6주차 결과 보고서

9조 팀장 김주송

팀원 강선호

박형주

손봉국

1. 실험 목적

A. 라이브러리를 활용하여 코드 작성

B. Clock Tree의 이해 및 사용자 Clock 설정

C. 오실로스코프를 이용한 clock 확인

D. UART 통신의 원리를 배우고 실제 설정 방법 파악

2. 배경 지식

가. HIS(High Speed Internal)

STM32에 내장되어 있는 clock으로 8MHz RC 오실레이터에서 생성된다. 생성된 clock은 시스템 클럭으로 사용하거나, PLL clock으로 사용 가능하다. 오차 발생확률이 높은 방식이다.

나. HSE(High Speed External)

STM32 외부에서 입력되는 높은 주파수 clock으로 HSE OSC에서 25 MHz clock을 생성한다. 생성된 clock은 바로 system clock으로 사용하거나, PLL clock으로 사용할 수 있다.

다. Clock Tree

STM32 내부의 Clock 흐름을 보여준다. HIS, HSE, PLL중 SW MUX에 의해 system clock(최대 72MHz)으로 설정될 수 있다. 시스템 클럭은 APB1, APB2에 전달된다. 시스템 클럭은 MCO MUX를 통해서 MCO에 출력해 오실로스코프로 확인이 가능하다.

System Clock을 prescaler를 이용하여 APB1에는 최대 36MHz를 PCLK1에, APB2에는 최대 72MHz를 PCLK2에 제공 가능하다.

A. FCLK

Cortex System(CPU)에서 사용되는 clock

B. HCLK

DMA, Core memory, AHB Bus에서 사용되는 clock

C. PCLK

APB Bus에 사용되는 clock

라. PLL(Phase-Locked Loop)

PLL은 위상 동기 회로이며, 입력 신호와 출력신호를 이용해 출력신호를 제어하

는 시스템을 말한다. 입력된 신호에 맞추어 출력 신호의 주파수 조절이 목적이다.

마. MCO
Clock을 외부에서 측정할 수 있게 하는 기능을 제공한다. MCO핀을 통해 MCO Clock Source를 출력한다.

바. Serial Communication(직렬 통신)
하나의 데이터 선을 이용해 비트를 차례로 보내는 방법이다. 속도는 느리나, 데이터 선을 연장하기 위한 비용이 적다.

사. USART(Universal Sync/Async Receiver/Transmitter)
동기식 통신을 지원하는 UART이다. 데이터 동기화를 위해 같은 클럭을 사용하고 클럭 전송을 위한 별도의 클럭 선이 필요하다. Start bits/Stop bits가 없고 Data bits를 클럭에 동기화해 전송한다.

아. UART(Universal Asynchronous Receiver/Transmitter)
비동기 통신 프로토콜로 Rx(수신)와 Tx(송신)을 교차 연결한다. 비동기 통신이기 때문에 Baud Rate를 일치시켜야 한다.

자. Baud Rate
초당 얼마나 많은 심볼(의미 있는 데이터 묶음)을 전송할 수 있는가를 정한 비율

차. Start bit
통신의 시작을 의미하며 1bit 길이만큼 유지.

카. Data bit
8~9 bits의 데이터를 전송한다. Bit의 개수는 레지스터 설정에 따른다.

타. Parity bit
에러 검사를 위한 값. 수신 측에서 이 bit를 이용해 에러를 검사한다. 에러 발생 여부는 확인이 가능하나, 오류 수정은 불가하다. 수신 측에서 에러 발생 여부 확인 후, 데이터 재요청이 가능하다.

A. Even Parity: 전송하고자 하는 데이터 + parity bit 중 1인 bit의 개수가 짝수

B. Odd Parity: 전송하고자 하는 데이터 + parity bit 중 1인 bit의 개수가 홀수

Even, Odd 설정은 송수신 측이 미리 동기화되어 있어야 한다.

파. Stop bit: 통신의 종료를 알림. 레지스터에 따라 1, 1.5, 2bit으로 설정한다.

3. 실험 내용

지정된 설정

SYSCLK	28MHz
PCLK2	14MHz
aud Rate	28800

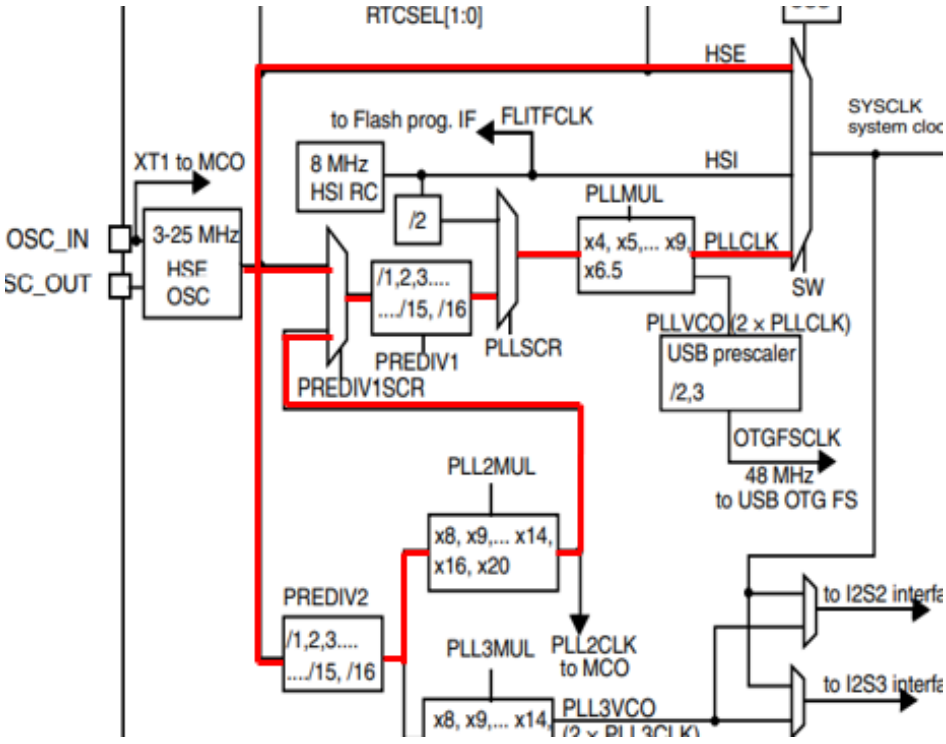
코드 설명

-Todo 1

```
//@TODO - 1 Set the clock, (//)  ?  ? ø     ?  ?         ??                      ?y
        /* HCLK = SYSCLK */
        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->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL | RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
        RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV10 | RCC_CFGR2_PLL2MUL8 | RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV5);
        RCC->CFGR &= (uint32_t)~( RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
        RCC->CFGR |= (uint32_t)( RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL7);
        // 28 = 25 / 10 * 8 / 5 * 7
//@End of TODO - 1
```

```
        /* Select PLL as system clock source */
        RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
        RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
        /* Wait till PLL is used as system clock source */
```



SYSCLK을 28MHz로 만들려고 HSE OSC에서 나오는 25MHz를 /10, *8, /5, *7의 루트를 타게

만들었다.

```
RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL |
RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
RCC->CFGR &= (uint32_t)~( RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
```

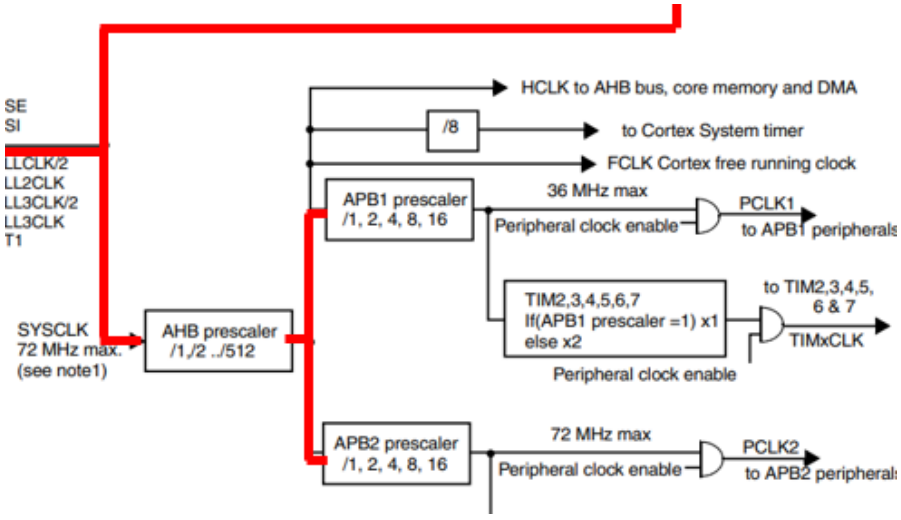
먼저 CFGR, CFGR2에서 PREDIV2, PLL2MUL, PREDIV1SCR, PREDIV1, PLLSCR, PLLMUL을 초기화 한다.

```
RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV10 | RCC_CFGR2_PLL2MUL8 |
RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV5);
RCC->CFGR |= (uint32_t)( RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL7);
```

이후, PREDIV1SCR에서 RCC_CFGR2_PREDIV1SRC_PLL2로 PLL2를 선택하게 하여 PREDIV2, PLL2MUL을 지날 수 있게 한다. PREDIV2는 RCC_CFGR_PREDIV2_DIV10을 이용하고 PLL2MULL은 RCC_CFGR2_PLL2MUL8을 이용하여 /10, *8을 수행한다. PLLSCR에서 RCC_CFGR_PLLSRC_PREDIV1으로 PREDIV1을 선택하게 하고 RCC_CFGR2_PREDIV1_DIV5로 PREDIV1에서 /5를 수행한다. RCC_CFGR_PLLMULL7으로 PLLMUL에서 *7을 수행한다.

```
RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
```

마지막으로 주어진 코드에서 SW MUX를 최기화한 후, RCC_CFGR_SW_PLL로 PLL을 선택하게 한다.



System clock으로 나온 28MHz는 PCLK2에서 14MHz로 나와야 한다.

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

먼저 AHB prescaler에서 28MHz를 그대로 통과시킨다.

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

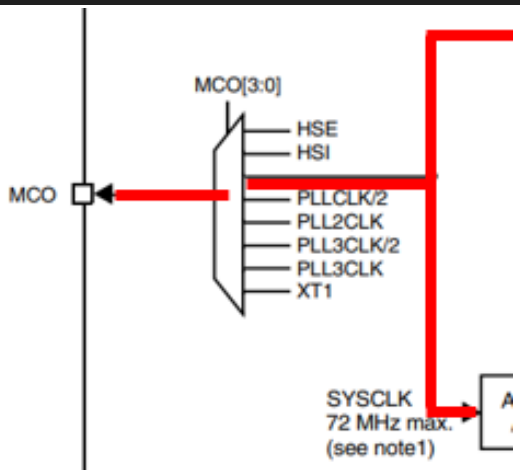
APB2 prescaler에서 /2를 통해 PCLK2에서 14MHz가 나오게 만든다.

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

APB1 prescaler에서는 28MHz를 그대로 내보내어 PCLK1으로 나가게 한다.

-Todo 2

```
91    //@TODO - 2 Set the MCO port for system clock output
92            RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
93            RCC->CFGR |= RCC_CFGR_MCO_SYSCLK;
94    //@End of TODO - 2
95        }
```



MCO MUX에서 SYSCLK을 선택하도록 하여 MCO에서 28MHz가 나오게 한다.

-Todo 3

```
102   void RCC_Enable(void) {
103   //@TODO - 3 RCC Setting
104       /*--------------------------- RCC Configuration ----------------------------*/
105       /* GPIO RCC Enable  */
106       /* UART Tx, Rx, MCO port */
107       RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPCEN;
108       /* USART RCC Enable */
109       RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
110      /* User S1 Button RCC Enable */
111      RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;
112   }
```

F9	39	65	PC8	I/O	FT	PC8	-
E9	40	66	PC9	I/O	FT	PC9	-
D9	41	67	PA8	I/O	FT	PA8	USART1_CK/OTG_FS_SOF / TIM1_CH1[8]/MCO
C9	42	68	PA9	I/O	FT	PA9	USART1_TX[7]/ TIM1_CH2[7]/ OTG_FS_VBUS
D10	43	69	PA10	I/O	FT	PA10	USART1_RX[7]/ TIM1_CH3[7]/OTG_FS_ID
C10	44	70	PA11	I/O	FT	PA11	USART1_CTS / CAN1_RX /

UART 통신과 버튼을 사용하기 위해 A, C포트에 Clock을 넣고, USART를 enable 시켜준다. USART1을 사용하기 위해서 RCC_APB2ENR_USART1EN을 이용한다.

-Todo 4

```
114 void PortConfiguration(void) {
115   //@TODO - 4 GPIO Configuration
116       /* Reset(Clear) Port A CRH - MCO, USART1 TX,RX*/
117       GPIOA->CRH &= ~(
118           (GPIO_CRH_CNF8 | GPIO_CRH_MODE8) |
119           (GPIO_CRH_CNF9 | GPIO_CRH_MODE9) |
120           (GPIO_CRH_CNF10 | GPIO_CRH_MODE10)
121       );
122       /* MCO Pin Configuration */
123       GPIOA->CRH |= GPIO_CRH_MODE8_1 | GPIO_CRH_CNF8_1;
124       /* USART Pin Configuration */
125       GPIOA->CRH |= GPIO_CRH_MODE9_1 | GPIO_CRH_CNF9_1;
126       GPIOA->CRH |= GPIO_CRH_CNF10_1;
127
128       /* Reset(Clear) Port C CRH - User S1 Button */
129       GPIOC->CRH &= ~(GPIO_CRL_CNF4 | GPIO_CRL_MODE4);
130
131       /* User S1 Button Configuration */
132       GPIOC->CRH |= GPIO_CRL_CNF4_1;
133   }
```

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

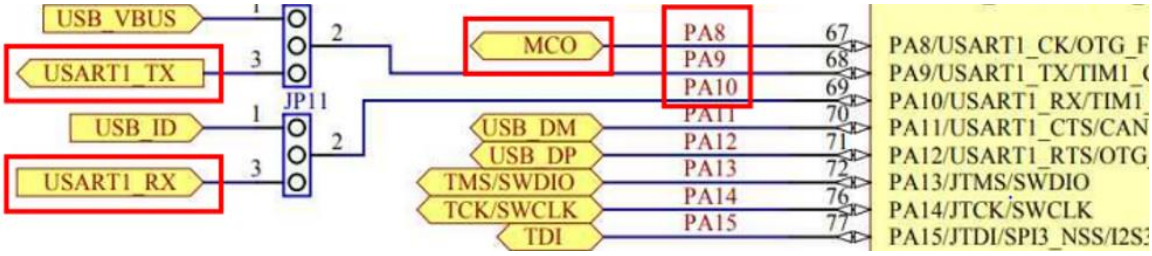
Address offset: 0x04

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.



MCO, Tx, Rx를 설정하기 위해 PA8, 9, 10을 설정하고, 1번 버튼을 이용하기 위해서 PC4를 설정해준다. MCO, Tx는 output모드로, Rx와 버튼은 input pull-up/pull-down모드로 설정한다.

-Todo 6

```
140     /* Set the M bits according to USART_WordLength value */
141   //@TODO - 6: WordLength : 8bit
142     USART1->CR1 |= (uint32_t)(USART_WordLength_8b);
```

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)

Word length를 8bit로 선택한다.

-Todo 7

```
144     /* Set PCE and PS bits according to USART_Parity value */
145   //@TODO - 7: Parity : None
146     USART1->CR1 |= (uint32_t)(USART_Parity_No);
```

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

Parity bit을 사용하지 않도록 설정한다.

-Todo 8

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

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

Tx, Rx를 사용하도록 enable로 설정한다.

-Todo 9

```
154   //@TODO - 9: Stop bit : 1bit
155       /* Clear STOP[13:12] bits */
156       USART1->CR2 &= ~(uint32_t)(USART_CR2_STOP);
157       /* Configure the USART Stop Bits, Clock, CPOL, CPHA and LastBit ------------*/
158       USART1->CR2 &= ~(uint32_t)(USART_CR2_CPHA | USART_CR2_CPOL | USART_CR2_CLKEN);
159       /* Set STOP[13:12] bits according to USART_StopBits value */
```

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.

Bit 11 **CLKEN**: Clock enable
This bit allows the user to enable the CK pin.
0: CK pin disabled
1: CK pin enabled
This bit is not available for UART4 & UART5.

Bit 10 **CPOL**: Clock polarity
This bit allows the user to select the polarity of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship
0: Steady low value on CK pin outside transmission window.
1: Steady high value on CK pin outside transmission window.
This bit is not available for UART4 & UART5.

Bit 9 **CPHA**: Clock phase
This bit allows the user to select the phase of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see figures *289* to *290*)
0: The first clock transition is the first data capture edge.
1: The second clock transition is the first data capture edge.
This bit is not available for UART4 & UART5.

Stop bit을 1bit으로 설정한다.

-Todo 10

```
164       /* Configure the USART HFC -------------------------------------------------*/
165       /* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */
166   //@TODO - 10: CTS, RTS : disable
167       USART1->CR3 |= USART_HardwareFlowControl_None;
```

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.

CTSE와 RTSE를 disable로 설정한다.

-Todo 11

```
169     /*---------------------------- USART BRR Configuration -----------------
170     /* Configure the USART Baud Rate -----------------------------------------
171     /* Determine the integer part */
172     /* Determine the fractional part */
173   //@TODO - 11: Calculate & configure BRR
174     USART1->BRR |= 0x1E6;
```

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)

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

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

Baud rate를 구하여 추가한다.

USARTDIV = 30.38

DIV_Mantissa = 0d30 = 0x1E

DIV_Fraction = 0.38 => 0x6

USART_BRR = 0x1E6

-Todo 12

```
176     /*---------------------------- USART Enable
177     /* USART Enable Configuration */
178   //@TODO - 12: Enable UART (UE)
179     USART1->CR1 |= USART_CR1_UE;
```

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

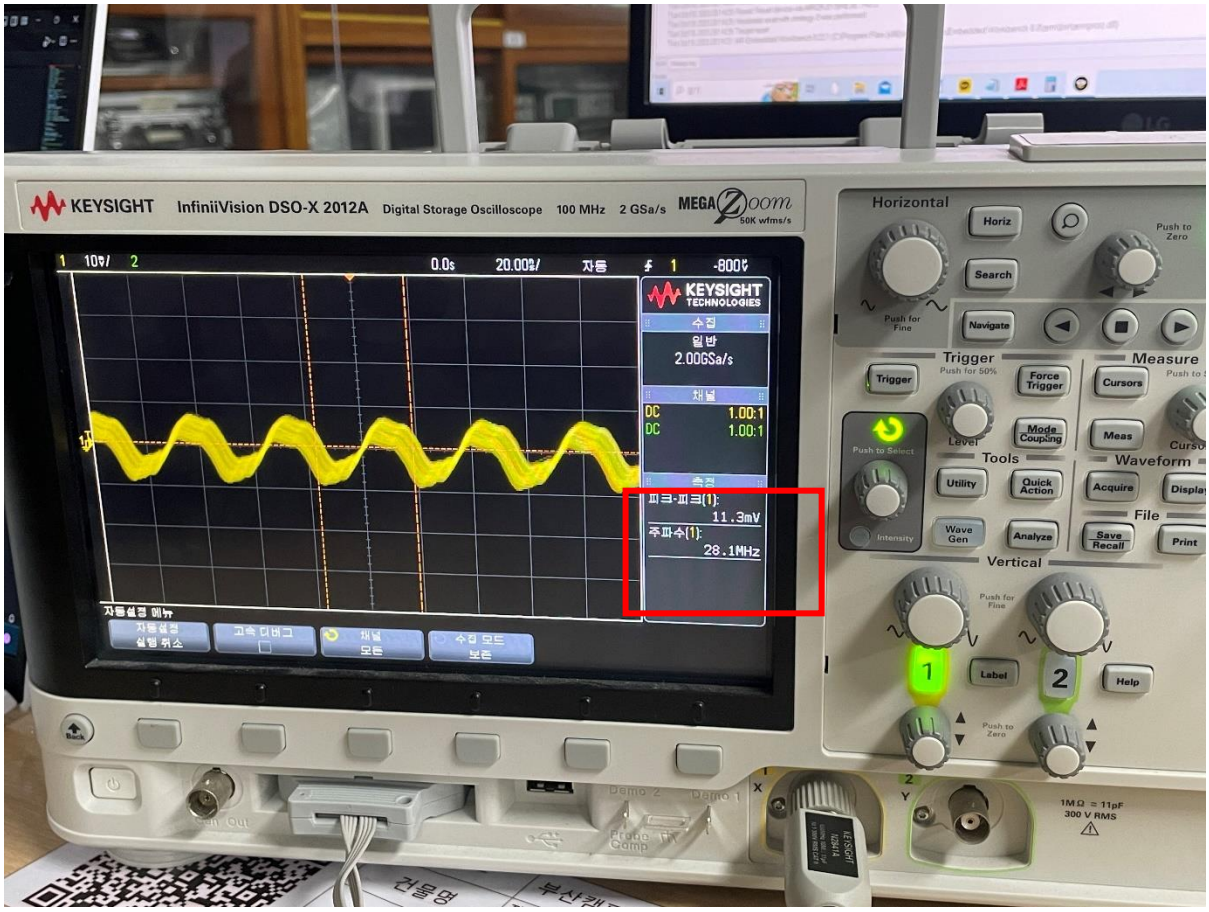
USART enable.

-Todo 13

```
208   while (1) {
209     //@TODO - 13: Send the message when button is pressed
210     if(~GPIOC->IDR & GPIO_IDR_IDR4) {
211         for(i=0; msg[i]!='\n'; i++) {
212             SendData(msg[i]); // \n이 될 때까지 msg출력
213         }
214         SendData('\n'); // \n마저 출력
215     }
216     delay();
217   }
```

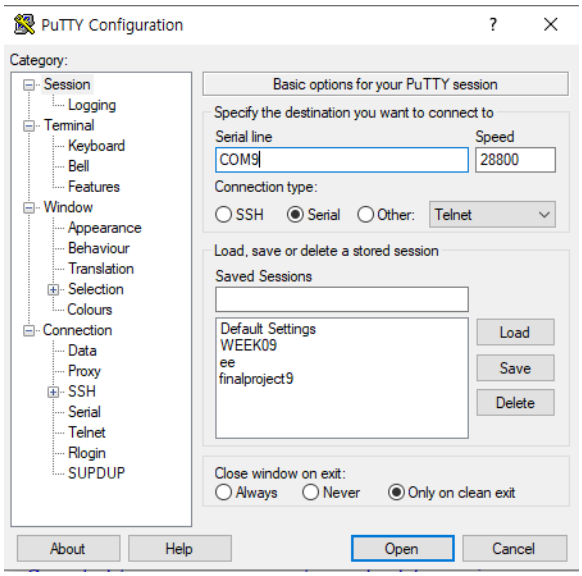
버튼 1번을 감지해서 메시지를 출력하게 만든다.

MCO에서 나오는 System Clock 확인



MCO 출력 단자 PA8과 보드 GND를 오실로스코프에 연결하여 28MHz를 확인했다.

Putty에서 출력확인



4. 실험 결과

이번 실험을 통해서 라이브러리를 활용하여 설정을 직관적으로 실행할 수 있었다. 앞으로 직관적으로 보드를 활용할 수 있게 되었다. 클럭에 대한 이해를 기반으로 커스텀 클럭을 각각 다른 포트에 집어넣고 활용할 수 있도록 클럭 트리를 이해하는 방법을 알게 되었다.

```
/* HCLK = SYSCLK */
        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;
```

이 부분에서 PCLK1,2에 다른 클럭을 집어넣을 수 있다는 개념이 수업시간에는 잘 잡히지 않았었는데 보고서를 쓰면서 이해할 수 있게 되었다.

시리얼 통신을 통해 컴퓨터와 통신을 진행했는데 앞으로 배울 블루투스 통신 등에 도움이 많이 될 것 같다.