임베디드 시스템
실험 및 설계
3주차 실험 보고서

2조 노윤정, 박건우, 이동근, 조영진

1. 실험목표

a) 라이브러리를 활용해 코드를 작성

b) Clock Tree 의 이해 및 사용자 Clock 설정

c) UART 통신의 원리를 배우고 실제 설정 방법 파악

2. 실험미션

a) Template.c 에 Todo 부분 라이브러리를 활용해 채워넣기

b) UART 통신을 이용하여 Putty 등의 시리얼 통신 프로그램으로 출력하기

ex) User S1 Button 누르는 동안, Putty에 “Hello Team02₩r₩n” 출력하기

3. 주의사항

a) 기기 작동중 케이블 뽑기 금지

b) 실험 후 자리 및 PC 깔끔히 뒷정리

4. 실험과정

1) Clcok Configure (Todo 1~2)

- i) Todo 1.

```
56  //@TODO - 1 Set the clock ////////////////////////
57          // -*- Reference Manual 102p
58          /* HCLK = SYSCLK */
59          RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
60          /* PCLK2 = HCLK / 2, use PPRE2 */
61          RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;
62          /* PCLK1 = HCLK */
63          RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;
```

< Figure 1. Todo 1 >

RCC(Reset and Clock Control)의 CFGR(Clock Configuration Register)을 이용해 HCLK, PCLK2, PCLK1의 Clock을 조정한다.

```
1735  #define  RCC_CFGR_HPRE_DIV1
1736  #define  RCC_CFGR_HPRE_DIV2
1737  #define  RCC_CFGR_HPRE_DIV4
1738  #define  RCC_CFGR_HPRE_DIV8
1739  #define  RCC_CFGR_HPRE_DIV16
1740  #define  RCC_CFGR_HPRE_DIV64
1741  #define  RCC_CFGR_HPRE_DIV128
1742  #define  RCC_CFGR_HPRE_DIV256
1743  #define  RCC_CFGR_HPRE_DIV512
```

< Figure 2. stm32f10x,h line 1735>

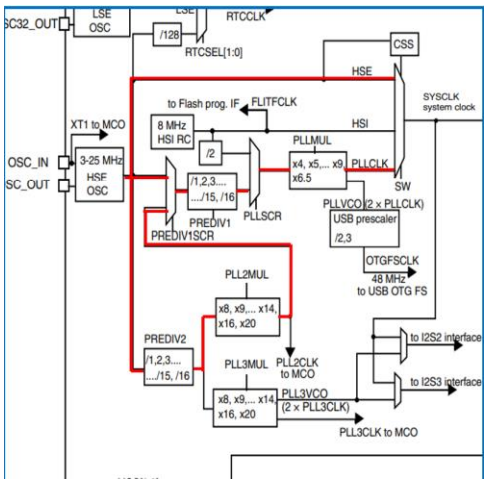
이때 stm32f10x.h 헤더파일을 보면 특정 레지스터의 옵션값을 상수로 선언해 둔 것을 알 수 있다. 이는 이 레지스터 뿐 아니라, 다른 레지스터 또한 마찬가지 이므로 앞으로 있을 코드에서도 이 pre Define 된 상수들을 사용한다면 유용할 것이다.

3주차 강의자료에서 제공된 stm32_ReferenceManual의 140p 부터의 내용을 참고하여 RCC_CFGR의 값을 조정해 주었다. HCLK는 SYS clock 과 일치시키라고 가이드를 주었으니 RCC_CFGR_HPRE _DIV1 (System clock/1)을 선택했다. PCLK2 는 RCC_CFGR_PPRE2_DIV2 (System clock / 2)를 선택하였다. 마찬가지로 PCLK1 또한 RCC_CFGR_PPRE1_DIV1 (System clock / 1)을 선택해 주었다.

```
/* Configure PLLs ------------------------------------------------------*/
// -*- default Clock : 25MHz
// 1. 28 MHz
RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1 |
            RCC_CFGR_PLLMULL7);
            // -*- /1 /1 *7
RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL8 |
            RCC_CFGR2_PREDIV1SRC_PLL2| RCC_CFGR2_PREDIV1_DIV10);
            // -*- /5 *8 /10 = 28MHz
// 2. 52 MHz
RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1 |
            RCC_CFGR_PLLMULL4)
            // -*- /1 /1 *4
RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL13 |
            RCC_CFGR2_PREDIV1SRC_PLL2| RCC_CFGR2_PREDIV1_DIV5);
            // -*- /5 *13 /5 = 52MHz
```

< Figure 3. Configure PLLs >

위의 코드와는 별개로, System clock을 조정해야 한다. 이때 PLL(Phase-Locked Loop)를 이용하는데 PLL의 주 용도는 입력된 신호에 맞추어 출력 Clock 주파수를 조절하는 것이다. 때문에 PLL을 적절히 이용한다면 원하는 Clock 주파수를 얻을 수 있다.



< Figure 4. PLLs Circuit >

System clock 은 기본적으로 HSF OSC에서 생성된 25MHz를 사용하므로, Default Value를 25MHz

로 맞추고, PLL을 조정해야 한다. Clock 은 HSF OSC에서 생성되어 PREDIV2, PLL2MUL, PREDIV1SCR, PREDIV1, PLLSCR, PLLMUL, SW를 거치게 된다. 따라서 이 값에 해당하는 레지스터를 조정하여야 한다. 우선 강의자료 ppt에서 힌트를 준 "28 = 25 /5 *4 /5 *7" 을 변형해 "28 = 25 /5 *8 /10 *7" 로 Clock을 28MHz 로 조정해 주었다.

Clock의 주파수를 조절해주는 방법은 위에서 언급한 순서대로 PREDIV2, PLL2MUL, PREDIV1, PLLMUL의 값을 조정하는 것이다. 따라서 PREDIV2_DIV5, PLL2MUL_MUL8, PREDIV1_DIV10, PLLMUL_MUL7의 상수를 이용해 설정해 주었다. 이때 DIV는 divide(나누기), MUL은 multiply(곱하기)를 뜻한다. DIV나 MUL 뒤에 오는 수는 얼마를 나누고 곱할 것인지를 뜻한다.

28MHz를 만들어준 후, 우리 팀은 ppt에 추가로 적힌 52MHz의 Clock을 만들어 보기로 했다. 본래 System clock 주파수는 하나만 써야 하므로 이는 쓰이지는 않겠지만 추가로 적어본 코드이다. 실험 제공 자료 ppt에 적힌 "52 = 25 /5 *13 /5 *4" 를 이용해 PREDIV2_DIV5, PLL2MUL_MUL13, PREDIV1_DIV5, PLLMUL_MUL4 로 52MHz의 Clock을 구현하였다.

28MHz로 설정하든 52MHz로 설정하든 PREDIV1SCR은 PLL2로 설정해 주어야 하는데 이는 PREDIV1SCR가 Figure 4의 회로에서 HSF OSC에서 나온 25MHz Clock을 사용할지 아니면 PLL 회로들을 거쳐서 조정된 Clock을 사용할지 결정하는 회로이기 때문이다. 따라서 이 값은 공통적으로 PREDIV1SCR_PLL2 로 설정해 주었다.

- ii) Todo 2.

```
110   //@TODO - 2 Set the MCO port for system clock output
111           // -*- Reference Manual 101p
112           RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
113           // -*- reset MCO
114           RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK;
115           // -*- set MCO Clock to System Clock = 28MHz
116   //@End of TODO - 2
```

< Figure 5. Todo 2 >

MCO(Microcontroller Clock Output)을 초기화 시키고, 이를 System clock으로 설정해 주었다. 이때 우리가 설정한 System clock의 주파수는 28MHz 이므로 MCO의 주파수 또한 28MHz가 된다.

2) RCC & GPIO Configure (Todo 3~4)

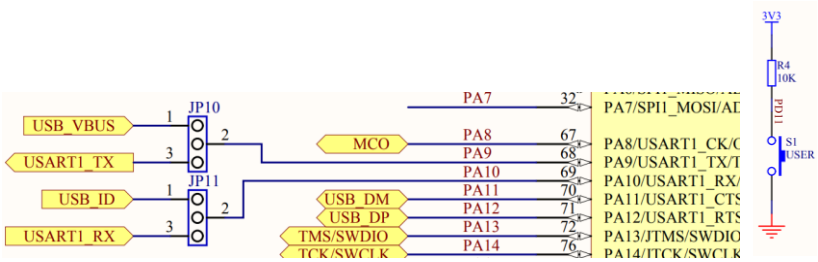
- i) Todo 3.

Clock 설정은 끝났으니, 이제 사용할 포트에 Clock을 인가해야 한다.

```
126 //@TODO - 3 RCC Setting
127     // -*- Reference Manual 112p
128     /*------------------------------ RCC Configuration ------------------------------*/
129     /* GPIO RCC Enable  */
130     /* UART Tx, Rx, MCO port */
131     RCC->APB2ENR |= (uint32_t)(RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPDEN);
132     // -*- Port A : USART & Port D : Switch Enable
133     /* USART RCC Enable */
134     RCC->APB2ENR |= (uint32_t)RCC_APB2ENR_USART1EN;
135     // -*- USART1 Enable
```

< Figure 6. Todo 3 >

Clock 인가는 RCC_APB2ENR를 이용한다. 사용할 포트는 GPIOA(USART) 와 GPIOD(Switch) 이기때문에 각각의 포트를 Enable 시켜 주었다. 또한, USART1을 사용해야 하므로 USART1도 Enable로 설정했다.



< Figure 7. GPIO Port >

- ii) Todo 4.

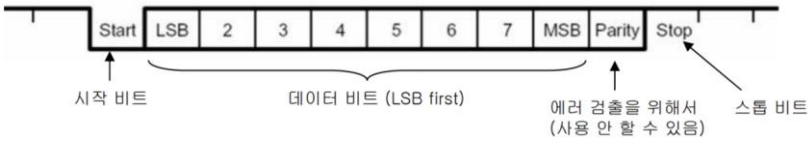
```
139 //@TODO - 4 GPIO Configuration
140     // -*- Reference Manual 172p
141     /* Reset(Clear) Port A CRH - MCO, USART1 TX,RX */
142     GPIOA->CRH &= ~(
143         (GPIO_CRH_CNF8 | GPIO_CRH_MODE8) |
144         (GPIO_CRH_CNF9 | GPIO_CRH_MODE9) |
145         (GPIO_CRH_CNF10 | GPIO_CRH_MODE10)
146     );
147     /* MCO Pin Configuration */
148     GPIOA->CRH |= (uint32_t)(GPIO_CRH_CNF8_1|GPIO_CRH_MODE8);
149                 // -*- MCO : Set Push-Pull, Max Hz = 50MHz
150     /* USART Pin Configuration */
151     GPIOA->CRH |= (uint32_t)(GPIO_CRH_CNF9_1 | GPIO_CRH_MODE9|
152                 // -*- USART1_TX : Set Push-Pull, Max Hz = 50MHz
153                 GPIO_CRH_CNF10_1);
154     GPIOA->CRH &= ~(uint32_t)(GPIO_CRH_MODE10);
155                 // -*- USART1_RX : Set Input, (Pull-up, Pull-down)
156     /* Reset(Clear) Port D CRH - User S1 Button */
157     GPIOD->CRH &= ~(uint32_t)(GPIO_CRH_CNF | GPIO_CRH_MODE); //
158     /* User S1 Button Configuration */
159     GPIOD->CRH |= (uint32_t)(GPIO_CRH_CNF11_1);
160     GPIOD->CRH &= ~(uint32_t)(GPIO_CRH_MODE11);
161                 // -*- Switch : Set Input, (Pull-up, Pull-down)
```

< Figure 8. Todo 4 >

포트에 Clock을 인가한 다음에는 몇 번째 포트를 쓸 건지 설정하는 과정이 필요하다. USART를 사용하기 위해 활성화 해야 할 포트는 GPIOA8(MCO), GPIOA9(USART_TX), GPIOA10(USART_RX) 이므로 GPIOA_CRH에 이들을 각각 설정해주었다.(line 148~154) Switch를 사용하기 위해 활성화 해야

하는 포트는 GPIOD11 이므로 우선 GPIOD를 초기화 시켜준 후, GPIOD11를 설정하였다(line 157~160). (자세한건 Figure 7 내 주석 참고.)

3) UART Configure (Todo 6~12)



< Figure 9. UART data transmission >

- i) Todo 6.

```
169   //@TODO - 6: WordLength : 8bit
170       // -*- ReferenceManual 814p
171       /* Set the M bits according to USART_WordLength value */
172       // USART1->CR1 |= (uint32_t)(USART_CR1_M); // -*- 9bit
173       USART1->CR1 &= ~(uint32_t)(USART_CR1_M); // -*- 8bit
```

< Figure 10. Todo 6 >

USART1_CR1_M을 통해 데이터를 주고받는 단위인 Word의 size를 설정할 수 있다. 가이드에서 8bit를 사용하라 하였기에 8bit로 설정하였다.

- ii) Todo 7.

```
174   //@TODO - 7: Parity : None
175       /* Set PCE and PS bits according to USART_Parity value */
176       USART1->CR1 &= ~(uint32_t)(USART_CR1_PCE | USART_CR1_PS);
```

< Figure 11. Todo 7 >

USART1_CR1_PCE로 Word의 data 끝에 parity bit의 추가 여부를 설정할 수 있다. USART1_CR1_PS는 PCE가 Enable일 때, parity bit를 even parity로 할지, odd parity로 할지 설정하는 값이다.

- iii) Todo 8.

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

< Figure 12. Todo 8 >

USART의 Tx와 Rx를 Enable 시키기 위해 CR1_TE와 CR1_RE를 설정해 주었다.

- iV) Todo 9.

```
186   //@TODO - 9: Stop bit : 1bit
187       // -*- Reference Manual 816p
188       /* Set STOP[13:12] bits according to USART_StopBits value */
189       USART1->CR2 &= ~(uint32_t)(USART_CR2_STOP);
190       // -*- 00 : 1bit, 01 : 0.5bit, 10 : 2bit, 11 : 1.5bit
```

< Figure 13. Todo 9 >

USART1_CR2_STOP은 USART의 Stop bit를 설정하는 값이다. 가이드에서 이를 1bit 로 설정하라 하였으니, Reference를 참고해 값을 설정해 주었다.

- V) Todo 10.

```
196   //@TODO - 10: CTS, RTS : disable
197       // -*- Reference Manual 817p
198       /* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */
199       USART1->CR3 &= ~(uint32_t)(USART_CR3_CTSE | USART_CR3_RTSE);
200       // -*- 0 : Disable, 1 : Enable
```

< Figure 14. Todo 10 >

USART1_CR3_CTSE 와 USART1_CR3_RTSE를 이용해 CTS와 RTS를 disable 시킨다. 둘 다 interrupt와 관련된 값이다.

- Vi) Todo 11.

```
206   //@TODO - 11: Calculate & configure BRR
207       USART1->BRR |= 0X1E4;
208       // -*- Mantissa : 0x1E = 30, Fraction : 0x4 -> 4/16 = 0.25
209       // -*- => 0d30.25
```

< Figure 15. Todo 11 >

USART1_BRR(Baud Rate Register)를 이용해 Baud Rate관련 값을 설정할 수 있다.

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

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

< Figure 16. Baud Rate >

위 수식은 Baud Rate를 계산하는 수식이다. 아래의 범례를 참고해 f_ck의 값을 구하면, 우리는 USART1를 사용하므로 f_ck = PCLK2 = 28/2 MHz = 14MHz (1-i 참고) 이다. 목표 하고자 하는 Baud Rate 는 28800Hz이다. 이를 위 수식에 대입하면 USARTDIV = f_ck / (baud * 16) = 14MHz / (28.8KHz*16) ≈ 30.38 이다.

USARTDIV(=BRR)은 부동소수점을 나타내는 레지스터이고, 상위 4bit가 Fraction, 하위 12bit가 Mantissa이다. 30.38을 Fraction과 Mantissa로 나눈다면 0.38 부분이 Fraction이 되고, 30이 Mantissa가 된다.

Fraction = x/16(x는 BRR의 상위 4bit)로 계산하고, 이 값이 0.38이 되어야 하므로 x/16 = 0.38, x = 0.38 * 16 = 6.08 ≈ 6 = 0x6 이다.

Mantissa = x(x는 BRR의 하위 12bit)로 계산하고, 이 값이 30이 되어야 하므로 x = 30 = 0x1E 이다.

위에서 구한 두 값을 토대로 설정해야 하는 USARTDIV(=BRR) 값은 0x1E6 이다. 이 값을 적용시켰지만 후술할 문제로 인해 실험 결과가 제대로 나오지 않아, 해당 값을 0x1E4로 수정하였다.

- Vii) Todo 12.

```
212   //@TODO - 12: Enable UART (UE)
213       USART1->CR1 |= USART_CR1_UE ;
214       // -*- 0 : Disable, 1: Enable
```

< Figure 17. Todo 12 >

마지막으로, CR1_UE를 이용해 UART를 Enable 시켜주었다.

4) Main Function

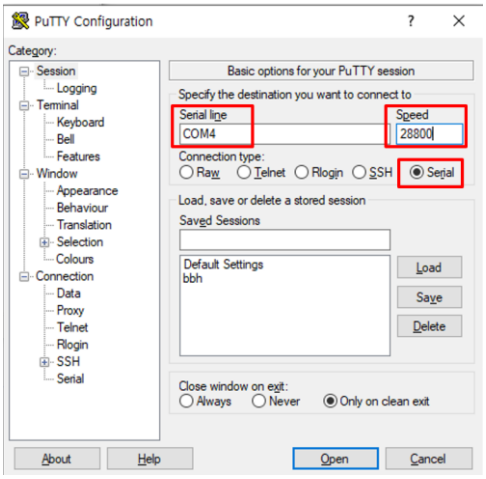
```
239   while (1) {
240     if (!(GPIOD->IDR & GPIO_IDR_IDR11)) { // -*- when swith is pushed
241       for (i=0;i<14; i++) {   // -*- repeat for msg.length
242         SendData(msg[i]);     // -*- Serial Communication
243       }
244     }
245     delay();
246   }
```

< Figure 18. Main Function >

Register 설정 부분을 끝마친 후, 짜여진 코드를 토대로 Main 함수를 작성하는 과정이 필요하다. 우선, 프로그램이 종료되지 않고 반복적으로 신호를 보내게 만들기 위해 while 문으로 가두었다. 240 line의 if 구문에서 GPIOD의 11번 포트(=Switch1)의 입력이 있을 때, 241~243 line의 for 구문을 실행한다. 이 for 문은 보드가 SendData 라는 함수를 “Hello Team02₩r₩n”의 글자 각각마다 시행하게 하는 부분이다. 문자열의 모든 문자를 다 전송한 후, 245 line의 delay 함수를 이용해 메시지의 연속적인 전송을 차단해 오류를 예방한다.

5) Serial Configure

실습 컴퓨터와 실험보드를 연결해 Putty를 실행시켜, Speed 부분을 목표로 했던 28.8KHz로 맞춘 후, 장치관리자의 포트 항목을 참고해 Serial line을 COM11로 설정하고, Connection Type를 Serial로 맞추어 주었다.



< Figure 19. Putty Configure >

5. 실험결과

< Figure 20. Research Result >

보드의 Switch 1을 누르면, “Hello Team02₩n”가 Putty에 정상적으로 연속되게 전송되는 것을 확인할 수 있었다. 스위치에 손을 때면 메시지는 출력되지 않는다.

6. 결론 및 제언

실험이 거의 끝나갈 무렵, Switch를 누르면 Putty에 텍스트가 출력되는 것 까지는 구현에 성공했지만, 알 수 없는 이유로 인해 텍스트 일부가 깨지는 현상이 발생했다. 이를 해결하기 위해서 Parity Bit, Data Length를 수정해보기도 하였지만, 문제는 그대로였다. 혹여나 Main function이 잘못된게 아닐까 하고 “Hello Team02₩r₩n” 대신 “Hello Team02₩r”이나 “Hello Team₩n”으로 바꾸어 보기도 하였지만 문제는 해결되지 않은 채였다.

갖은 우여곡절 끝에 우리 팀은 혹시나 하는 마음에 Baud Rate를 조정하는 레지스터인 USART1_BRR의 값을 0x1E6에서 0x1E5로 바꾸어 보았다. 그랬더니 텍스트의 오류빈도가 낮아지는 모습을 보였다. 그 후, BRR의 값을 0x1E4로 바꾸니 Figure 20의 그림처럼 정상적으로 텍스트가 출력되는 것을 확인할 수 있었다.

Figure 16에서 본 수식에 따르면 BRR을 0x1E4로 설정했을 경우 계산되는 Baud Rate는 14MHz/(16*30.25) ≈ 28925Hz 이기 때문에, 우리가 설정한 28800Hz에서 크게 벗어나게 된다. 그 이유를 조교님께 여쭈어 보니, 보드가 낡아, 기본 클럭을 생성하는 HSF OSC가 불량이 되었거나, 중간중간 거쳐가는 PLL 회로의 불량이 의심된다고 말씀하셨다.

컴퓨터 소프트웨어를 만들 때는, 소프트웨어의 정확성이나 효율만 생각하면 된다. 개인용 컴퓨터는 하드웨어에 문제가 생길 확률이 극히 희박하기 때문이다. 디버그를 시행했을 때 하드웨어의 문제로 판명나는 경우는 흔치 않다. 그런데 Embedded 분야에서는 하드웨어를 동시에 생각해야 한다는 점이 이번 실험을 통해 크게 와 닿았다. 우리 팀은 프로그램의 문제를 소프트웨어에서 찾고자 하였지만 문제는 하드웨어에 있었다. 하지만, 결국엔 소프트웨어로 그 문제를 해결하였다. 이를 통해 또 하나 깨달은 것은 하드웨어에서 생긴 문제점을 단순히 하드웨어를 수리함으로써 해결할 수도 있지만, 소프트웨어의 수정을 이용해 하드웨어의 문제점을 해결 가능하다는 것이었다.

또 하나, Serial Communication에 쓰인 문자열 “Hello Team02₩r₩n”에서, ‘₩r’이 평소 접하지 않았던 문자이기에 팀원들과 검색해본 후, 이에 대해 알게 되었다. ₩r은 문장의 처음으로 Cursor를 옮기는 것이고, ₩n은 해당라인을 끝마치고 새로운 line으로 넘어가는 것이다. 때문에 둘을 합하면 해당 줄을 끝마치고, 새로운 줄로 넘어가는데 새로운 줄의 첫 부분에 커서가 위치하도록 하는 것이다. 이렇게 모르는 부분을 상세히 짚고 넘어가는게 앞으로 있을 실험에도 도움이 될 것이다.