: 1: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.

2: When TE is set there is a 1 bit-time delay before the transmission starts.

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

[그림 23] Control register 1 (USART_CR1) TE, RE

[그림 23]에서 볼 수 있듯이 데이터 수신과 송신을 위해서는 3 번 bit TE 와 2 번 bit RE 에 디폴프 값이 아닌 1 을 줘야함을 알 수 있다. 따라서 TODO8 에 추가한 코드는 다음과 같다.

```
USART1->CR1 |= (uint32_t)(USART_CR1_RE | USART_CR1_TE);
```

```
//@TODO - 12: Enable UART (UE)
    USART1->CR1 |= USART_CR1_UE;
}
```

[그림 24] TODO12 코드

Bit 13 **UE**: USART enable
When this bit is cleared the USART prescalers and outputs are stopped and the end of the current
byte transfer in order to reduce power consumption. This bit is set and cleared by software.
0: USART prescaler and outputs disabled
1: USART enabled

[그림 25] Control register 1 (USART_CR1) UE

[그림 25]에서 볼 수 있듯이 USART 통신을 위해서는 13 번 bit 인 UE 에 1 을 줘야한다. 다음은 TODO12 에 추가한 코드이다.

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

TODO - 9: Stop bit : 1bit

```
    /*---------------------------- 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
```

[그림 26] TODO9 코드

27.6.5 Control register 2 (USART_CR2)

Address offset: 0x10

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
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

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.

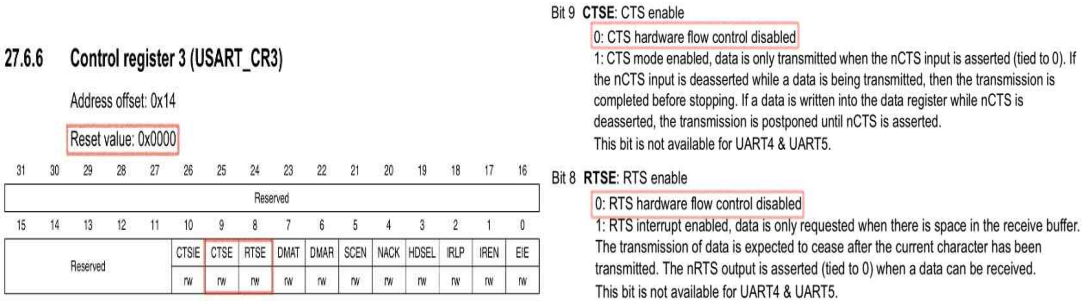
[그림 27] Control register 2 (USART_CR2) STOP

[그림 27]에서 볼 수 있듯이 STOP bit 값을 1 로 설정하기 위해서는 [13:12]번 bit STOP 에 디폴프 값인 00 을 줘야함을 알 수 있다. 따라서 TODO9 에 추가할 코드는 없다.

TODO - 10: CTS, RTS : disable

```
    /*---------------------------- 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
```

[그림 28] TODO10 코드



27.6.6 Control register 3 (USART_CR3)

Address offset: 0x14

Reset value: 0x0000

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 9 **CTSE**: CTS enable
0: CTS hardware flow control disabled
1: CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0). If the nCTS input is deasserted while a data is being transmitted, then the transmission is completed before stopping. If a data is written into the data register while nCTS is deasserted, the transmission is postponed until nCTS is asserted.
This bit is not available for UART4 & UART5.

Bit 8 **RTSE**: RTS enable
0: RTS hardware flow control disabled
1: RTS interrupt enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The nRTS output is asserted (tied to 0) when a data can be received.
This bit is not available for UART4 & UART5.

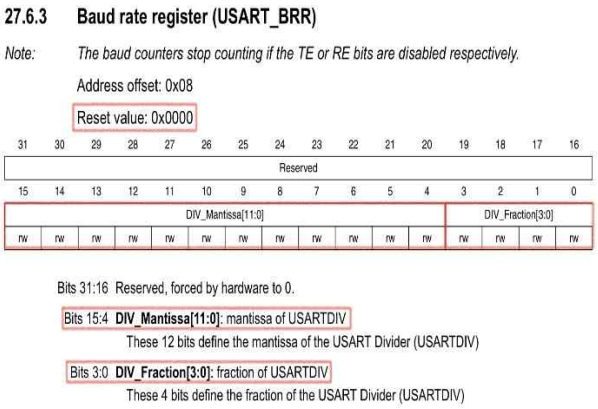
[그림 29] Control register 3 (USART_CR3)

[그림 29]에서 볼 수 있듯이 9 번 pin CTSE 와 8 번 pin RTSE 를 disable 하기 위한 비트는 디폴트 값인 0 으로 TODO10 에 추가할 코드는 없다.

TODO - 11: Calculate & configure BRR

```
    /*---------------------------- USART BRR Configuration -----------------------*/
    /* Configure the USART Baud Rate -----------------------------------------*/
    /* Determine the integer part */
    /* Determine the fractional part */
//@TODO - 11: Calculate & configure BRR
    USART1->BRR |= 0xA94;
```

[그림 30] TODO11 코드



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

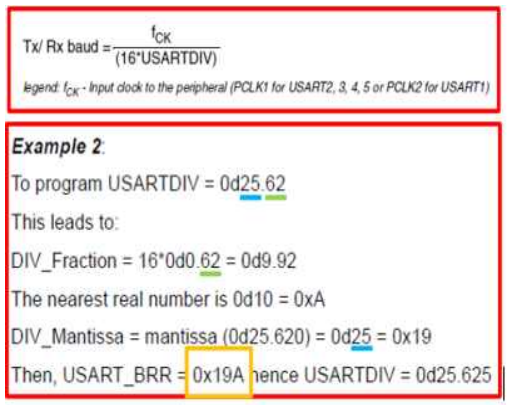
31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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)

[그림 31] Baud rate register (USART_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

[그림 32] USARTDIV 계산

Baud rate 9600 을 사용하기 위해서는 USART_BRR 레지스터에 USARTDIV 값을 주어야 한다. **[그림 32]**에 주어진 계산법을 이용해 구할 수 있다. 이때, fck 값은 PCLK2 값인 26MHz 이다. 따라서 USARTDIV, DIV_Fraction, Div_Mantissa 값은 계산하면 다음과 같다.

$$\text{USARTDIV} = \frac{\text{Fck}}{\text{Baud rate} * 16} = \frac{26\text{M}}{9600 * 16} = 169.27..$$

$$\text{DIV}_{\text{Fraction}} = 16 * (0\text{d}0.27) = 0\text{d}4.32 = 0\text{d}4 = 0\text{x}4$$

$$\text{DIV}_{\text{Mantissa}} = 0\text{d}169 = 0\text{xA}9$$

USART_BRR 레지스터에 주어야 하는 값은 0XA94 이다. 다음은 TODO11 에 추가한 코드이다.

```
USART1->BRR |= 0xA94;
```

TODO - 13: Send the message when button is pressed

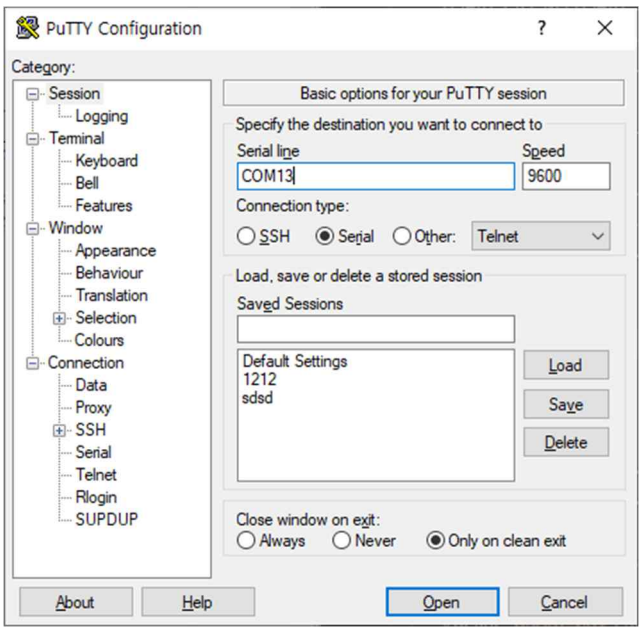
```
while (1) {
    //@TODO - 13: Send the message when button is pressed
  if(!(GPIOA -> IDR & 0x0001)) {
    for (int i=0; i<14; i++) {
      SendData(msg[i]);
    }
    delay();
  }
}
```

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

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

[그림 33] TODO13 코드

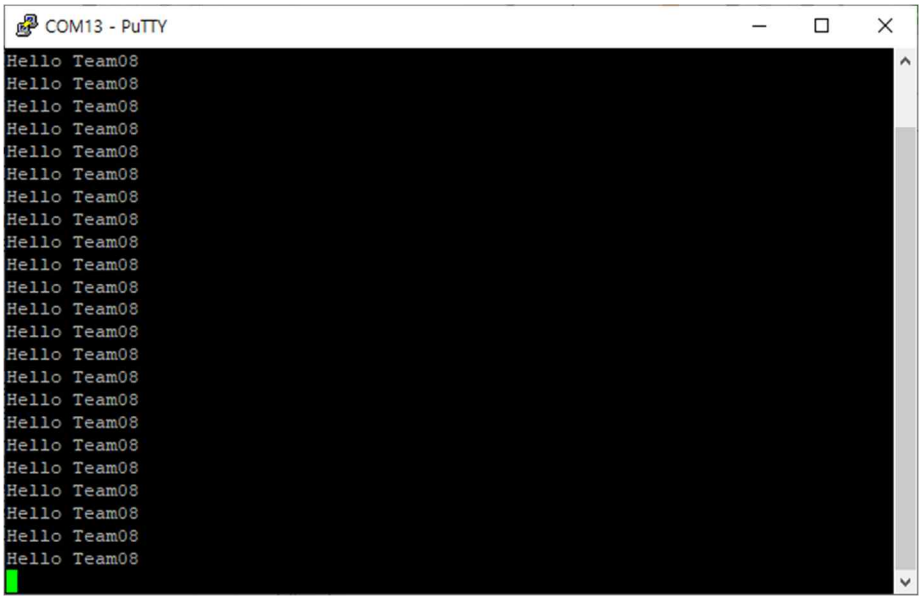
Putty 터미널을 이용해 UART 통신을 위한 main 문의 코드이다. "hello TeamXX" 문자열을 전송하기 위해 버튼이 눌러지면 SendData 반복해 호출하는 것을 확인할 수 있다.

3. 터미널 프로그램 Putty 연결과 UART 통신 결과



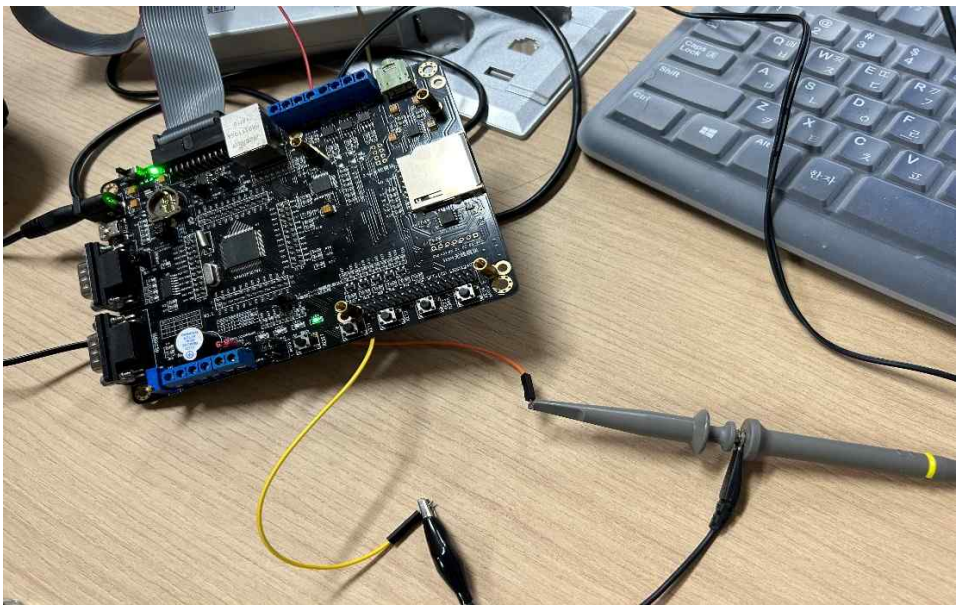
[그림 34] Putty 터미널 연결

UART 통신을 위해서는 실험실 PC 에 설치되어있는 Putty 터미널을 이용한다. Connection type 을 Serial 로 선택한다. PC 장치 관리자에서 보드와 연결된 Serial Port 를 확인하고 Putty 터미널 설정의 Serial line 에 입력한다. 우리조의 경우 COM13 에 연결되어 있었다. Speed 에는 Baud rate 값인 9600 을 입력해 준 후 통신을 시작한다. 이후 버튼을 누르면 누르면 아래의 **[그림 35]**와 같이 동작한다.



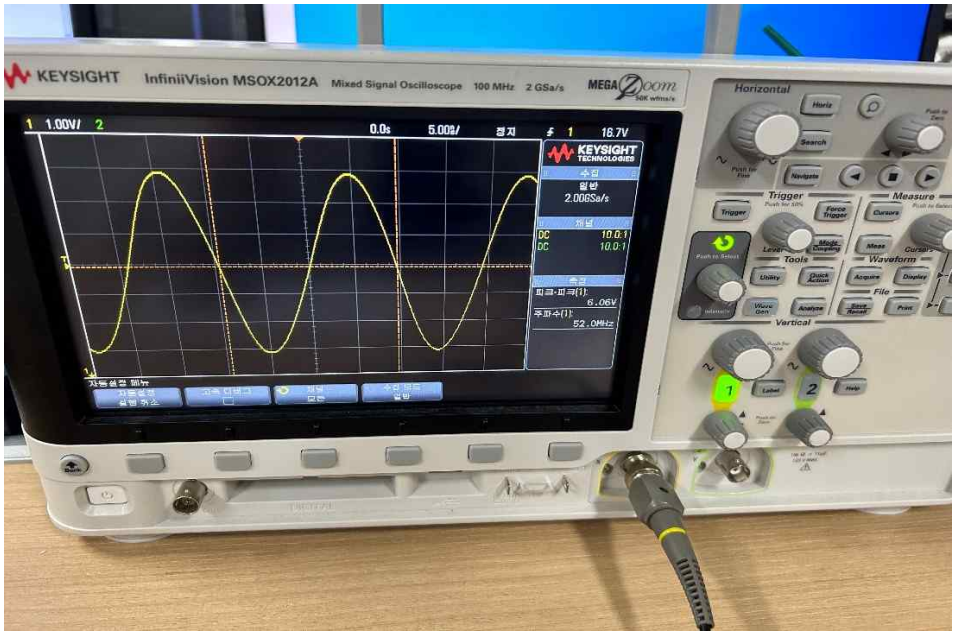
[그림 36] UART 통신 결과

4. 오실로스코프를 이용한 MCO 주파수 측정



[그림 37] 오실로스코프 연결

MCO 로 출력되는 클럭의 주파수 측정을 위해 보드의 PA8 핀과 grond 핀을 점프선으로 연결하여 오실로스코프의 아날로그 측정 핀과 **[그림 37]**과 같이 연결한다. 오실로스코프의 Meas 버튼을 누르고 frequency 모드를 설정하면 화면에 측정 파형과 측정된 frequency 값이 출력된다. Auto scale 버튼을 누르면 좀 더 깔끔한 그래프를 볼 수 있다. 아래의 **[그림 38]**에서 확인할 수 있듯이 MCO 로 52MHz 의 SYSCLK 가 출력된다.



[그림 38] 오실로스코프 측정 결과

4. 결론 및 느낀점

이번 실험에서는 Clock Tree 의 구조와 UART 통신에 대해 이해하는 것을 목표로 하여 사용자 Clock 을 설정하고 실제로 UART 통신을 진행해 보았다. 직접 주소에 접근해 코드를 작성했던 이전 실험들과는 달리 이번 실험에서는 라이브러리에 정의된 변수명을 이용하여 코드를 작성하였다. 라이브러리 변수를 이용한 코드 작성 방식이 처음에는 낯설었지만 제공된 코드 파일과 reference 를 참고하여 이해해가다 보니 금방 적응하였다. 코드 작성을 위해서는 사용자 Clock 설정을 위한 Clock Tree 구조의 MUX, DIV, MUL 의 역할을 이해하는 것이 중요했다. 이를 위해 조원들과 함께 수업 자료를 꼼꼼히 읽으며 TODO1 의 Clock 설정을 해결하니 이후의 TODO 는 순조롭게 해결할 수 있었다. 코드를 작성하고 Putty 터미널을 이용한 UART 통신까지는 문제 없이 진행되었지만, 오실로스코프를 통한 MCO 주파수 측정에서 어려움이 있었다. 기대했던 파형이 출력되지 않아 문제 해결을 위해 연결 핀을 교체하거나 코드를 수정해보는 등 다양한 시도 끝에 정상적으로 실험을 끝마칠 수 있었다.