실험 7. USART 통신

전자공학과 21611646 유준상

Ⅰ 실험 목적

➔ [1] 폴링을 이용하여 각 Atmega128의 비동기 통신을 위한 문자 송신, 수신 함수를 작성

[2] printf(), scanf()함수와 문자 송수신 함수를 연결

[3] scanf()함수로 입력 받은 정수를 연산하여 LCD 창에 출력

Ⅱ 실험 도구 및 소자

➔ AVR Studio 4, 브레드 보드, ATmega128 보드, DC 어댑터, PWR B/D, 와이어 스트리퍼, 피복 단선 0.6mm, JTAG 다운로더, LCD, 가변 저항, Molex connector 2개, 클림프 단선(white, black, red) 1개씩

Ⅲ 소스 코드

[수신]

```
#include <avr/io.h>
#include <stdio.h>
#include "lcd.h"
#include "usart.h"
#include <util/delay.h>

int main(void)
{ char lcd_string[2][MAX_LCD_STRING];

 LCD_init(); // LCD 초기화

 USART_init(USART1, 12); // USART1로 통신 선로 지정(보오레이트 : 38400(UBRR=12), 8MHz)
```

```
 sprintf(lcd_string[0], "USART(RECEIVE)");

   LCD_str_write(0, 0, lcd_string[0]);

unsigned char data=0;

while(1){

    data=USART1_receive(); // 송신된 데이터를 수신
    sprintf(lcd_string[1], "Data=%-5d", (int)data); // 수신한 데이터를 형식에 맞춰 저장
    LCD_str_write(1, 0, lcd_string[1]); // LCD에 출력

    }
 return 0;
}
```

[송신]

```
#include <avr/io.h>
#include <stdio.h>
#include "lcd.h"
#include "usart.h"
#include <util/delay.h>

int main(void)
{
    char lcd_string[2][MAX_LCD_STRING];

     LCD_init(); // LCD 초기화

    USART_init(USART1, 12); // USART1로 통신 선로 지정(보오레이트 : 38400(UBRR=12), 8MHz)

    sprintf(lcd_string[0], "USART(TRANSMIT)");

    LCD_str_write(0, 0, lcd_string[0]);
 //LCD에 커서위치 지정을 위해 row=0, col=0, 첫 번째 행의 모든 문자열을 함수를 통해 전달함

unsigned char data = 0;
```

```
    while(1){

    data = data + 1; // 데이터 1씩 증가
     sprintf(lcd_string[1], "Data=%-5d", (int)data); // LCD 아랫줄 첫번째부터 data를 형식에 맞게 메모리 상에 저장
      LCD_str_write(1, 0, lcd_string[1]); // LCD에 출력
    USART1_send(data); // 데이터 송신
    _delay_ms(1000); // 원활한 통신 및 출력을 위해 1초 딜레이 지정

    }

 return 0;
}
```

[lcd.h]

```
#ifndef  __LCD_H__ // if not defined, #define 문장으로 정의하지 않았다면 아래에 등장하는
                   // #endif 문장 이전의 내용들을 include 영역에 포함시키라는 뜻
#define  __LCD_H_
// 디스플레이할 칸 수를 지정 (0x40 : 64칸)
#define  MAX_LCD_STRING          0x40
// 다른 소스 파일에 선언된 함수 사용
extern void        gen_E_strobe(void);
extern void        wait_BusyFlag(void);
extern void        LCD_command(unsigned char data);
extern void        LCD_data_write(unsigned char data);
extern void        LCD_init(void);
extern void        set_cursor(unsigned int row, unsigned int col);
extern void        LCD_str_write(unsigned int row, unsigned int col, char *str);
#endif
```

[lcd.c]

```
#include <avr/io.h>
#include "lcd.h"

#define  RS        PD5      // LCD 문자디스플레이에 연결된 포트D 의 핀번호
#define  RW        PD6      // RW 제어 신호를 PD6로 받기
#define  E         PD7      // Enable 신호를 PD7로 받기
// E 신호를 일정기간 유지시키는 함수
void      gen_E_strobe(void)
```

```
{       volatile  int       i;

        PORTD |= 1<<E;          // E 신호를 High로
        for(i=0; i<10; i++);            // E 스트로브 신호를 일정기간 High로 유지
        PORTD &= ~(1<<E);               // E 신호를 Low로
}
// 다른 명령이 동작 중인지 확인하고 기다리는 함수
void    wait_BusyFlag(void)
{       volatile int                i;
        unsigned char               bf;
// 0번지 읽고 Busy Flag 읽기 위해 입력으로 지정
        DDRC = 0x0;                     // 포트C를 입력핀으로 설정
        PORTD = (PORTD & ~(1<<RS)) | 1<<RW; // RS <- Low, RW <- High
        do{
                PORTD |= 1<<E;          // E 신호를 High로
                for(i=0; i<10; i++);    // E 스트로브 신호를 일정기간 High로 유지
                bf = PINC & 1<<PC7;     // busy flag 읽어 냄
                PORTD &= ~(1<<E);       // E 신호를 Low로
        }while( bf );                   // bf 값이 0이 아니면 busy, 0 일 때까지 반복
}
// 명령어를 수행하는 함수
void    LCD_command(unsigned char data)
{       wait_BusyFlag();        // busy flag가 0될 때까지 대기, 1이면 다음 동작으로 넘어가기
        DDRC = 0xFF;                    // command 날리기 위해 출력모드로 지정
        PORTC = data;                   // 데이터 버스 상에 올리기
        PORTD &= ~(1<<RS | 1<<RW);      // RS <- 0, RW <-0 = 0번지 write
        gen_E_strobe();         // E 스트로브 신호 만들기
}
// 데이터를 쓰기 위한 함수
void    LCD_data_write(unsigned char data)
{       wait_BusyFlag();
        DDRC = 0xFF; // 출력모드로 지정
        PORTC = data; // 데이터 버스 상에 올리기
        PORTD = (PORTD | 1<<RS) & ~(1<<RW); // RS <- 1, RW <-0 = 1번지 write
        gen_E_strobe();
}
// LCD를 쓰기 전 초기화하는 함수
void    LCD_init(void)
{
```

```
        DDRD |= 1<<RS | 1<<RW | 1<<E; // RS(5), RW(6), E(7) 핀을 출력핀으로 설정

        PORTD &= ~(1<<RS | 1<<E | 1<<RW);       // 초기에 RS, E, RW <- 0
// 인터페이스/디스플레이 설정 : DL(1-8bit 인터페이스), N(1-두줄표시), F(1-문자5x10도트)
        LCD_command(0x3C);
// 커서 초기 위치 : 커서 위치를 1행 1열로 움직임
        LCD_command(0x02);
// 화면 클리어 : 스페이스에 해당되는 ASCII 문자 0x20인가
//                 커서 위치를 1행 1열에 움직임
        LCD_command(0x01);
// 문자 입력 모드 : I/D(1), SH(0) – 커서를 오른쪽을 이동하면서 문자 입력
        LCD_command(0x06);
// 디스플레이 ON/OFF 제어 : D(1-디스플레이ON), C(1-커서ON), B(1-커서깜박임ON)
        LCD_command(0x0F);
}
// (row, col)을 받아 해당 위치로 커서 세팅
void    set_cursor(unsigned int row, unsigned int col)
{
        LCD_command(0x80 + (row % 2) * 0x40 + (col % 0x40));
}
// 함수 정의 : row, col 위치에서 문자열 str 을 LCD에 출력시킨다.
void    LCD_str_write(unsigned int row, unsigned int col, char *str)
{
        int     i;

        set_cursor(row, col);
// 문자열(space 포함)이 끝날 때까지 반복
        for(i=0; (i+col < MAX_LCD_STRING) && (str[i] != '₩0'); i++)
                LCD_data_write(str[i]);
}
```

[usart.c]

```
#include <avr/io.h>
#include "usart.h"

// 폴링 USART 초기화
void USART_init(unsigned char ch, unsigned int ubrr_baud)
{
        if( ch == USART0){
```

```
                UCSR0B |= 1<<RXEN0 | 1<<TXEN0; // 송신, 수신 Enable
                UBRR0H = ubrr_baud >> 8;
                UBRR0L = ubrr_baud;
        }else if(ch == USART1){
                UCSR1B |= 1<<RXEN1 | 1<<TXEN1; // 송신, 수신 Enable
                UBRR1H = ubrr_baud >> 8;
                UBRR1L = ubrr_baud;
        }
}

// 폴링에 의한 문자 전송
int USART1_send(char data)
{
        while ( !( UCSR1A & (1<<UDRE1)) );          // UDR 레지스터가 빌 때까지 폴링한다.
        UDR1 = data;                            // UDR 레지스터에 값을 기록한다.
        return data;
}

// 폴링에 의한 문자 수신
int USART1_receive()
{
        while ( !(UCSR1A & (1<<RXC1)) );  // UDR 레지스터에 문자 수신 검사
        return UDR1;
}
```

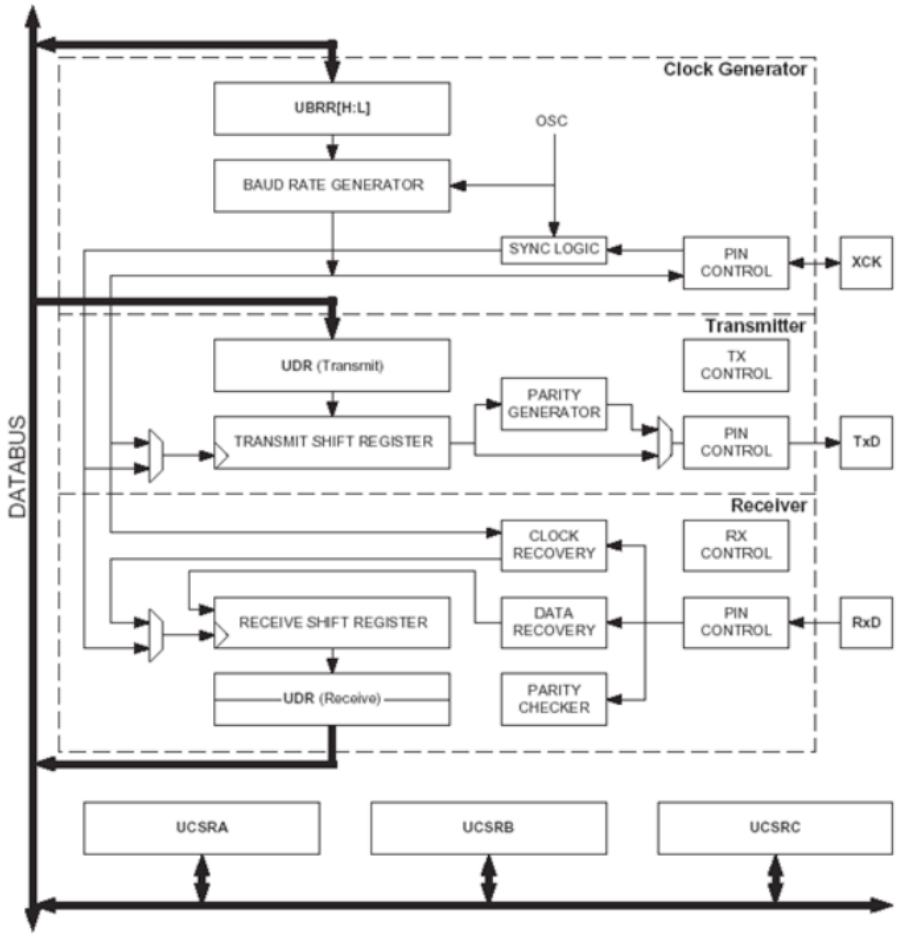
[usart.h]

```
#ifndef  __USART_H__
#define  __USART_H__
#include <stdio.h>

#define  USART0 ((unsigned char)0)
#define  USART1 ((unsigned char)1)

extern   void USART_init(unsigned char ch, unsigned int ubrr_baud);
extern   int USART1_send(char data);
extern   int USART1_receive();

#endif
```

Ⅳ 사전 지식

➔

USART는 Universal Synchronous/Asynchronous Receiver Transmitter의 약자로 동기 또는 비동기의 직렬 통신 방식을 말한다. 여기서 직렬 통신이란 데이터들이 동일한 한 개의 통신선로를 따라 한 비트씩 시간의 흐름에 따라 차례로 보내는 통신 인터페이스이다. 참고로 본 실험에서는 비동기로 사용한다.



위의 회로도를 이용해서 USART의 동작에 대해 간략히 설명한다. 비동기 동작을 하므로 각 내부 동작을 내부 클럭과 bit rate에 맞춰 송신할 데이터를 UDR(USART Data Register)에서 TRANSMIT SHIFT REGISTER로 옮긴다. 다 옮겨지면 TxD를 통해 통신할 상대의 TxD로 송신한다. 수신은 RxD를 통해서 RECEIVE SHIFT REGISTER로 받는다. 받은 데이터를 UDR로 옮기고 수신을 완료한다.

송신, 수신 동작을 해내기 위해서는 여러 제어, 상태 레지스터를 사용해야 한다. 각 동작이 완료됨 또는 준비됨을 알 수 있는 status, flag를 읽어 들여 동작을 진행한다.

USART Register Description

USARTn I/O Data Register – UDRn

Bit	7	6	5	4	3	2	1	0	
	RXBn[7:0]								UDRn (Read)
	TXBn[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

USART Control and Status Register A – UCSRnA

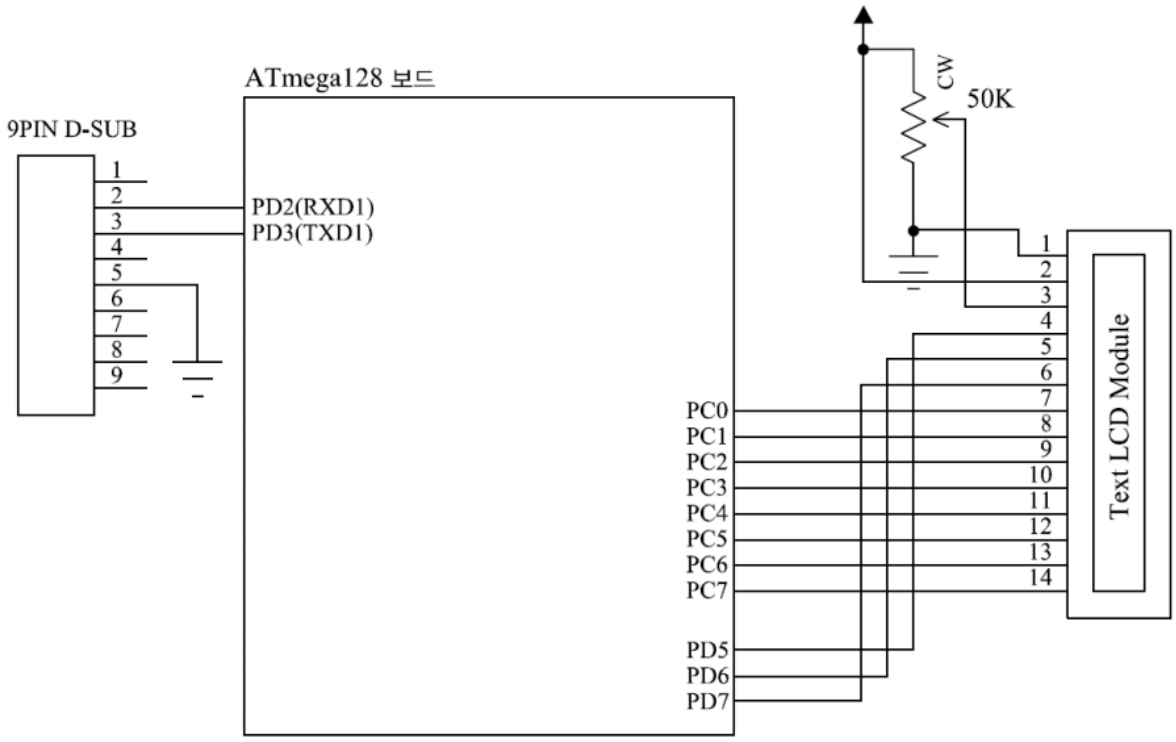
Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- Bit 7 – RXCn: USART Receive Complete
- Bit 6 – TXCn: USART Transmit Complete
- Bit 5 – UDREn: USART Data Register Empty
- Bit 4 – FEn: Frame Error
- Bit 3 – DORn: Data OverRun
- Bit 2 – UPEn: Parity Error
- Bit 1 – U2Xn: Double the USART Transmission Speed
- Bit 0 – MPCMn: Multi-Processor Communication Mode

USARTn Control and Status Register B – UCSRnB

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – RXCIEn: RX Complete Interrupt Enable
- Bit 6 – TXCIE: TX Complete Interrupt Enable
- Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable
- Bit 4 – RXENn: Receiver Enable
- Bit 3 – TXENn: Transmitter Enable
- Bit 2 – UCSZn2: Character Size
- Bit 1 – RXB8n: Receive Data Bit 8
- Bit 0 – TXB8n: Transmit Data Bit 8

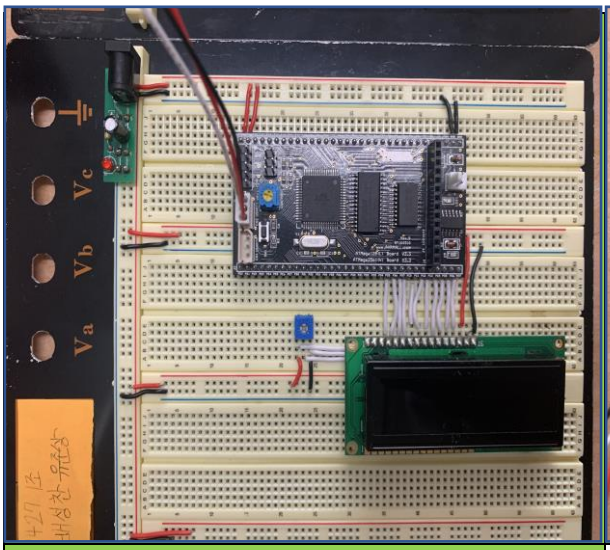


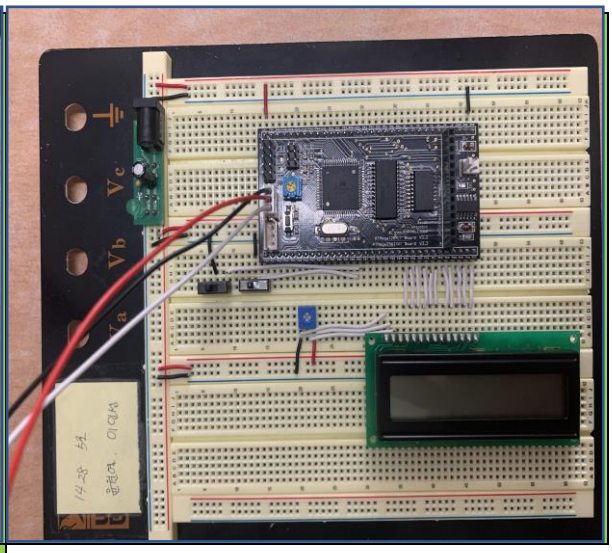
본 실험에서 사용한 회로는 위와 같은 구성이다. 이전 실험과 같은 구성으로 보드와 LCD, 가변 저항을 연결한다. 추가적으로 이전의 실험 후 LCD 고장으로 새롭게 교체하여서 위 회로에는 표시되어 있지 않지만 15, 16에 각각 Vcc와 GND 연결을 통해 백라이트를 사용하여 LCD를 밝게 디스플레이한다. 그리고 9PIN D-SUB는 이번 실험에서 추가로 사용한 것인데 보드에 포함된 연결 핀을 통해 통신할 상대의 보드와 클림프 선을 이용하여 연결 후 직렬통신을 한다.

V 회로도 및 7-segment 작동

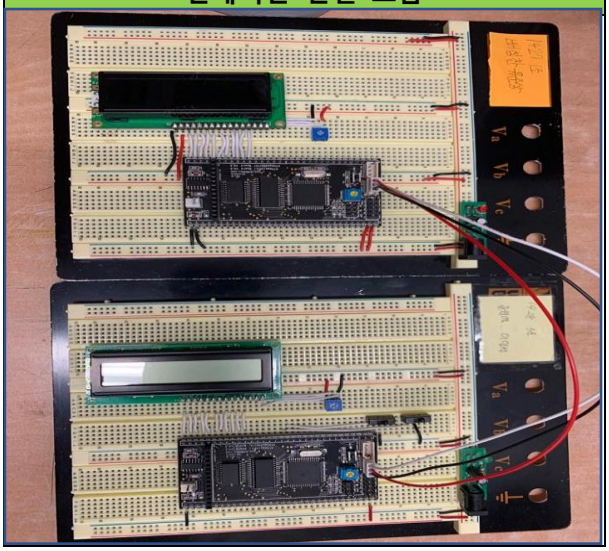
➔ 위의 회로와 동일하게 연결하여 회로를 구성하였다. 본인의 조를 A, 통신할 상대 조의 보드를 B로 나타내었다. 두 보드는 클림프 선 연결을 통해 통신한다.

기본 회로도(A)	기본 회로도(B)



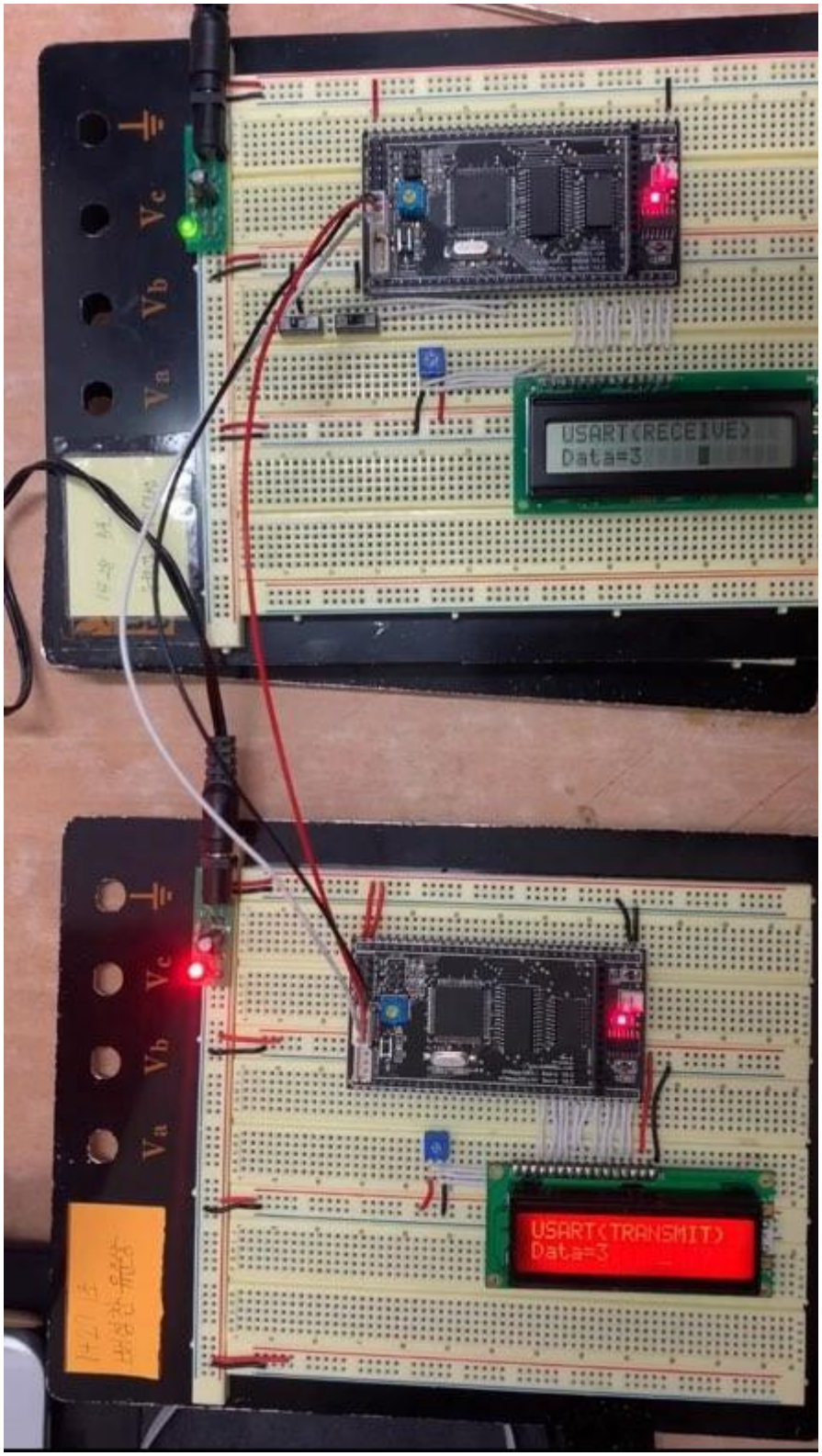


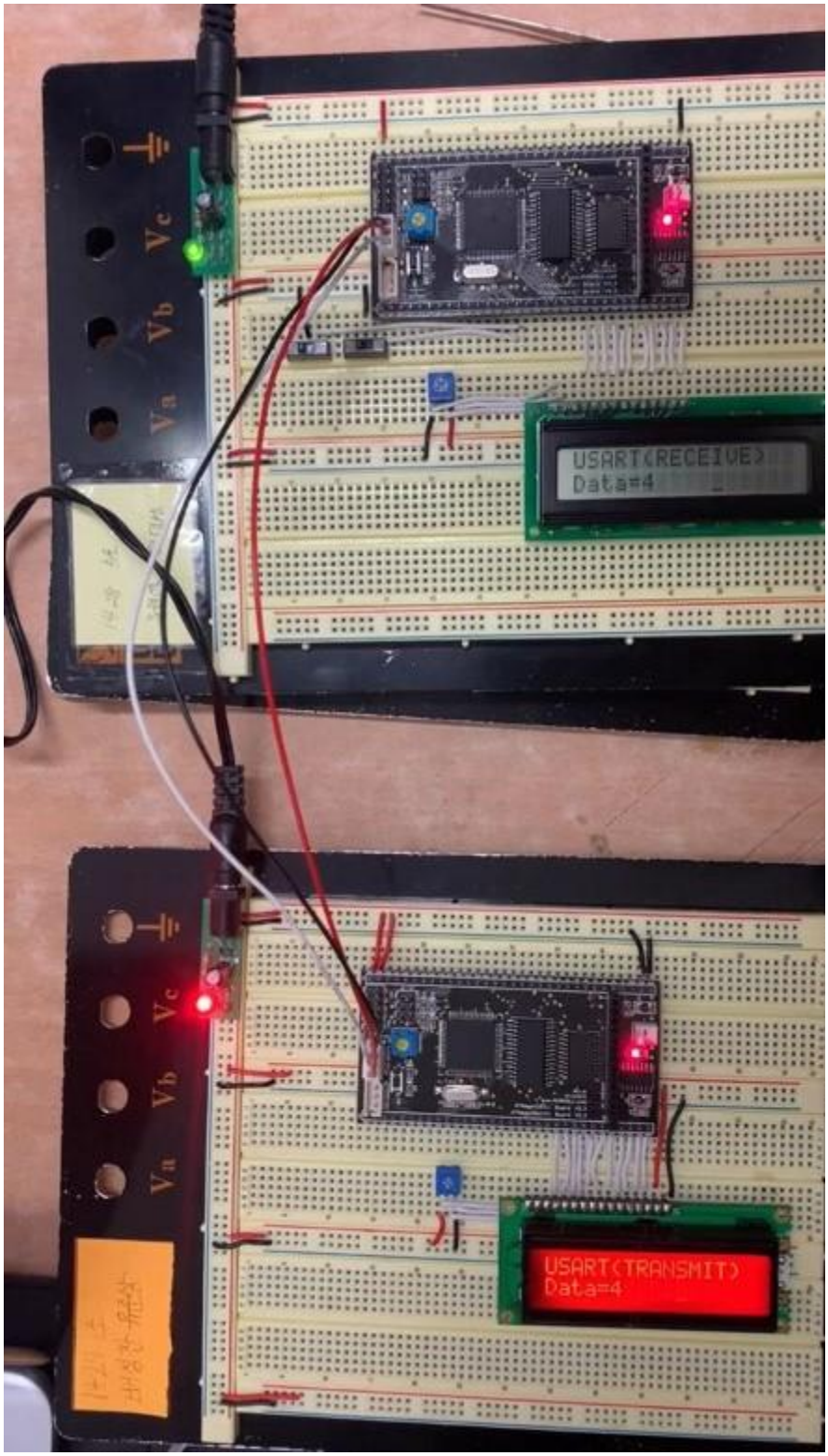
전체적인 연결 모습

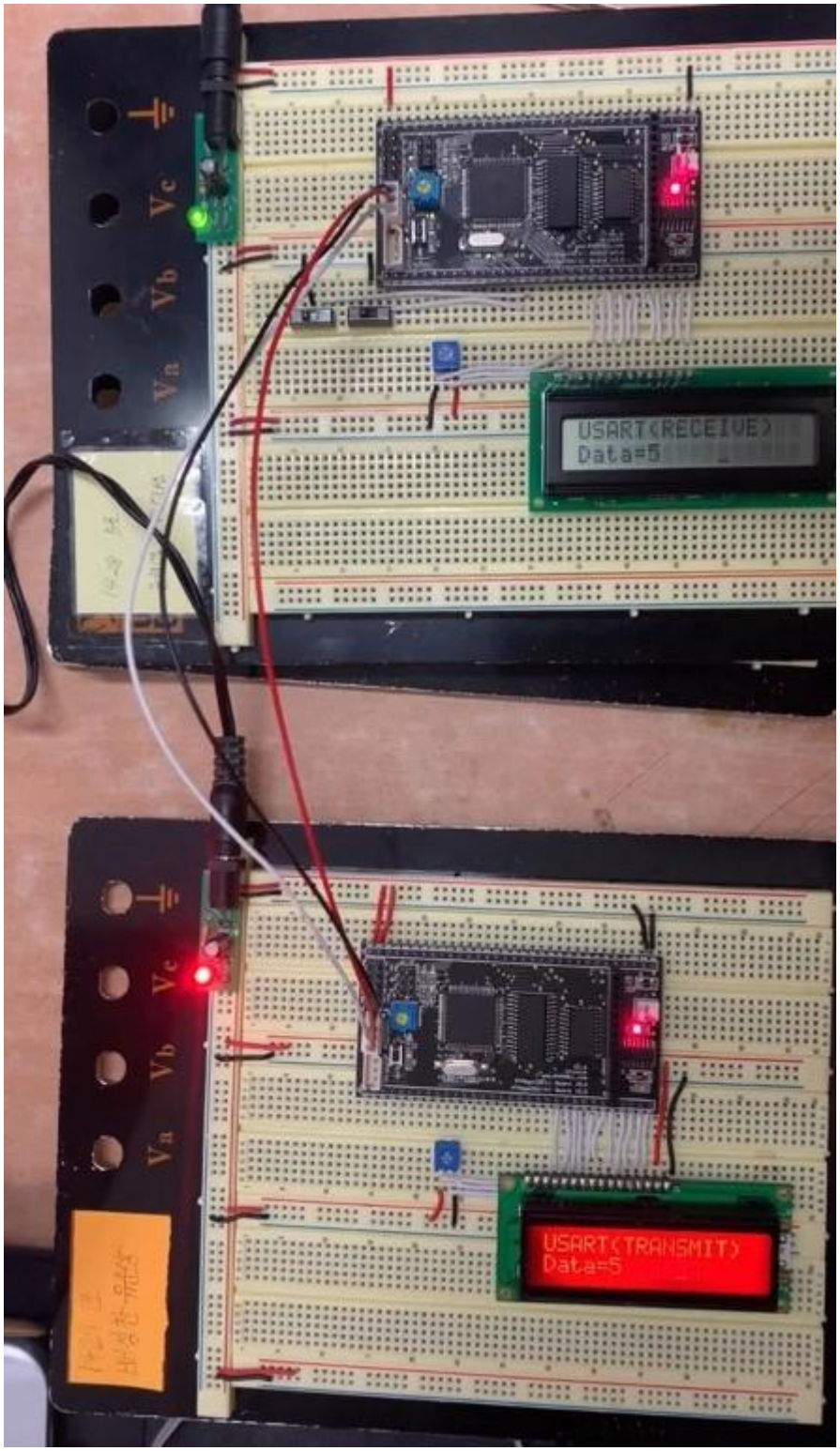


송, 수신이 잘 이루어 지는지 간단하게 몇가지 이미지로 결과를 보인다.

[A:송신, B:수신]

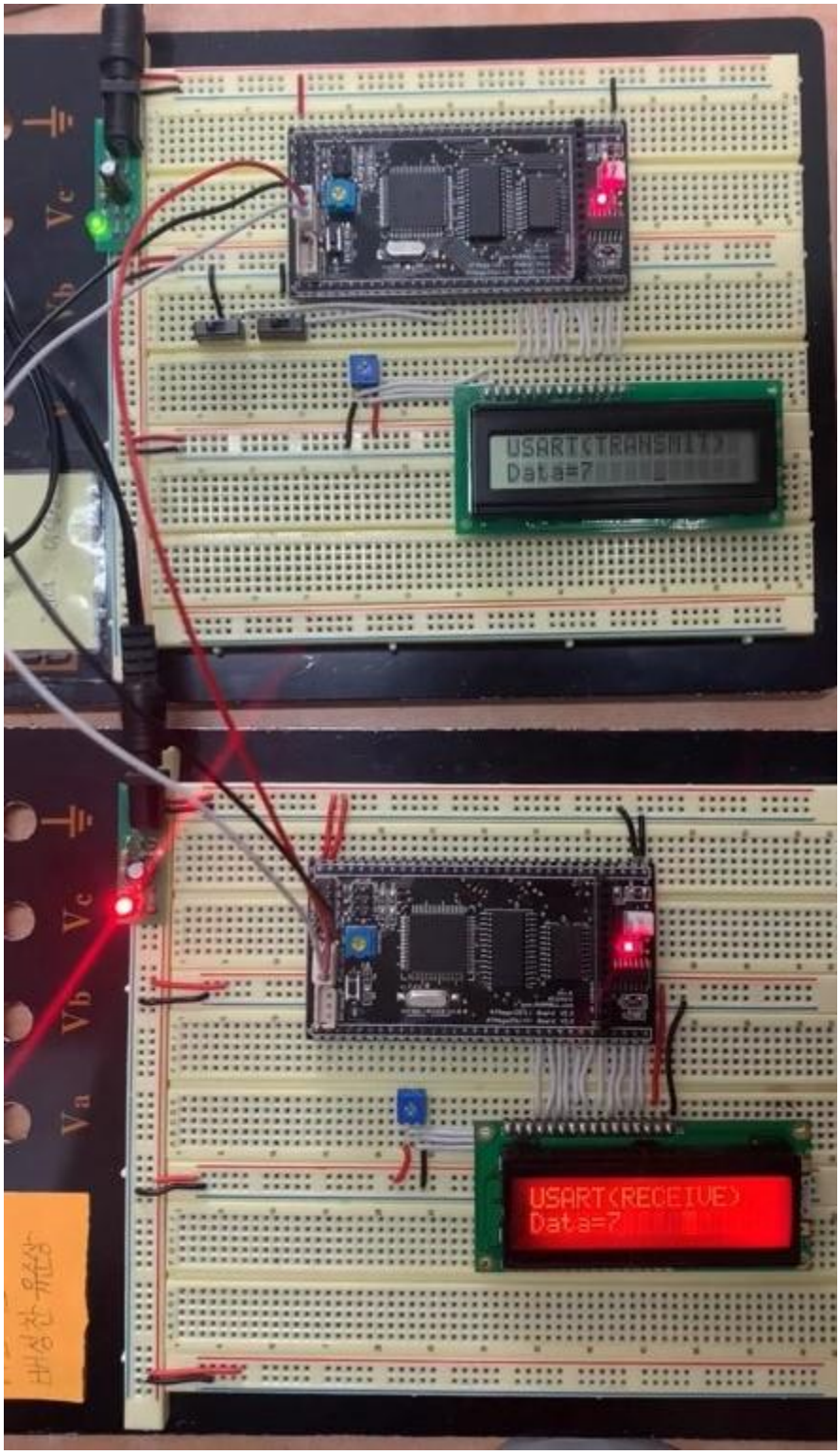

Vc
Vb
Va
USART(RECEIVE)
Data=3
Vc
Vb
Va
USART(TRANSMIT)
Data=3

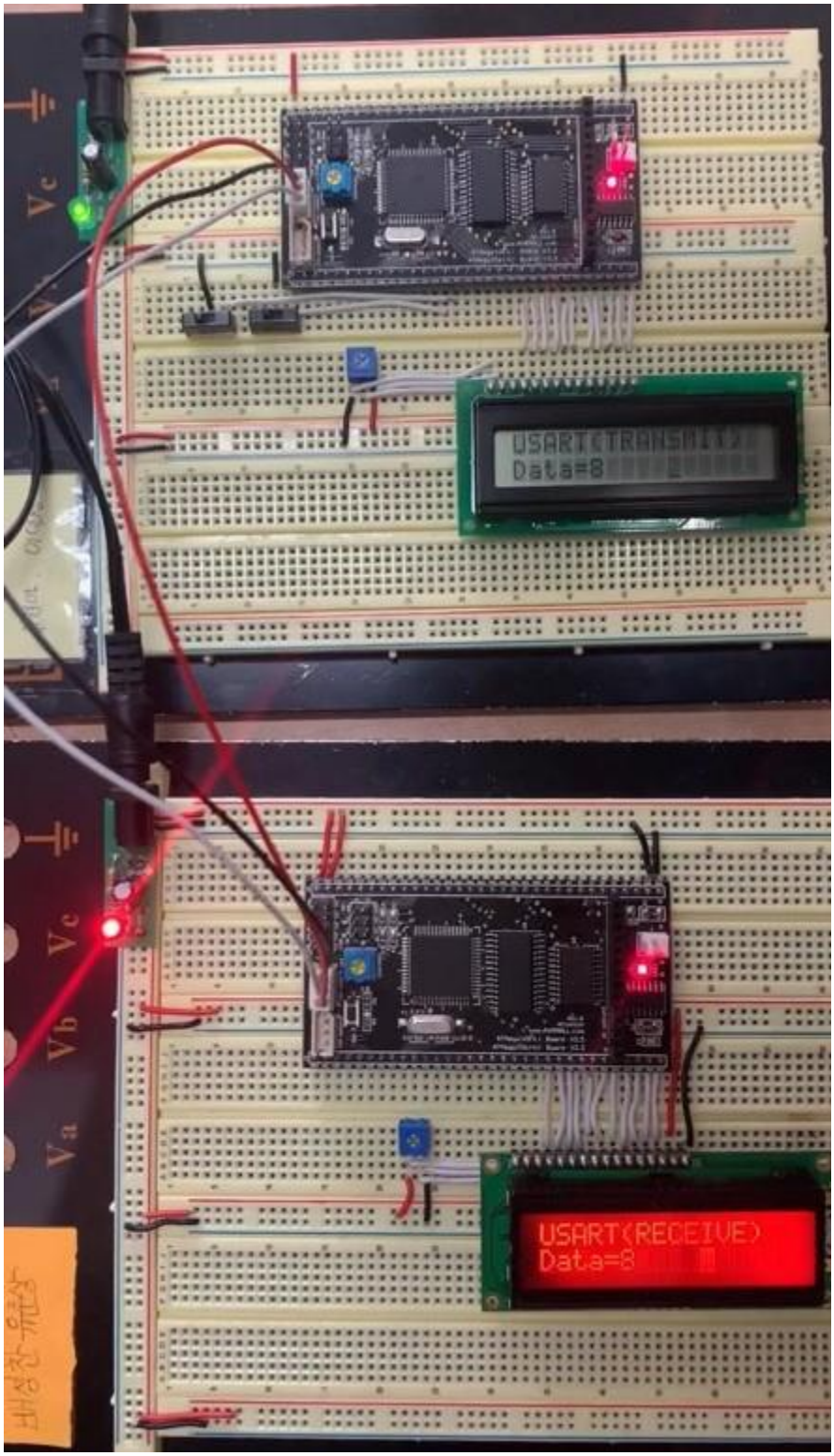

Vc
Vb
Va
USART(RECEIVE)
Data=4
Vc
Vb
Va
USART(TRANSMIT)
Data=4

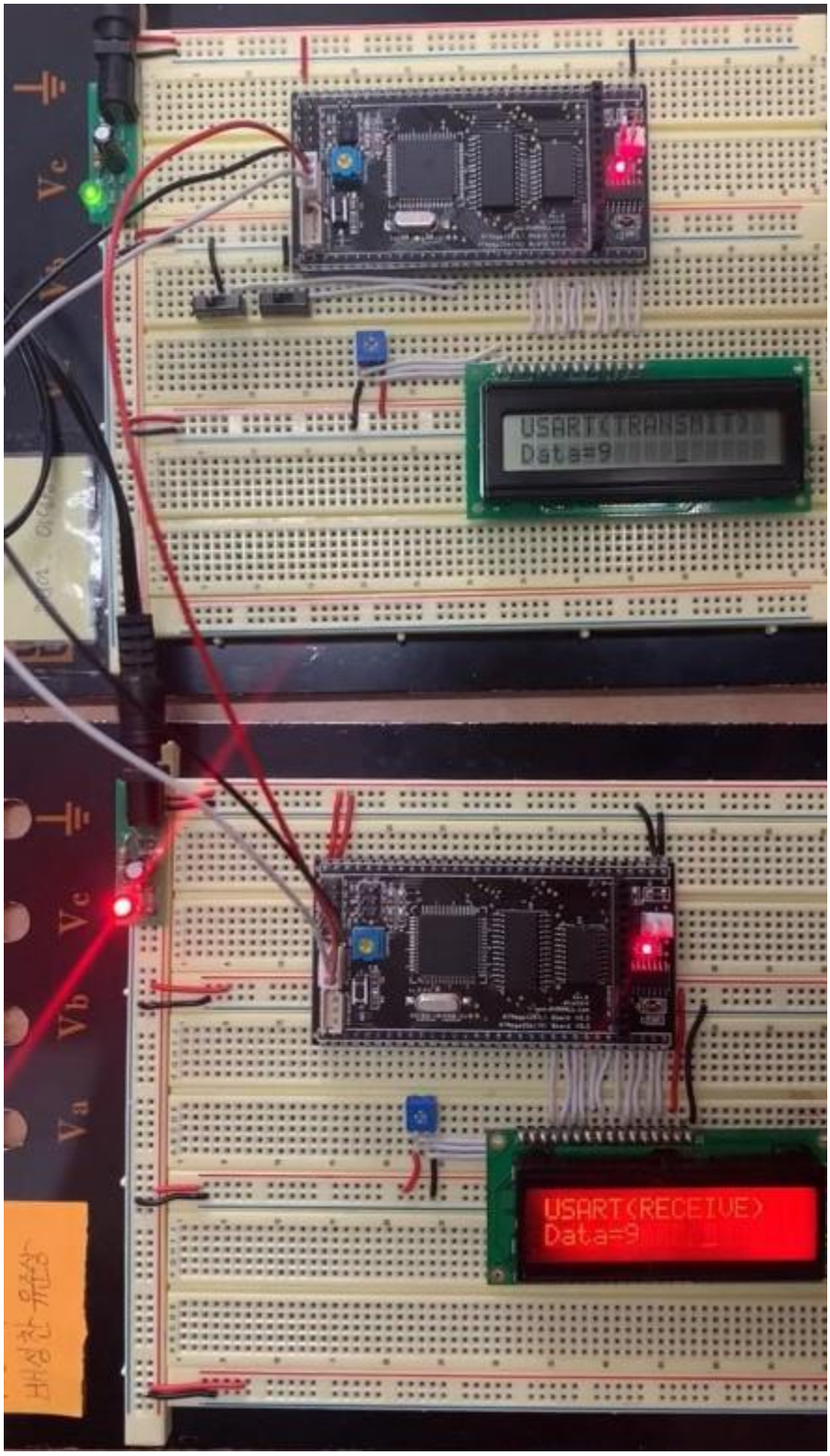


각 보드의 역할을 바꾸어서 실행한 결과

[A:수신, B:송신]


USART(TRANSMIT)
Data=7
USART(RECEIVE)
Data=7
Vc
Vb
Va


USART(TRANSMIT)
Data=8
USART(RECEIVE)
Data=8



각각 송신, 수신이 잘 이루어짐을 확인했다. 역할을 바꾸어서 할 때도 똑같이 잘 작동함을 확인했다.

Ⅴ 결과 및 토의

➔ 이번 실험에서는 Atmega128 2개 간의 USART 직렬통신을 수행하였다. 비동기 모드를 사용하여 각 보드의 내부 클럭으로 진행하였다. 송신, 수신이 잘 되는 지는 각 보드와 연결된 LCD로의 송신, 수신한 데이터가 잘 출력되는 지로 결과를 확인하였다. 각 LCD의 첫째 줄에는 해당 보드가 송신 파트인지, 수신 파트인지 표시하기 위해 "USART(TRANSMIT)", "USART(RECEIVE)" 를 출력하여 구분하였다. 그리고 지정한 1초 딜레이에 맞게 LCD의 둘째 줄에서 데이터가 형식에 맞고, 1씩 증가하는 것을 확인하였다. 또한 처음에는 A:송신, B:수신으로 진행하였고 각 파트를 바꾸어서도 진행할 때도 잘 작동함을 확인하였다. 본 실험을 통해 마이크로 컴퓨터 간의 직렬통신의 세세한 과정에 대해 공부하고 이해하며, 직접 수행해보는 좋은 경험을 했다. 또한 Atmega128을 가지고 조원과 두 달 남짓의 기간동안 다양한 실험을 수행하며 같이 고민하고 전략을 세우고 수정해가며 이론으로 공부한 것을 성공적인 실험으로 이어가는 과정을 매우 흥미롭게 진행한 것 같다.