임베디드시스템 설계 및 실험

8조 5주차 실험 결과 보고서

과 목 : *임베디드설계및실험*

조 원 : *박동진*(201824469) *신재환*(201824514) *예진욱*(201824528) *윤건우*(201824532)

분 반 : 002*분반*

실험 목표

1. *라이브러리를 활용하여 코드 작성*

직접 주소로 접근	(*(volatile unsigned int *) 0x40021018) &= ~0x20; (*(volatile unsigned int *) 0x40021018) |= 0x20; (*(volatile unsigned int *) 0x40011000) &= ~0x00000F00; (*(volatile unsigned int *) 0x40011000) |= 0x00000400;
정의된 주소 값 사용	RCC->APB2ENR &= ~(RCC_APB2ENR_IOPDEN); RCC->APB2ENR |= RCC_APB2ENR_IOPDEN; GPIOD->CRL &= ~(GPIO_CRL_CNF2 | GPIO_CRL_MODE2); GPIOD->CRL |= GPIO_CRL_MODE2_0;

2. *Clock Tree의 이해 및 사용자 Clock 설정*

2.1 Clock의 개념

2.1.1 HSI Clock (High-speed internal)

- HSI Clock은 8MHz RC 오실레이터에서 생성됨
- HSI생성된 Clock은 시스템 클럭으로 사용하거나, PLL Clock으로 사용 사능하다.

PLLCK = HSI RC / 2 * PLLMUL

2.1.2 HSE Clock (High-speed external)

- HSE OSC에서 25 MHz 클럭을 생성한다.
- 생성된 클럭은 바로 시스템 클럭으로 사용하거나, PLL Clock으로 사용할 수 있다.

2.1.3 Clock Tree

- HSI, HSE, PLL 중 SW MUX에 의해 시스템 클럭(최대 72MHz)으로 설정될 수 있다.
- 시스템 클럭은 APB1, APB2에 전달된다.
- 시스템 클럭은 MCO MUX를 통해서 MCO에 출력해 오실로스코프로 확인이 가능하다.
- 시스템 클럭은 APB1 prescaler을 이용해 최대 36MHz 클럭을 PCLK1에 제공 가능하다.
- 시스템 클럭은 APB2 prescaler을 이용해 최대 72MHz 클럭을 PCLK2에 제공 가능하다.
- FCLK : Cortex System(CPU)에서 사용되는 Clock
- HCLK : DMA, Core memory, AHB Bus에서 사용되는 Clock
- PCLK : APB Bus에 사용되는 Clock

2.1.4 PLL (Phase-Locked Loop)

- PLL은 위상 동기 회로이며, 입력 신호와 출력신호를 이용해 출력신호를 제어하는 시스템을 말한다.
- 입력된 신호에 맞추어 출력 신호의 주파수 조절이 목적이다.
- 시스템 클럭은 MCO MUX를 통해서 MCO에 출력해 오실로스코프로 확인이 가능하다.

3. *UART 통신의 원리를 배우고 실제 설정 방법 파악*

실험 과정

1. reference 및 datasheet 참고하여 코드 채워 넣기

주소 및 대입 값 정의된 이름으로 입력하기

1.1 TODO. 1

```
//@TODO - 1 Set the clock. (//) ??? ??ø? ????? ??? ???? ???? ????? ?? ?????? ??????ý?
      /* 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_PLLXTPRE | RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
      RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1 | RCC_CFGR_PLLMULL7);

      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_PLL2MUL8 | RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV10);
//@End of TODO - 1
```

: 먼저 Clock Tree의 구성을 설정해주기 위해서는 RCC레지스터의 CFGR과 CFGR2의 값을 설정해주어야 한다. System Clock을 28MHz로 설정해주면 HCLK는 System Clock과 주파수가 같고 PLCK2는 System Clock 주파수의 절반이다. 그래서 AHB prescaler의 값을 /1로 설정하고 APB2 prescaler의 값을 /2로 설정해 주었다. System Clock을 28MHz로 설정하기 위해 RCC_CFGR_PLLSRC_PREDIV1을 통해 PLLSCR에서 PREDIV1을 선택하도록 하고 RCC_CFGR_PLLMULL7를 통해 PLLMUL의 값을 x7로 설정하였다.
RCC_CFGR2_PREDIV2_DIV5를 통해 PREDIV2의 값을 /5로 설정, RCC_CFGR2_PLL2MUL8 통해 PLL2MUL 값을 x8로 설정, RCC_CFGR2_PREDIV1SRC_PLL2를 통해 PREDIC1SCR에서 PLL2 값을 값을 선택하도록 설정, RCC_CFGR_PREDIV1_DIV10을 통해 PREDIV1의 값을 /10로 설정해 주었다.
이렇게 해서 System Clock은 25MHz /5 *8 /10 * 4 = 28MHz로 설정이 된다.

1.2 TODO. 2

```
//@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
```

: MC0 port의 System Clock을 활성화 시켜준다.

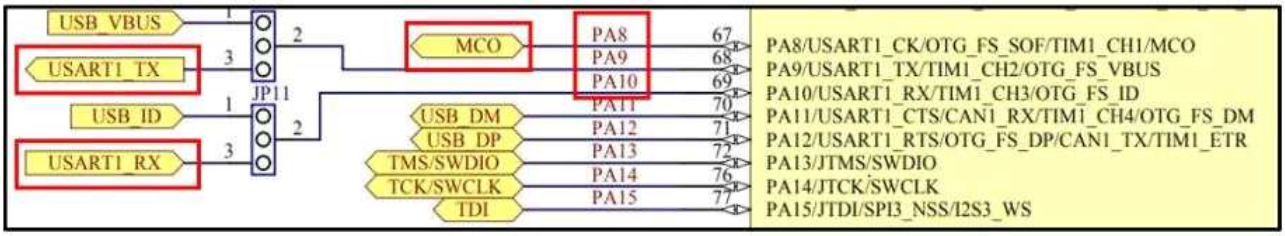
1.3 TODO. 3

```
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 | RCC_APB2ENR_IOPBEN);
    /* USART RCC Enable */
  RCC->APB2ENR |= (uint32_t) RCC_APB2ENR_USART1EN;
    /* User S1 Button RCC Enable */
  RCC->APB2ENR |= (uint32_t) RCC_APB2ENR_IOPDEN;
}
```

: RCC를 enable시켜줘야 하기 때문에 해당하는 port의 System Clock을 활성화 시켜준다.

1.4 TODO. 4

```
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 |= (uint32_t) (GPIO_CRH_MODE8 | GPIO_CRH_CNF8_1);
    /* USART Pin Configuration */
  GPIOA->CRH |= (uint32_t) (GPIO_CRH_MODE9 | GPIO_CRH_CNF9_1 | GPIO_CRH_CNF10_1);

    /* Reset(Clear) Port D CRH - User S1 Button */
  GPIOD->CRH &= ~(GPIO_CRH_CNF11 | GPIO_CRH_MODE11);
    /* User S1 Button Configuration */
  GPIOD->CRH |= (uint32_t) (GPIO_CRH_CNF11_1);
}
```



: MCO는 8번핀을 사용하고 Alternate functuon output Push-pull을 설정해줘야 하기 때문에 MODE8에 CNF8_1로 설정해주고

동일하게 UART TX도 같은 MODE에 CNF값을 가지기 때문에 동일하게 pin만 바꿔주고 넣어준다.

UART RX는 10번핀에 Input with pull-up / pull-down이기 때문에 Input mode는 초기값이므로 MODE는 건드리지 않고 해당하는 CNF값만 넣어준다.

1.5 TODO. 6, 7, 8

```
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);
```

: RE 와 TE bit를 set해주기 위해서 위와 같은 연산을 한다.

1.6 TODO. 9

```
//@TODO - 9: Stop bit : 1bit


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

: CTSE와 RTSE값을 비워주기 위해서 연산을 해준다

1.7 TODO. 10, 11, 12

```
//@TODO - 10: CTS, RTS : disable


    /*---------------------------- USART BRR Configuration -----------------------*/
    /* Configure the USART Baud Rate -------------------------------------------*/
    /* Determine the integer part */
    /* Determine the fractional part */
//@TODO - 11: Calculate & configure BRR
    //board rate = 28800
     USART1->BRR |=  0x1E6;

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

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

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

: $f_{ck} = 14MHz$이고 $USARTDIV$는 $putty$에 넣어주는 속도이다. :
따라서 $T_X/R_X\ baud\ = 30.3819$가나온다.
이때 정수부분 $Mantisa$ 값이 되고 이 값을 16진수로 표현하면 0_X1E이다.
그리고 Fraction 값은 0.3819에서 16을 곱해주면 6.1104가 나오는데 소수점 자리를 버리고 16진수로 바꿔주면 Fraction 값이 나온다. 그러면 Mantisa가 4~15bit이고 Fraction이 0~3비트이기 때문에 0x1E6을 BRR에 넣어주면 된다.

1.8 TODO 13

```
while (1) {
  if((~GPIOD->IDR) & GPIO_IDR_IDR11){
    for(int i=0; i<16; i++){
      SendData(msg[i]);
    }
  }
    //@TODO - 13: Send the message when button is pressed
}
```

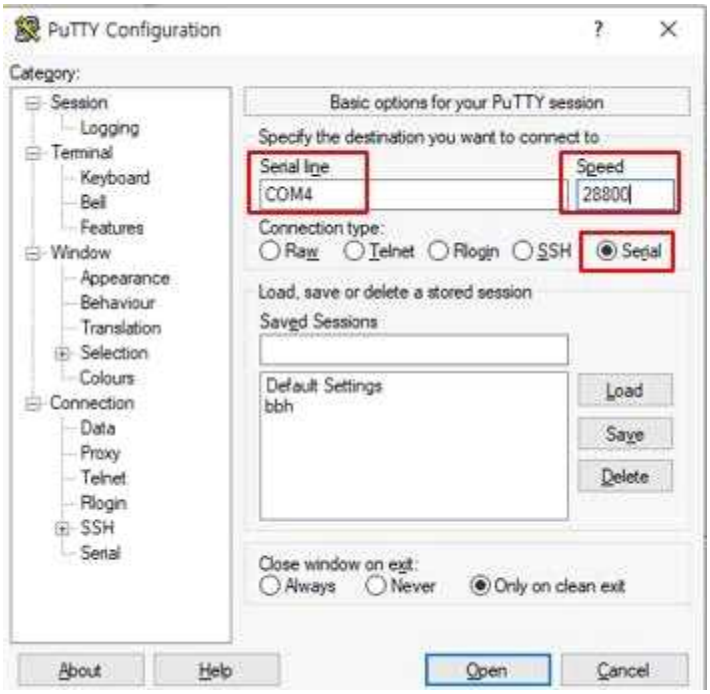
```
void SendData(uint16_t data) {
    /* Transmit Data */
    USART1->DR = data;

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

: user S1 Button이 Dport 11번 pin에 연결되어 있기 때문에 이때 버튼을 누르면 입력 값이 나오기 때문에 IDR을 사용해서 11번 핀에 입력 값이 들어오면 SedData함수를 이용해서 값을 출력하도록 통신한다.

2. Putty를 이용하여 시리얼 통신 확인하기.

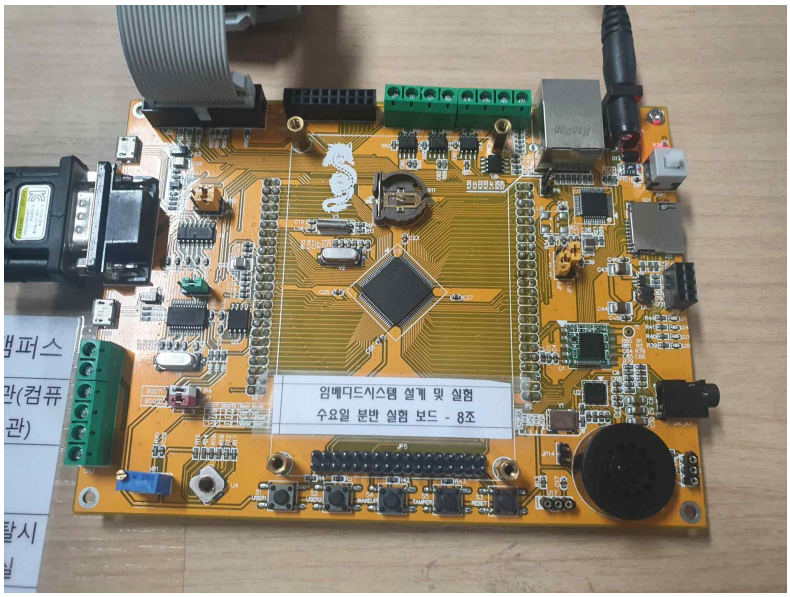
2.1 PC 장치 관리자에서 보드와 연결된 Serial Port 확인

2.2 Connection type – Serial 선택 후 확인한 Port 입력, Baud rate 입력



실험 결과

1. user S1 Button 누르는 동안, "Hello Team8"이 출력된다.



2. 출력 예시

(실험 과정 중 결과 화면을 사진으로 찍지 못해서 ppt에 있는 결과물로 대체)

ref:

1. 임베디드시스템설계및실험 5주차 강의 자료