layout: post author: 한성희, 이호욱, 한경수 date: 2016-10-14 17:48 +0900 tags:

comments: true

개요

비동기(UART) 통신에 대해서 알아보는 실험이다. 다른것으로 USART 가 있는데 이것은 동기화 통신이라고 UART, 즉 비동기까지 지원하는 통신방법이다. 이번 실험에서는 UART 를 이용하여 테스트를 해봤다. UART 통신 동작의 이해는 여러 응용장치를 만드는데, 매우 필요한 기술이다. 비동기 통신 방법이 플래그 체크에 의한 Polling, 인터럽트 방식 등등이 있는데 우리가 실험해본 방식은 Polling 방식이다.

이 과정에서 오실로스코프 파형 관찰을 통한 전자적 신호에 대한 디버깅도 배울 수 있다.

세팅

eclipse

이전의 실험에서 rvc 나 기타 db 는 이미 생성되었다고 가정하고, eclipse 에서 추가적으로 stm32 에 대한 library 를 include 에 넣어놓는다. scatter 를 이용한 방식이 아닌 첫 실험때 사용한 방식으로 flash load 를 시킨다.

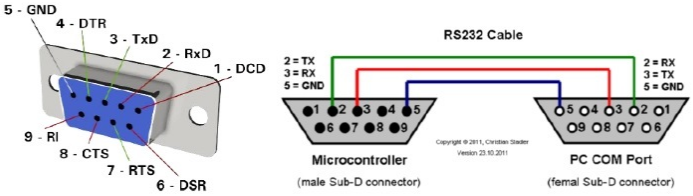
UART 통신 케이블

UART 통신을 위해 보드와 PC 를 케이블로 연결한다. 통신 전 "PL2303*Prolific*DriverInstaller_v130" 을 설치한다.

오실로스코프

먼저 커플링 DC 를 선택한다. 그리고 프로브를 보드 확장 포트의 GND 에 연결하고, 나머지 하나의 연결 고리를 우리가 측정하고자 하는 port 에 접촉시킨다. GND 는 공유되어있기 때문에 확장포트의 아무곳이나 프로브로 집으면 되고, PortA8, PortA9, PorA10 중에 하나에 연결하여 원하는 전압을 측정하면 된다.

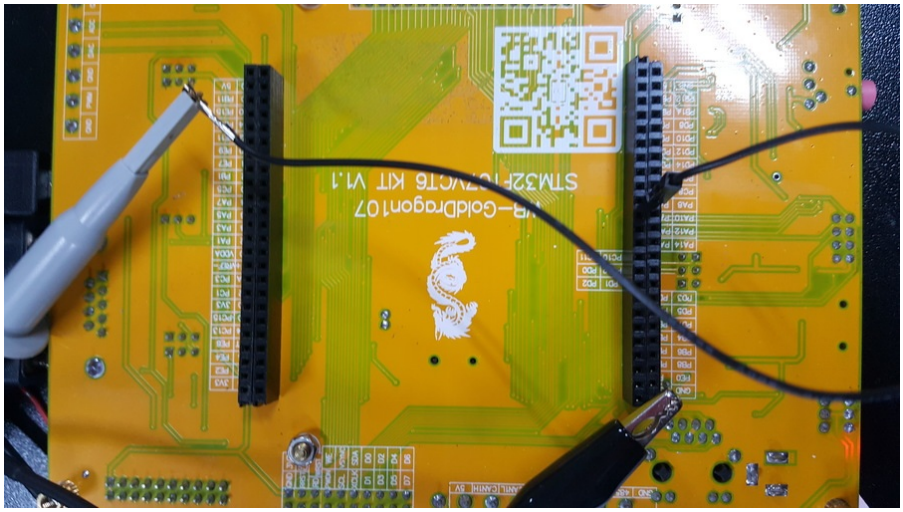
RS-232C



위 그림과 같은 RS-232C 포트는 양방향 통신이 가능하며 각 단말에 TX, RX 가 쌍으로 있다. 그래서 한쪽에서 보내는것도 가능하고, 받는것도 가능하다. (1:1 통신만 가능하다.) 컴퓨터와 기기와 RS-232C 를 연결하면 컴퓨터의 장치관리자에 보면 COM port 가 추가로 잡힌모습을 볼 수 있는데, 만약 com port 의 번호가 우리가 원하지 않는 번호로 잡혀있다면 com port 수정을 해줄 수 있다. 터미널프로그램이 높은 번호의 com port 는 인지 못하는 경향이 있으니 꼭 체크한다.

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 / TIM1_CH4[7]/OTG_FS_DM	-

USART 통신에 맞게 port를 연결하고 모드를 설정한다. MCO, TX, RX 에 해당하는 PA8, PA9, PA10에 연결한다.

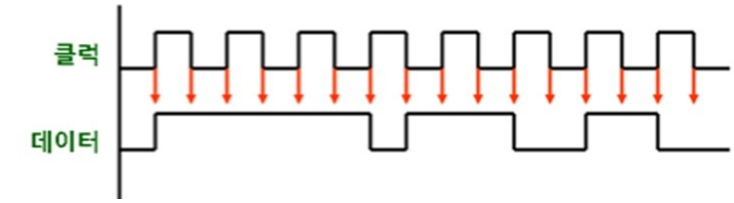


전체적인 기계 연결 모습

기본 개념

동기식 / 비동기식 통신

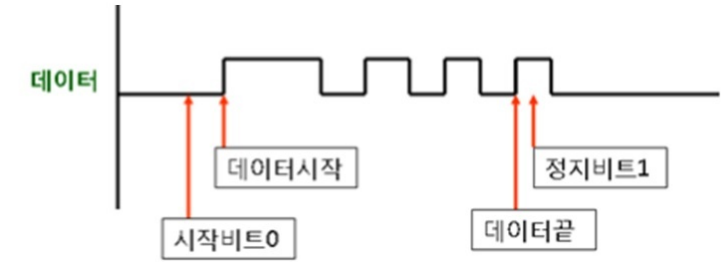
동기식 통신



동기식 통신은 그림과 같이 데이터와는 별도로 송/수신 측이 하나의 기준 클럭으로 동기신호를 맞춰 동작한다. 수신측에서 클럭에 의해 비트를 구별하기 때문에 데이터와 클럭을 위한 2회선을 필요로 한다.

미리 정해진 수 만큼의 문자열을 한 묶음(블록 단위) 으로 하여 동시에 전송할 수 있다. 그래서 동기식은 고속 통신이다.

비동기식 통신



비동기식 통신은 동기식과 달리 클럭과 상관 없이 데이터는 송/수신간 동기를 맞추지 않고 문자 단위로 전송된다. 문자는 데이터 비트 부분과 시작비트(0), 정지비트(1)로 이루어져 전송된다.

한번에 한 문자씩 송/수신할 수 있다. 그래서 비동기식은 저속 통신이다.

시리얼 통신

시리얼 통신

시리얼 통신이란, 데이터를 직렬 형태로 한 bit 씩 전송하는 통신을 말한다.

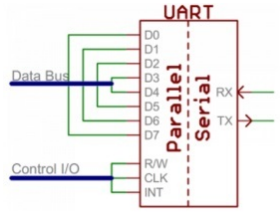
비동기식 시리얼 통신

비동기식 시리얼 통신이란, 데이터가 외부 클럭 신호 도움 없이 몇 가지 규칙에 기반해 동작하는 통신을 말한다.

- Data bits : 전송되는 데이터
- Synchronization bits : 데이터의 시작과 끝, start/stop bit가 이에 해당된다.
- Parity bits : 단순한 error 보정 bit
- Baud rate : 시리얼 라인으로 전송되는 데이터 속도(bps), 보통 115200을 상한선으로 사용한다.

UART 와 USART

UART



UART(Universal Asynchronous Receiver/Transmitter) 는 범용 비동기화 송수신기로

병렬 데이터를 직렬 형식으로 전환하여 데이터를 전송하는 컴퓨터 하드웨어의 일종이다.

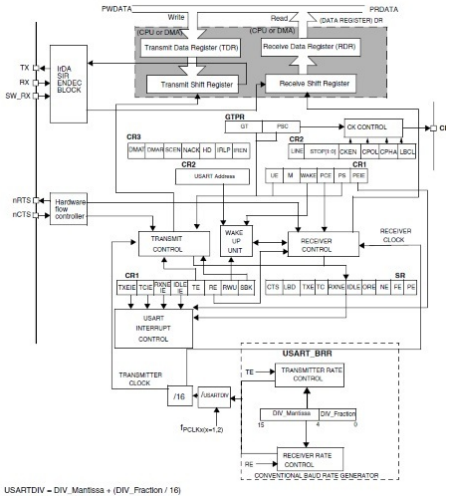
시리얼 기반 통식 방식으로 보통 RS_232 를 통해 통신을 지원한다.

USART

USART(Universal Syn/Asynchronous Receiver/Transmitter) 는 범용 동기화 송수신기로

동기화 통신까지 지원하는 UART이다.

USART 흐름



USARTDIV = DIV_Mantissa + (DIV_Fraction / 16)

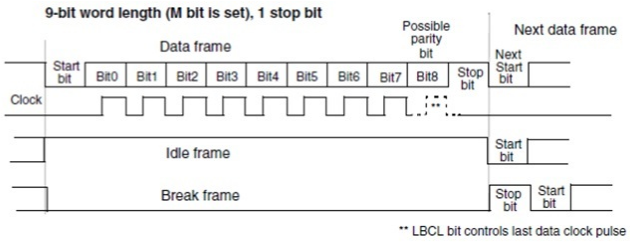
그림에서 CR1, 2, 3 을 이용하여 통신 설정을 변경할 수 있다. Baud rate 를 설정하여 BRR 값을 구한다.

Baud rate 를 구하는 식은 다음과 같다.

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

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

USART 데이터 송/수신간



- Start bits : 통신의 시작을 의미하는 것으로 0으로 설정된다.
- Data Bits : 송/수신되는 데이터를 8-9 bit 로 나타낸다.
- Parity bits : 오류 검증을 위한 값으로 레지스터 설정에 따라 짝/홀/사용안함 으로 선택된다.
- Stop bits : 통신 종료를 의미하는 것으로 레지스터 설정에 따라 비트 수가 나뉜다.

Clock

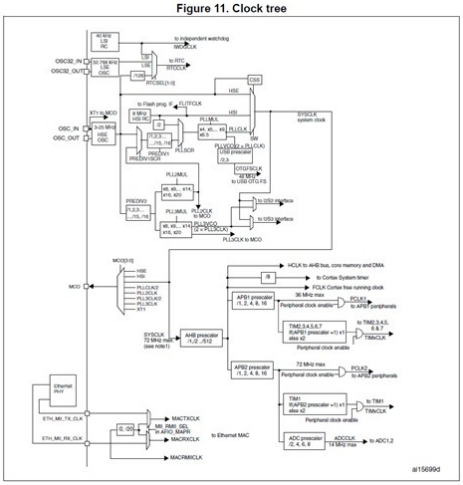
STM32 MCU 의 clock 은 SYSCLK 으로 나타나 있는 system clock 을 기반으로 결정된다.

우리가 사용하는 Cortex-M3 보드에서는 2가지 clock 이 발생된다.

1. HSE clock : Cortex-M3 보드의 Oscillator 에서 방생되는 clock으로 25Mhz 의 값을 가진다.
2. HSI clock : 보드의 내부 클락으로 8Mhz 의 값을 가진다.

Clock Tree

Figure 11. Clock tree



Clock 의 이동 경로를 보여주는 Clock Tree 이다.

Tree를 보면, SYSCLK(System Clock) 은 HSI, HSE, PLL 의 출력 중 하나를 사용한다.

PLL은 HSI 와 HSE 를 곱하거나 나누어서 원하는 주파수 값을 만들 수 있다.

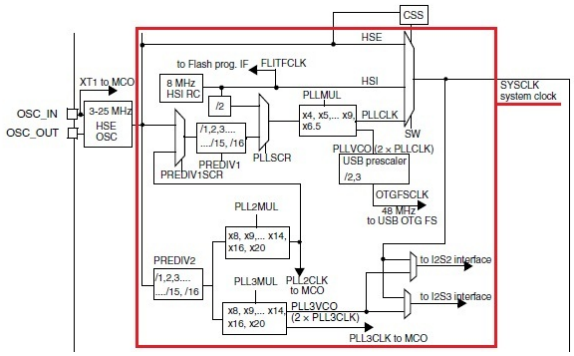
만약 48Mhz 가 필요하다면 기본으로 25Mhz 의 주파수가 HSE OSC 에서 나오게 되는데, 가능한 조합의 수중에 하나가

25Mhz / 5 / 5 * 12 * 4 = 48Mhz 이 된다.

PREDIV2 = /5, PLL2MUL = *12, PREDIV1 = /5, PLLMUL = *4

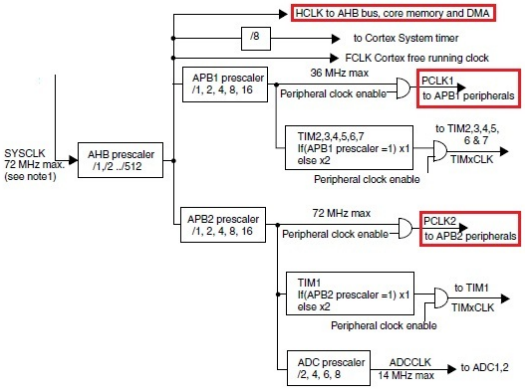
위 박스의 값을 통해 최종적으로 SYSCLK 이 48Mhz 이 된다.

1. System Clock



Clock Tree 중 SYSCLK 값은 Oscillator 에서 나오는 HSE clock 또는 내부 HSI clock 을 PLL로 변환하여 결정할 수 있다.

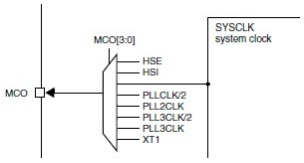
2. Clock 이용



위에서 구한 SYSCLK 값을 이용하여 FCLK, HCLK, PCLK 값을 구할 수 있다.

- FCLK : CPU 에서 사용되는 Clock.
- HCLK : AHB 버스에 사용되는 Clock 으로 memroy controller, interrupt controller 등에 사용된다.
- PCLK : APB 버스에 사용되는 Clock 으로 GPIO, UART, SPI 등에 사용된다.

3. MCO



MCO(Microcontroller Clock Output) 이란, 보드 내부에서 사용되는 clock 을 외부로 출력하는 기능을 담당한다.

Oscilloscope 와 연결하여 clock 설정이 정상적인지 확인할 수 있고 어떤 내부 clock 을 외부로 출력할지 결정한다.

구현

register 초기화

8.3.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

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		PLL3 RDY	PLL3 ON	PLL2 RDY	PLL2 ON	PLLRD Y	PLLON	Reserved				CSSON	HSEBY P	HSERDY	HSEON
		r	rw	r	rw	r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

7.3.2 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 clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					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
ADCPRE[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

RCC*CR 과 RCC* CFGR 을 이용하여 clock 관련 레지스터를 초기화 한다.

RCC_CR 로 HSI 를 enable, HSE 와 PLL 을 OFF 시키는 것으로 초기화.

RCC_CFGR 로 MCO 가 clock을 받지 않고 HSI 가 SYSCLK 가 되도록 초기화한다. 또, 각각의 레지스터가 곱하거나 나누는 동작을 하지 않도록 초기화한다.

```
void SysInit(void) {
  /* Set HSION bit */
  /* Internal Clock Enable */
  RCC->CR |= (uint32_t) 0x00000001;


  /* 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;
}
```

초기화 시킨 코드

Clock 값 설정

이번 실험에서 주어진 Clock 값은 다음과 같다.

```
System Clock : 48MHz
HCLK : 24MHz
PCLK1 : 24MHz
PCKL2 : 12MHz
```

먼저 HSE, HSI 값을 가지고 SYSCLK(System Clock) 값을 48MHz 로 설정해야 한다.

이를 위해서는 HSE = 25MHz 값을 이용하여 아래와 같은 단계로 code 의 Configure PLLs 부분을 수정한다.

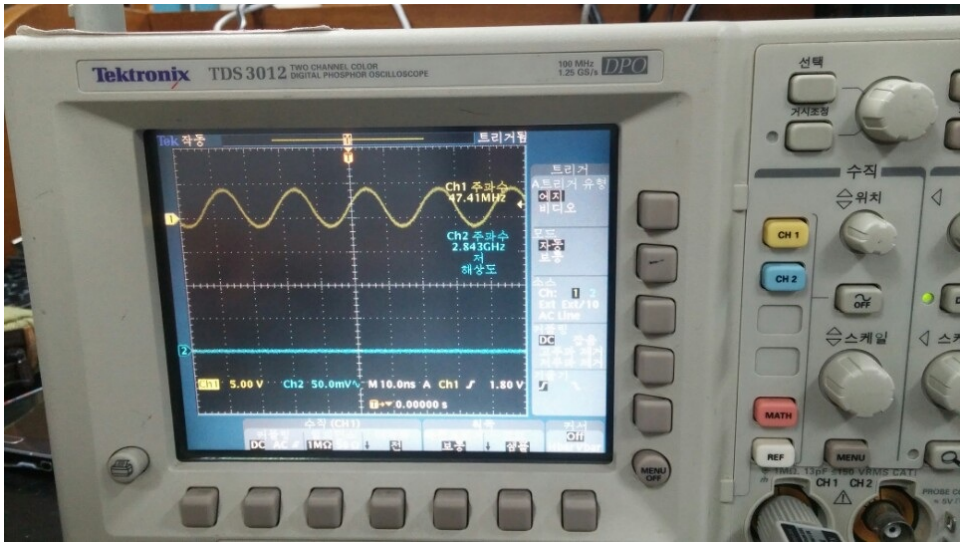
```
1. PREDIV2 에서 /5
2. PLL2MUL 에서 *12
3. PREDIV1 에서 /5
4. PLLMUL 에서 *48
5. SW multiplexer 에서 PLLCLK 값 선택
```

```
/* Configure PLLs ------------------------------------------------------*/
    /* PLL configuration: PLLCLK = ???? */
    // HSE = 25, PREDIV1_div2 : /2 , PLLMULL4: *4

        //HSE, PLL, PULLMUL 사용을 위한 초기화
        RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);

        //아래 코드에서 계산한 값에 *4
    RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1 |
        RCC_CFGR_PLLMULL4);

    /* PLL2 configuration: PLL2CLK = ???? */
    /* PREDIV1 configuration: PREDIV1CLK = ???? */

        //PREDIV2, PLL2MUL, PREDIV1 사용을 위한 초기화
    RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL |
        RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);

        //HSE /5, *12, /5
    RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL12 |
        RCC_CFGR2_PREDIV1SRC_PLL2  | RCC_CFGR2_PREDIV1_DIV5);
```

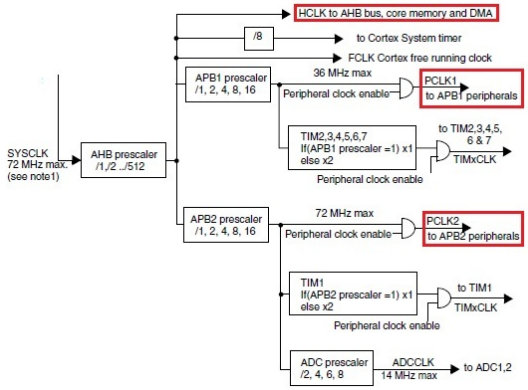
요구 조건에 따라 수정한 코드.

Bits 26:24 **MCO:** Microcontroller clock output
Set and cleared by software.
0xx: No clock
100: System clock (SYSCLK) selected
101: HSI clock selected
110: HSE clock selected
111: PLL clock divided by 2 selected

RCC_CFGR 명령어에서 MCO 관련 명령어를 통해 MCO 값으로 SYSCLK 가 선택되도록 설정하여 오실로스코프로 원하는 클럭 값 SYSCLK = 48MHz 가 나오는지 확인한다.



오실로스코프를 이용하여 SYSCLK = 48MHz 확인.



설정한 SYSCLK 값을 prescalor 를 이용하여 HCLK, PCLK1, PCLK2 값을 생성한다.

SYSCLK 값이 48MHz 임으로 먼저 AHB precalar 를 통해 /2 연산을 해준다.

그리고 PLCK2 를 위해 APB2 prescalar 에서 /2 를 추가로 해준다.

```
/* HCLK = SYSCLK */
RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV2;

/* PCLK2 = HCLK */
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;

/* PCLK1 = HCLK */
RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;
```

수정한 코드는 다음과 같다.

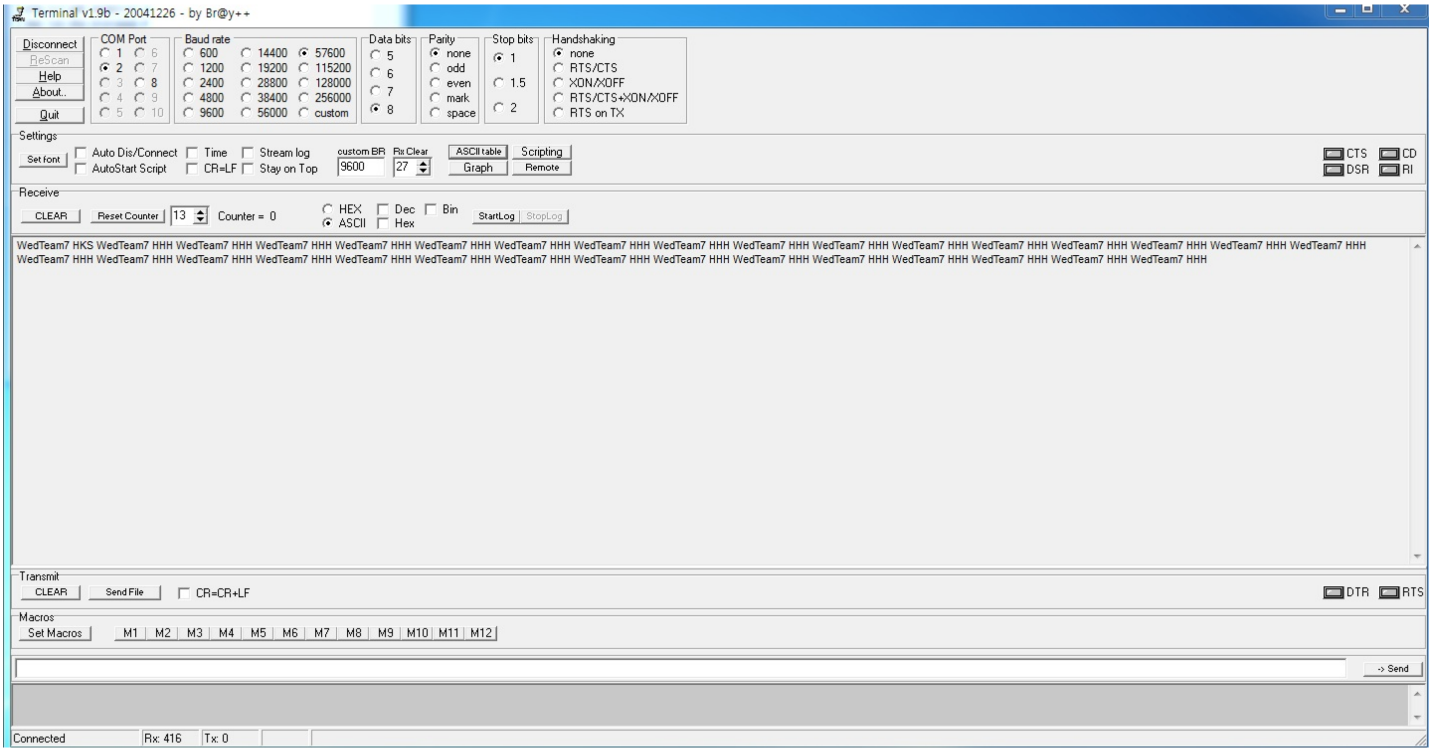
USART 사용

```
void UartInit(void) {
  /*---------------------------- RCC Configuration -----------------------------*/
  /* GPIO RCC Enable  */
  /* USART RCC Enable */
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPCEN | RCC_APB2ENR_IOPDEN;
  RCC->APB2ENR |= RCC_APB2ENR_USART1EN;

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

  /*---------------------------- USART CR2 Configuration -----------------------*/
  /* Clear STOP[13:12] bits */

  /* Configure the USART Stop Bits, Clock, CPOL, CPHA and LastBit ------------*/
  /* Set STOP[13:12] bits according to USART_StopBits value */
  USART1->CR2 &= ~(USART_CR2_STOP | USART_CR2_CPOL | USART_CR2_CPHA);


  /*---------------------------- USART CR1 Configuration -----------------------*/
  /* Clear M, PCE, PS, TE and RE bits */
  USART1->CR1 &= ~(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 */
  /* Set PCE and PS bits according to USART_Parity value */
  /* Set TE and RE bits according to USART_Mode value */
  USART1->CR1 |= (USART_CR1_RE | USART_CR1_TE);


  /*---------------------------- USART CR3 Configuration -----------------------*/
  /* Clear CTSE and RTSE bits */
  USART1->CR3 &= ~(USART_CR3_CTSE | USART_CR3_RTSE);

  /* Configure the USART HFC -------------------------------------------------*/
  /* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */

  /* Write to USART CR3 */

```

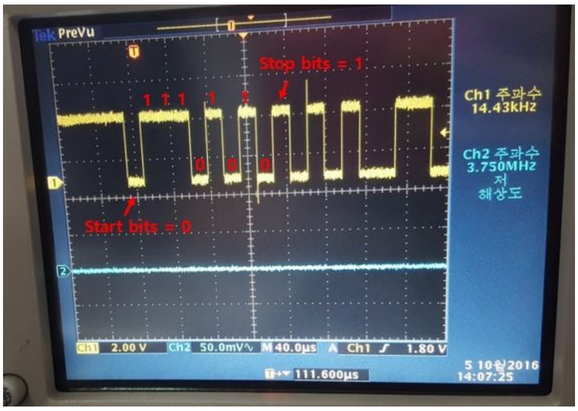
```
  /*---------------------------- USART BRR Configuration ---------------------
-*/
  /* Configure the USART Baud Rate -------------------------------------------*
/
  /* Determine the integer part */
  /* Determine the fractional part */
  USART1->BRR = 0xd0;
  /*---------------------------- USART Enable --------------------------------
-*/
  /* USART Enable Configuration */
  USART1->CR1 |= USART_CR1_UE;
  /*---------------------------- USART DATA output ---------------------------
-*/
  /* USART DATA Transmission */
}
```

실험 결과

terminal 에서 해당하는 Port Number 와 코드에서 설정한 값들을 맞게 옵션을 선택한 뒤 Connect 를 눌러 결과를 확인한다.



terminal 을 통해 확인한 결과.



오실로스코프로 확인한 결과.

문자의 ASCII 값에 맞는 binary 형태의 파형을 얻을 수 있다. 값은 오른쪽에서 왼쪽 방향으로 읽어야 한다.

그림을 보면, 01010111(2) 값을 나타냄으로 이 값은 10진수로 87 즉, "W" 에 해당된다.

결론

System 내부/외부 클럭을 조절하여 원하는 클럭을 생성하고 이를 이용하여 UART(비동기식) 통신을 할 수 있었다. eclipse 를 통해 보드에서 생성된 코드를 terminal 이라는 실행 프로그램으로 확인할 수 있었다. 그리고 오실로스코프로 클럭을 확인하고 보낸 문자에 해당하는 ASCII 값을 binary 형태의 파형으로 볼 수 있어 쉽게 확인할 수 있었다. 이때 Start bit가 0, Stop bit가 1, 왼쪽이 낮은 비트이다. 이번 실험에선 clock tree 를 많이 다루어 tree 구조, 내부의 multiplexer, divider, mco 등의 기능 그리고 클럭의 흐름을 잘 이해할 수 있었다. 또, 새롭게 알게된 UART, USART 의 용도와 차이점을 알 수 있었다.

전체 소스

```
#include "stm32f10x_gpio.h"
#include "stm32f10x.h"
#include "stm32f10x_usart.h"
#include <vector>
#include <string>
using namespace std;
#define STM32F10X_CL

void delay(){
```

```
  for(int i = 0; i < 1000000; i++);
}

void SysInit(void) {
  /* Set HSION bit */
  /* Internal Clock Enable */
  RCC->CR |= (uint32_t) 0x00000001;


  /* 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;
}

void SetSysClock(void)
{
  volatile uint32_t StartUpCounter = 0, HSEStatus = 0;

  /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ---------------------------*/
  /* Enable HSE */
  RCC->CR |= ((uint32_t)RCC_CR_HSEON);

  /* Wait till HSE is ready and if Time out is reached exit */
  do
  {
    HSEStatus = RCC->CR & RCC_CR_HSERDY;
    StartUpCounter++;
  } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

  if ((RCC->CR & RCC_CR_HSERDY) != RESET)
  {
    HSEStatus = (uint32_t)0x01;
  }
  else
  {
    HSEStatus = (uint32_t)0x00;
  }

  if (HSEStatus == (uint32_t)0x01)
  {
    /* Enable Prefetch Buffer */
    FLASH->ACR |= FLASH_ACR_PRFTBE;

    /* Flash 0 wait state */
    FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
    FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_0;

    /* HCLK = SYSCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV2;

    /* PCLK2 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;

    /* PCLK1 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;

    /* Configure PLLs ------------------------------------------------------*/
    /* PLL configuration: PLLCLK = ???? */
    // HSE = 25, PREDIV1_div2 : /2 , PLLMULL4: *4
    RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC | RCC_CFGR_PL
LMULL);
    RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV
1 |
        RCC_CFGR_PLLMULL4);

    /* PLL2 configuration: PLL2CLK = ???? */
    /* PREDIV1 configuration: PREDIV1CLK = ???? */
    // HSE, MUL_0,
    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_PLL2MUL12 |
        RCC_CFGR2_PREDIV1SRC_PLL2  | RCC_CFGR2_PREDIV1_DIV5);

    /* Enable PLL2 */
    RCC->CR |= RCC_CR_PLL2ON;
    /* Wait till PLL2 is ready */
    while((RCC->CR & RCC_CR_PLL2RDY) == 0)
    {
    }

    /* Enable PLL */
    RCC->CR |= RCC_CR_PLLON;

    /* Wait till PLL is ready */
    while((RCC->CR & RCC_CR_PLLRDY) == 0)
    {
    }

    /* Select PLL as system clock source */
    RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
    RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
    RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_2;

    /* Wait till PLL is used as system clock source */
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08)
    {
    }
  }
  else
  { /* If HSE fails to start-up, the application will have wrong clock
       configuration. User can add here some code to deal with this error */
  }
}

void UartInit(void) {
  /*---------------------------- RCC Configuration -----------------------------
--*/
  /* GPIO RCC Enable  */
  /* USART RCC Enable */
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPCEN | RCC_APB2ENR_IOPDEN
;
  RCC->APB2ENR |= RCC_APB2ENR_USART1EN;

  /* USART Pin Configuration */
  GPIOA->CRH = (GPIO_CRH_MODE8 | GPIO_CRH_CNF8_1 | GPIO_CRH_MODE9 | GPIO_CRH_C
NF9_1);

  /*---------------------------- USART CR2 Configuration -----------------------
--*/
  /* Clear STOP[13:12] bits */

  /* Configure the USART Stop Bits, Clock, CPOL, CPHA and LastBit ------------
*/
```

```
  /* Set STOP[13:12] bits according to USART_StopBits value */
  USART1->CR2 &= ~(USART_CR2_STOP | USART_CR2_CPOL | USART_CR2_CPHA);


  /*---------------------------- USART CR1 Configuration ----------------------
--*/
  /* Clear M, PCE, PS, TE and RE bits */
  USART1->CR1 &= ~(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 */
  /* Set PCE and PS bits according to USART_Parity value */
  /* Set TE and RE bits according to USART_Mode value */
  USART1->CR1 |= (USART_CR1_RE | USART_CR1_TE);
  /*---------------------------- USART CR3 Configuration ----------------------
--*/
  /* Clear CTSE and RTSE bits */
  USART1->CR3 &= ~(USART_CR3_CTSE | USART_CR3_RTSE);

  /* Configure the USART HFC -------------------------------------------------
*/
  /* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */

  /* Write to USART CR3 */


  /*---------------------------- USART BRR Configuration ----------------------
--*/
  /* Configure the USART Baud Rate -------------------------------------------
*/
  /* Determine the integer part */
  /* Determine the fractional part */
  USART1->BRR = 0xd0;
  /*---------------------------- USART Enable ---------------------------------
--*/
  /* USART Enable Configuration */
  USART1->CR1 |= USART_CR1_UE;
  /*---------------------------- USART DATA output ----------------------------
--*/
  /* USART DATA Transmission */
}

void SendData(int data){
  while(!(USART1->SR&USART_SR_TXE));
  USART1->DR = data & 0xFF;

  //데이터를 다 보내는 것을 기다린다.
  //        while(!USART1->SR | !USART_SR_TXE));
}



int main() {
  RCC->APB2ENR = (RCC_APB2ENR_IOPAEN | RCC_APB2ENR_AFIOEN); //0x3c;
  GPIOA->CRH = GPIO_CRH_MODE8 | GPIO_CRH_CNF8_1;
  GPIOC->CRH = (GPIO_CRH_MODE8 | GPIO_CRH_MODE9 | GPIO_CRH_MODE10 | GPIO_CRH_M
ODE11);
  GPIOC->IDR = 0x00000000;

  SysInit();
  SetSysClock();
  UartInit();


  /*---------------------------- USART DATA output ----------------------------
--*/
  //        char* names[] = {"hks", "lhw", "hsh"};

  int count = 0;
  while (1) {
    //                if((~GPIOC->IDR & (GPIO_IDR_IDR2) ) ){
    //                        for(int i=0; i<strlen(names[count]); i++){
    //                                SendData(names[count][i]);
    //                        }
    //                        delay();
    //                        count++;
    //
    //                        if(count >= 3){
    //                                count = 0;
    //                        }
    //                }

    if((~GPIOC->IDR & (GPIO_IDR_IDR2)) && count == 0){
      SendData('W');
      SendData('e');
      SendData('d');
      SendData('T');
      SendData('e');
      SendData('a');
      SendData('m');
      SendData('7');
      SendData(' ');
      SendData('H');
      SendData('K');
      SendData('S');
      SendData(' ');

      delay();
      count = (count+1)%3;
    }

    if((~GPIOC->IDR & (GPIO_IDR_IDR2)) && count == 1){
      SendData('W');
      SendData('e');
      SendData('d');
      SendData('T');
      SendData('e');
      SendData('a');
      SendData('m');
      SendData('7');
      SendData(' ');
      SendData('H');
      SendData('H');
      SendData('H');
      SendData(' ');

      delay();
      count = (count+1)%3;
    }

    if((~GPIOC->IDR & (GPIO_IDR_IDR2)) && count == 2){
      GPIOD->BSRR = GPIO_BSRR_BS2;
      SendData('W');
      SendData('e');
      SendData('d');
      SendData('T');
      SendData('e');
      SendData('a');
      SendData('m');
      SendData('7');
      SendData(' ');
      SendData('L');
      SendData('H');
      SendData('W');
      SendData(' ');

      delay();
      count = (count+1)%3;
```

```
    }
  }
}
```