

## 실험 6. USART 직렬 통신 실험

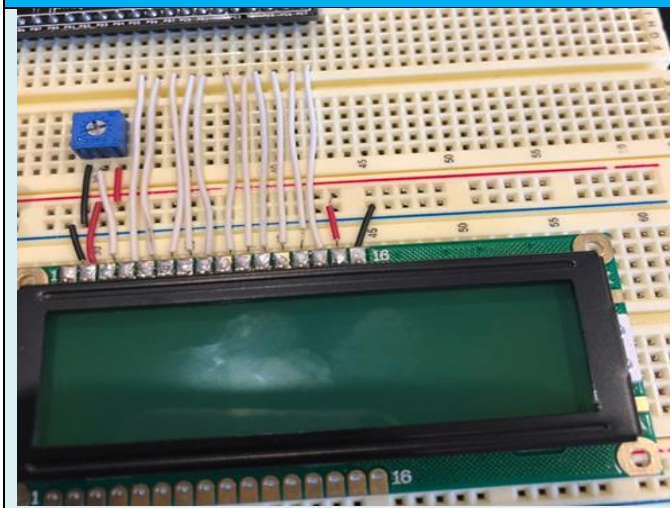
전자공학과 21611591 김 난 희

### 실험 목적

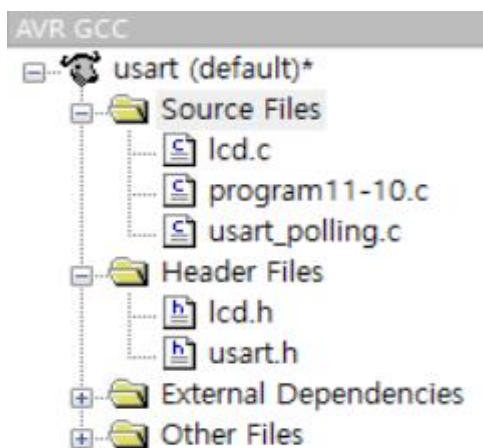
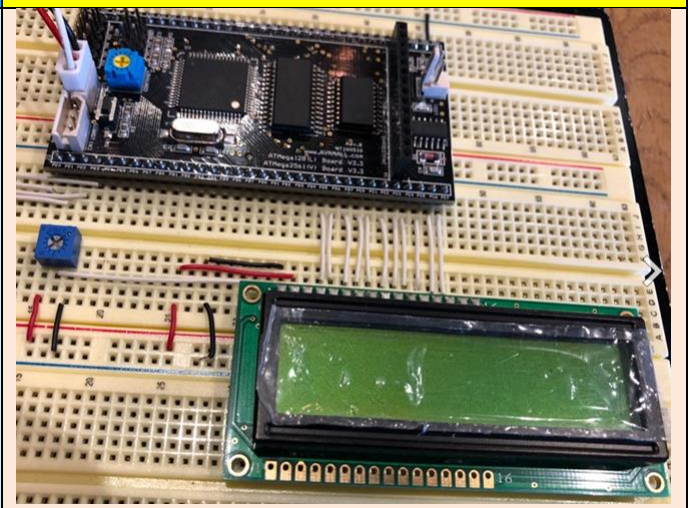
1. usart를 이해하고, 이를 사용하여 두 mcu 사이에 통신으로 이용할 줄 안다.
2. 아스키 코드를 알고 이로 약속된 문자와 숫자 사이의 관계를 이용하여 프로그램을 만들어본다.
3. 스위치와 LED를 회로에 추가하여 USART 통신을 이용하여 간단한 게임을 만들어본다.

### 1-1. 회로 분석과 구동

송신부



수신부



(송신부 수신부 공통, 소스파일, 헤더파일 모습)

송신부 main.c(파일 이름 : program11-10.c)

```
#include <stdio.h> // standard input output 헤더파일
#include <avr/io.h> // avr의 input output 헤더파일
#include <util/delay.h> // delay 구문 사용하기 위함
#include "lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언
#include "usart.h" //usart.c의 함수들을 사용하기 위한 usart 관련 함수 및 변수들이 정의된 헤더파일 선언
```

```
int main(void) // main 함수
{
```

```
    char lcd_string[2][MAX_LCD_STRING]; // 문자 혹은 문자열을 넣기 위한 배열 선언
```

```
    LCD_init(); // lcd 초기화 함수
```

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

```
    sprintf(lcd_string[0], " 11 and 12 "); // lcd_string[0]에 문자열 넣음
```

```
    LCD_str_write(0, 0, lcd_string[0]); // lcd에 출력
```

```
    char x[5] = {72,69,76,76,79}; // 아스키코드로 HELLO 의미
```

제어 문자			공백 문자			구두점			숫자			알파벳		
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`			
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a			
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b			
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c			
4	0x04	EOT	36	0x24	\$	68	0x44	D	100	0x64	d			
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e			
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f			
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g			
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h			
9	0x09	HT	41	0x29	)	73	0x49	I	105	0x69	i			
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j			
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k			
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l			
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m			
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n			
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o			
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p			

이 외에도 여러 문자, 숫자로 약속해놓은 것이 많음.

```
for(int i=0;i<5;i++){
    sprintf(lcd_string[0], " Send : %c",x[i]); // 보내는 값 lcd_string[0]에 넣음
    LCD_str_write(1, 0, lcd_string[0]); // lcd에 출력
    _delay_ms(1000); // 1초 delay 후 송신
    USART1_send(x[i],0); // 폴링을 이용한 문자 송신 //한글자씩 송신
}
return 0; //종료
```

```
}
```

수신부 main.c(파일 이름 : program11-10.c)

```
#include <stdio.h> // standard input output 헤더파일
#include <avr/io.h> // avr의 input output 헤더파일
#include <util/delay.h> // delay 구문 사용하기 위함
```

```

#include "lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언
#include "usart.h" //usart.c의 함수들을 사용하기 위한 usart 관련 함수 및 변수들이 정의된 헤더파일 선언
#define F_CPU 16000000UL //16MHz clock을 사용, 미리 옵션으로 저장하면 안 써줘도 됨
int main(void) // main 함수
{
    char lcd_string[2][MAX_LCD_STRING]; // 문자 혹은 문자열을 넣기 위한 배열 선언

    LCD_init(); // lcd 초기화 함수
    USART_init(USART1, 12); // USART1 보오레이트 : 38400(UBRR=12), 8MHz

    sprintf(lcd_string[0], " 11 and 12 "); // lcd_string[0]에 문자열 넣음
    LCD_str_write(0, 0, lcd_string[0]); // lcd에 출력

    char receive[5]= 0; //받은 문자를 저장하기 위한 배열 변수 선언

    for(int i=0; i<5; i++)
    {
        receive[i] = USART1_receive(0);
        // HELLO를 receive[]배열에 순서대로 받아 저장, receive(0)인 것은 stream사용 안함
    }
    _delay_ms(5000); // 문자를 제대로(끝까지) 받기 위한 5초 기다림

    sprintf(lcd_string[1], "Receive : %c%c%c%c%c", receive[0], receive[1], receive[2], receive[3], receive[4]); //받은 문자를 순서대로 lcd_string[0]에 저장
    LCD_str_write(1, 0, lcd_string[1]); // lcd에 출력

    return 0; //종료
}

```

#### usart.c

```

#include <avr/io.h> // avr의 input output 헤더파일
#include "usart.h" //usart.c의 함수들을 사용하기 위한 usart 관련 함수 및 변수들이 정의된 헤더파일 선언
// 폴링 USART 초기화
void USART_init(unsigned char ch, unsigned int ubrr_baud)
{
    if( ch == USART0){ //USART0을 사용
        UCSR0B |= 1<<RXEN0 | 1<<TXEN0;
    }
}

```

#### USARTn Control and Status Register B – UCSRnB

Bit	7	6	5	4	3	2	1	0	
	RXCIE <sub>n</sub>	TXCIE <sub>n</sub>	UDRIE <sub>n</sub>	RXEN <sub>n</sub>	TXEN <sub>n</sub>	UCSZ <sub>n2</sub>	RXB8 <sub>n</sub>	TXB8 <sub>n</sub>	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	

만약 USART0을 사용하면 다음 UCSRnB의 n에 0이 들어간다. 나머지 레지스터의 n에도 마찬가지이다.

RXEN0과 TXEN0에 HIGH(1)로 SET될 경우, 전송전에 미리 송수신을 허용한다. 나머지 비트는 0으로 초기화한 것과 같다.

5~7번 비트는 Interrupt와 관련된 비트이고 HIGH(1)로 설정하면 Interrupt를 허용한다는 것이다.

UCSZn2는 UCSRnC의 1~2번 비트(UCSZn0과 UCSZn1)와 함께 사용한다. character size를 정한다. 즉, 몇 비트를 전송할지 정하는 역할이다.

```
UBRR0H = ubrr_baud >> 8;
UBRR0L = ubrr_baud;
}else if(ch == USART1){
    UCSR1B |= 1<<RXEN1 | 1<<TXEN1;
```

### USARTn Control and Status Register B – UCSRnB

Bit	7	6	5	4	3	2	1	0	
	RXCIE <sub>n</sub>	TXCIE <sub>n</sub>	UDRIE <sub>n</sub>	RXEN <sub>n</sub>	TXEN <sub>n</sub>	UCSZn2	RXB8 <sub>n</sub>	TXB8 <sub>n</sub>	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	

여기서도 마찬가지로, USART1을 사용하면 다음 UCSRnB의 n에 1이 들어가며, 각 3번, 4번 해당 비트의 역할은 같다.

```
UBRR1H = ubrr_baud >> 8;
UBRR1L = ubrr_baud;
}
}
```

```
int USART1_send(char data, FILE *stream) // 폴링에 의한 문자 전송
{
```

```
    while ( !( UCSR1A & (1<<UDRE1)) ); // UDR 레지스터가 빌 때까지 폴링한다.
```

### USART Control and Status Register A – UCSRnA

Bit	7	6	5	4	3	2	1	0	
	RXC <sub>n</sub>	TXC <sub>n</sub>	UDRE <sub>n</sub>	FEN	DOR <sub>n</sub>	UPEN	U2X <sub>n</sub>	MPCM <sub>n</sub>	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UDRE1 레지스터는 USART Data Register Empty로 데이터가 비었을 때 참이 된다. 참이 되면 while(!1)은 while(0)이 되어 반복문을 빠져나온다.

```
    UDR1 = data; // UDR 레지스터에 값을 기록한다. //TX, RX 번지 같음
    return data; // data 리턴
}
```

```
int USART1_receive(FILE *stream) // 폴링에 의한 문자 수신
{
```

폴링을 이용하여 usart1 채널로 수신하는 함수이다.

```
    while ( !(UCSR1A & (1<<RXC1)) ); // UDR 레지스터에 문자 수신 검사
```

## 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	

RXCn 비트를 반복 검사한다. 참이 되면 while(!1)로 거짓이 되므로 반복문을 빠져나온다. RXCn 비트는 수신이 완료되면 참이 된다. 즉, 수신이 완료될 때까지 기다리는 루프문이다.

```
return UDR1;
```

반복문을 빠져나오면 UDR1을 반환한다.

```
}
```

## usart.h

```
#ifndef __USART_H__
#define __USART_H__
#include <stdio.h> // standard input output 헤더파일

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

//extern은 외부에서도 사용할 수 있게 함
extern void USART_init(unsigned char ch, unsigned int ubrr_baud); //사용할 usart 0인지 1인지 결정, 보레이트 결정(통신 속도)
extern int USART1_send(char data, FILE *stream); //문자 전송할 때 사용하는 함수
extern int USART1_receive(FILE *stream); //문자 수신할 때 사용하는 함수

#endif
```

## lcd.c

```
#include <avr/io.h> // avr의 input output 헤더파일
#include "lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언

#define RS PD5 // LCD 문자디스플레이에 연결된 포트D의 핀번호
// RS는 명령인지 DATA 인지 결정
#define RW PD6 // 포트D의 6번 핀에 RW 핀 연결
#define E PD7 // Enable 연결

void gen_E_strobe(void) // 사용할 수 있게 Enable 해주는 함수
{
    volatile int i; // volatile 변수로, 사이즈 최적화를 피함.

    PORTD |= 1<<E; // E 신호를 High로
    for(i=0; i<10; i++); // E 스트로브 신호를 일정기간 High로 유지
    PORTD &= ~(1<<E); // E 신호를 Low로
}

void wait_BusyFlag(void) // busy flag를 읽어 0이 될 때까지 기다림
{
    volatile int i; // volatile 변수로, 사이즈 최적화를 피함.
    unsigned char bf; // buffer를 의미하는 unsigned 형 변수 선언
```

```

DDRC = 0x0; // 포트C를 입력핀으로 설정
PORTD = (PORTD & ~(1<<RS)) | 1<<RW; // RS <- Low, RW <- High
do{ // 먼저 실행 후 while()의 조건을 보고 후 판단
    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
}

void LCD_command(unsigned char data) // lcd에 주는 명령 함수, 명령어 보냄
{
    wait_BusyFlag(); // busy flag가 0될 때까지 대기
    DDRC = 0xFF; // 포트C를 출력핀으로 설정
    PORTC = data; // data 출력
    PORTD &= ~(1<<RS | 1<<RW); // RS <- 0, RW <-0
    gen_E_strobe(); // E 스트로브 신호 만들기
}

void LCD_data_write(unsigned char data) // 여기 data는 아스키 코드가 들어갈 것.
{
    wait_BusyFlag(); // busy 체크
    DDRC = 0xFF; // 출력으로 설정
    PORTC = data; // data의 값이 여기 들어옴
    PORTD = (PORTD | 1<<RS) & ~(1<<RW); // RS <- 1, RW <-0
    // 데이터 내보냄 //명령어 x
    gen_E_strobe(); // E 스트로브 신호 만들기
}

void LCD_init(void) // lcd 초기화 함수
{
    DDRD |= 1<<RS | 1<<RW | 1<<E; // RS, RW, E 핀을 출력핀으로 설정

    PORTD &= ~(1<<RS | 1<<E | 1<<RW); // 초기에 RS, E, RW <- 0

    LCD_command(0x3C); // 인터페이스, 디스플레이 설정
    LCD_command(0x02); // cursor 초기화
    LCD_command(0x01); // clear display
    LCD_command(0x06); // entry 모드
    LCD_command(0x0F); // display on/off
}

LCD 사용 순서는 다음과 같이 진행된다. 위의 LCD_init(void) 와도 매칭해보며 비교해보면 된다.

```



## LCD 초기화 순서

1. 초기화 전 30msec 를 기다림
2. **Function set 명령(0011xx00B)을 내보냄**
3. **Display On/Off control 명령(00001xxxB) 을 내보냄**
4. **Entry Mode set 명령(000001xxB)을 내보냄**
5. **DD RAM 어드레스를 내보냄**
6. 표시할 문자데이터를 내보냄
7. 5,6과정을 반복

Function set에는 다음과 같은 것들이 초기에 설정된다.

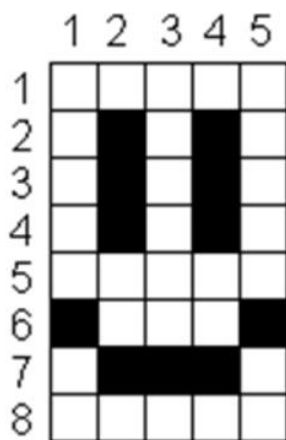
인터페이스/ 디스플레이 설정	DL	1 : 8비트 인터페이스 0 : 4비트 인터페이스
	N	1 : 두 줄 표시 0 : 한 줄 표시
	F	1 : 문자 5×10 도트 0 : 문자 5×7 도트

font를 보면 우리는 5x10 도트를 사용한다.

이런 것들이 어떻게 디스플레이되는 지 살펴보면, 다음 그림과 같다.

## A Custom 5x8 Pixel Character:

Image Coding:



Binary Coding:

→ 0b000 00000  
→ 0b000 01010  
→ 0b000 01010  
→ 0b000 01010  
→ 0b000 00000  
→ 0b000 10001  
→ 0b000 01110  
→ 0b000 00000

1 = Black, 0 = White

위 그림은 5x8 도트를 예시로 나타낸 것이다. 검은색으로 디스플레이할 모양을 다음과 같이 1로 설정해주어 보이게 한다.

```
void set_cursor(unsigned int row, unsigned int col) // 커서 위치 설정
{
```

```
    LCD_command(0x80 + (row % 2) * 0x40 + (col % 0x40));
```

```
} // 0x80은 DDRAM 주소이고 row를 설정해주는 부분과 column을 설정하는 부분으로 되어있다.
```

만약 위의 함수에 set\_cursor(0,0)을 넣어주면, 1000 0000이 들어가, 0x80 command를 실행한다. 만약 (1,2),를 넣어주면, 0x80 + 1x(0x40) + 2 이므로,

```
= 1000 0000
  0100 0000
  0000 0010
```

-----  
1100 0010 = C2 가 된다.  
노란색으로 표시해 놓은 부분은 DDRAM 주소 부분이다.

결국 아래의 함수를 통해 위 set\_cursor 내부의 Command 함수는 data 주소를 날리는 부분이  
라 이해할 수 있겠다.

```
// 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); // 위 함수 인자로부터 받은 곳에 Cursor 위치 조정, DDRAM 주
소
    for(i=0; (i+col < MAX_LCD_STRING) && (str[i] != '\0'); i++) // \0은 문자 끝을 말함
        LCD_data_write(str[i]); // data를 날림
}
```

#### lcd.h

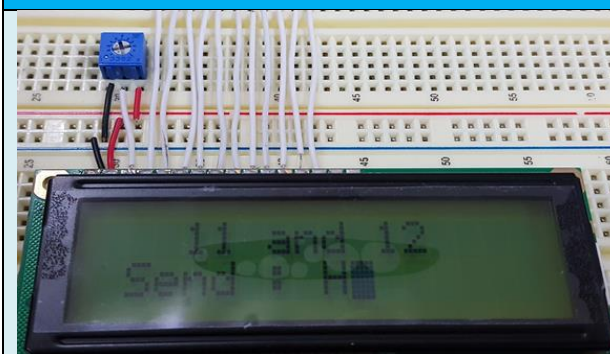
```
#ifndef __LCD_H__
#define __LCD_H__

#define MAX_LCD_STRING    0x40 // 한 줄에 64글자 최대 디스플레이

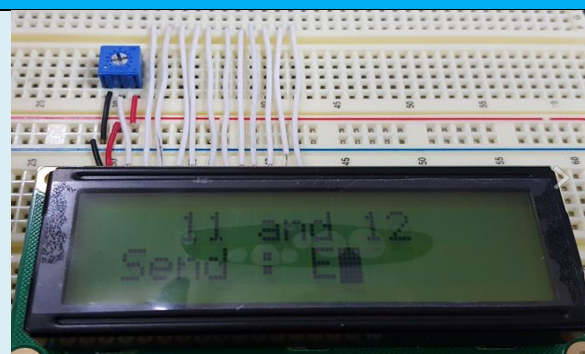
// 아래는 외부에서도 함수를 쓸 수 있게 extern 으로 선언해 놓은 부분.
extern void gen_E_strobe(void);
extern void wait_BusyFlag(void);
extern void LCD_command(unsigned char data);
extern void LCD_data_write(unsigned char data);
extern void LCD_init(void);
extern void set_cursor(unsigned int row, unsigned int col);
extern void LCD_str_write(unsigned int row, unsigned int col, char *str);
#endif
```

## 1-2. 실험 결과

송신부 결과 : 순서대로 H, E, L, L, O를 내보낸다. LCD에도 똑같이 출력

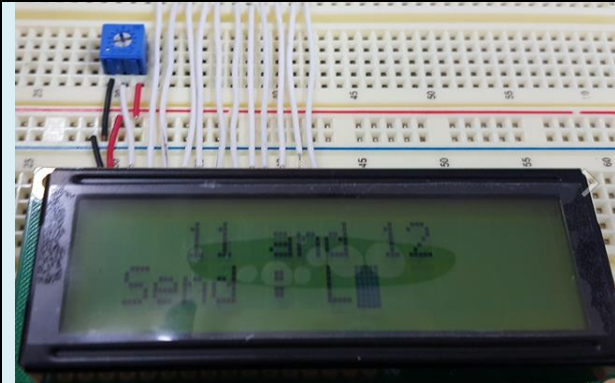
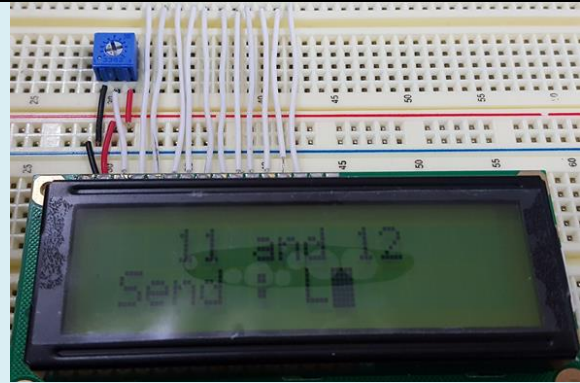
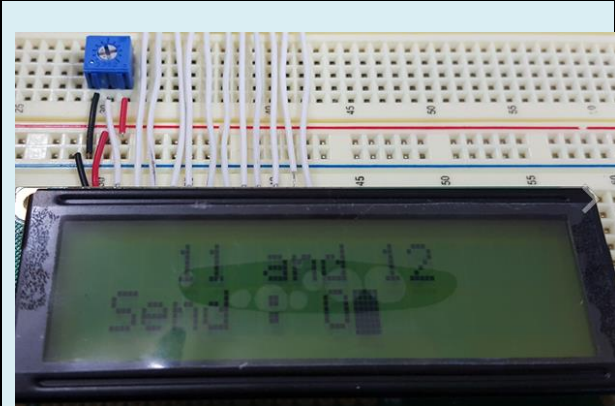
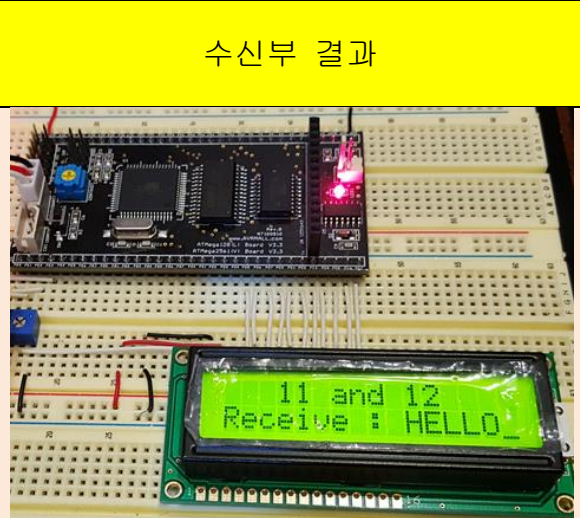


H를 우선 LCD 출력 후 1초후 송신



E를 우선 LCD 출력 후 1초후 송신



	
L를 우선 LCD 출력 후 1초후 송신	L를 우선 LCD 출력 후 1초후 송신
	수신부 결과
O를 우선 LCD 출력 후 1초후 송신	
	글자를 모두 받은 후 모아서 LCD 출력

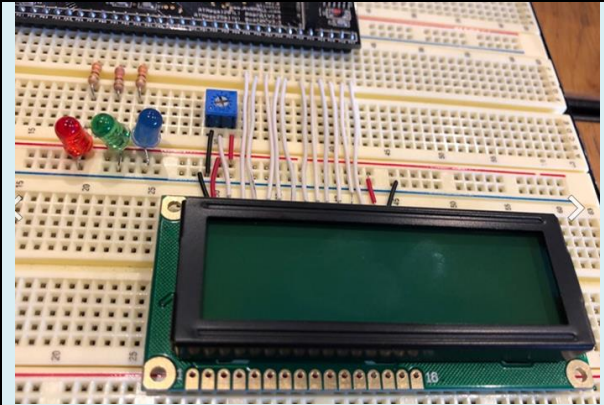
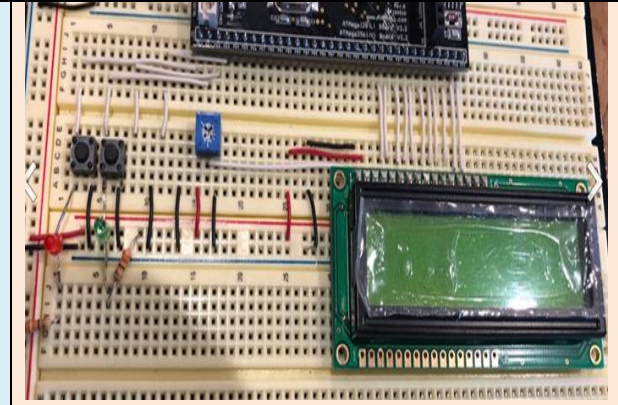
송신하는 문자를 문자열로 만들어 보낼 수 도 있었지만, usart.c에 이미 만들어진 USART 함수들을 이용하여 실험을 해보고 싶었다. USART 함수들을 수정하면 문자열로도 송신이 가능할 것 같다. 숫자와 문자사이의 변환을 이용하여 아스키 코드를 잘 이용할 수 있게 되었다. 송신할 때의 딜레이가 수신받을 때에도 영향을 미친다는 것을 알았다. 수신할 때의 딜레이를 적절히 설정하여 글자가 중복되어 받아지는 것을 수정할 수 있었다. 딜레이 설정을 빼고 좀 더 완벽한 프로그램이 될 수 있도록 다른 방법을 모색해봐야할 것 같다.

## 2. 추가 실험

### (1) 청기 백기 게임

#### 2-1. 실험 회로

송신부	수신부
-----	-----

	
<p>백기 == 붉은색 LED == PB0</p> <p>청기 == 초록색 LED == PB2</p> <p>송신 완료 == 파란색 LED == PB4</p>	<p>백기 == 붉은색 LED == PE0 == 왼쪽 스위치</p> <p>청기 == 초록색 LED == PE1 == 오른쪽 스위치</p>

이전의 본 실험에서 송신부는 깃발을 표현하기 위한 LED를 추가해주었고, 수신부는 각 깃발 색에 매칭되는 스위치가 눌러짐 표현을 위한 LED를 사용해주었다. 물론 깃발에 매칭되는 문제를 풀기 위한 스위치도 사용했다.

## 2-2. 프로그램 분석 및 결과

송신부와 수신부의 main 함수가 포함된 소스파일(.C)을 제외하고는 다 같습니다.

송신부 main.c(파일 이름 : program11-10.c)	
#include	<stdio.h> // standard input output 헤더파일
#include	<avr/io.h> // avr의 input output 헤더파일
#include	<util/delay.h> // delay 구문 사용하기 위함
#include	"lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언
#include	"usart.h" //usart.c의 함수들을 사용하기 위한 usart 관련 함수 및 변수들이 정의된 헤더파일 선언
#define D0 PB0	// 붉은색 LED, 백기
#define D1 PB2	// 초록색 LED, 청기
#define D2 PB4	// 파란색 LED, 수신 완료
int main(void)	// main 함수
{	
DDRB  = (1<<D0) (1<<D1) (1<<D2);	// LED는 출력으로 설정
char lcd_string[2][MAX_LCD_STRING];	// LCD에 출력할 문자열이 들어가는 배열 변수
LCD_init();	// LCD 초기화
USART_init(USART1, 12);	// USART1 보오레이트 : 38400(UBRR=12), 8MHz
//본 실험에서 자세히 설명하여 여기서 설명은 생략합니다	
sprintf(lcd_string[0], "    Flag Game    ");	// lcd_string[0]에 출력할 문자 저장
LCD_str_write(0, 0, lcd_string[0]);	// LCD 첫번째 줄에 출력

```

int question[5] = {0,1,0,0,1}; // 문제 정의, 백기, 청기, 백기, 백기, 청기

for(int i=0;i<5;i++) // 배열 5칸, 5문제 제시
{
    sprintf(lcd_string[0], " Send : %d",question[i]); // 송신 숫자 lcd_string에 저장
    LCD_str_write(1, 0, lcd_string[0]); // 송신 숫자 for문의 i에 따라 순서대로 출력

    if(question[i]==0){ PORTB |= (1<<D0); PORTB &= ~(1<<D1);} // 백기이면, 백기에 해당하는 붉은색 LED ON, 청기에 해당하는 초록색 LED OFF
    else { PORTB &= ~(1<<D0); PORTB |= (1<<D1);} // 청기이면, 청기에 해당하는 초록색 LED ON, 백기에 해당하는 붉은색 LED OFF

    _delay_ms(1000); // 바로 송신되는 것을 방지하기 위한 1초 delay
    USART1_send(question[i],0); // 폴링을 이용한 송신

    //백기이면, 0을 송신
    //청기이면, 1을 송신

    PORTB &= ~((1<<D0)|(1<<D1)); // 청기와 백기에 해당하는 LED를 켜후 다시 끄기
    _delay_ms(50); // 0.05초 딜레이해야 꺼졌다가 다음 문제(다음 LED 깃발) 색이 보임
    // 최대한 짧게, LED가 꺼졌다 켜지는 것이 보일 정도
}

sprintf(lcd_string[0],
"Complete : %d%d%d%d%d",question[0],question[1],question[2],question[3],question[4]);
// 모두 송신 완료 후 전송한 청기 백기 5문제를 모두 lcd_string[0]에 문자열 저장
LCD_str_write(1, 0, lcd_string[0]); // 위에서 저장한 문자열 LCD 둘째 줄에 출력
PORTB |= (1<<D2); PORTB &= ~((1<<D0)|(1<<D1)); // 모두 송신 완료 후 파란색 LED
를 ON하고 나머지 LED(백기, 청기)를 OFF하며 마침

return 0; //종료
}

```

수신부 main.c(파일 이름 : program11-10.c)

```

#include <stdio.h> // standard input output 헤더파일
#include <avr/io.h> // avr의 input output 헤더파일
#include <util/delay.h> // delay 구문 사용하기 위한
#include "lcd.h" // lcd.c의 함수들을 사용하기 위한 함수들이 정의된 헤더파일 선언
#include "usart.h" //usart.c의 함수들을 사용하기 위한 usart 관련 함수 및 변수들이 정의된 헤더파일 선언

#define F_CPU 16000000UL // 미리 옵션으로 정해주면 따로 선언할 필요 없음

int check(int receive[],int answer[]) // 수신한 청기 백기 문제와 입력한 답이 일치하는지 확인
하는 함수
{
    for(int i=0;i<5;i++) // 5문제 하나 하나를 비교하며 확인함
    {
        if(receive[i]!=answer[i]) return 1; // 하나하나 비교하며 답이 다를 경우 바로 함수를 종료
        하고 1을 반환함. 함수는 int형이므로 적절함
    }
    return 0; // 답을 모두 비교후 5문제 모두 일치할 경우 0을 반환하고 함수를 종료함
}

```

```

int main(void) // main 함수
{
    char lcd_string[2][MAX_LCD_STRING]; // LCD에 출력할 문자열이 들어가는 배열 변수

    LCD_init();// LCD 초기화

    USART_init(USART1, 12); // USART1 보오레이트 : 38400(UBRR=12), 8MHz
    //본 실험에서 자세히 설명하여 여기서 설명은 생략합니다

    sprintf(lcd_string[0], " Flag Game // lcd_string[0]에 출력할 문자 저장
    LCD_str_write(0, 0, lcd_string[0]); // LCD 첫번째 줄에 출력
    int receive[5]; // 수신한 숫자 배열을 저장할 배열 변수 선언

    for(int i=0; i<5; i++)
    {
        receive[i]= USART1_receive(0); // 여기서 폴링을 이용하여 수신받음
    }

    _delay_ms(1000); // 모두 수신받은 후, 완벽한 수신을 위한 1초 기다림

    DDRE &= ~((1<<PE1)|(1<<PE0)); //스위치 입력 핀으로 설정
    PORTE |= (1<<PE1)|(1<<PE0); // 입력 핀들을 내부 풀업저항으로 연결

    int answer[5]; // 스위치로 입력한 숫자가 저장될 배열 변수
    _delay_ms(500); // 스위치로 답을 입력하기 전(문제를 풀기 전) 0.5초 딜레이
    for(int i=0; i<5; i++)
    {
        if(!(PINE & (1<<PE0))) //풀업저항이므로 스위치를 누르면 LOW가 입력됨
        {
            answer[i]=0; _delay_ms(500); i++;}
            // 백기(붉은색 LED)에 해당하는 스위치를 누를 경우 0이 answer[]에 저장됨
            // 0.5초 딜레이를 통해 다음 입력을 기다림
            // 다음 입력을 위한 i++
        else if(!(PINE & (1<<PE1)))
        {
            answer[i]=1; _delay_ms(500); i++;}
            // 청기(초록색 LED)에 해당하는 스위치를 누를 경우 1이 answer[]에 저장됨
            // 0.5초 딜레이를 통해 다음 입력을 기다림
            // 다음 입력을 위한 i++
        } // 딜레이를 통해 소프트웨어적으로 채터링 방지

        // 문제를 모두 푼 후, 제시한 문제의 답을 모두 출력
        sprintf(lcd_string[1],
input : %d%d%d%d%d",answer[0],answer[1],answer[2],answer[3],answer[4]);

        //만약 답으로 백기 스위치, 청기 스위치, 백기 스위치, 백기 스위치, 청기 스위치를 입력할
        경우 01001이 출력됨
        //만약 답으로 청기 스위치, 청기 스위치, 백기 스위치, 백기 스위치, 청기 스위치를 입력할
        경우 11001이 출력됨

        LCD_str_write(1, 0, lcd_string[1]); // LCD에 출력
        _delay_ms(2000); // 완벽히 출력되고 다음 LCD 출력전 잘 보이기 위한 2초 딜레이

        int result= check(receive,answer); // 제시한 답과 수신받은 문제와 일치하는지 판단하는 함

```



수의 반환값을 저장

//result == 0 : 5 문제 모두 맞춤

//result == 1 : 1 문제라도 틀림

if(result==0) sprintf(lcd\_string[1], " Correct "); // 맞으면 lcd\_string[1]에 맞다고 저장

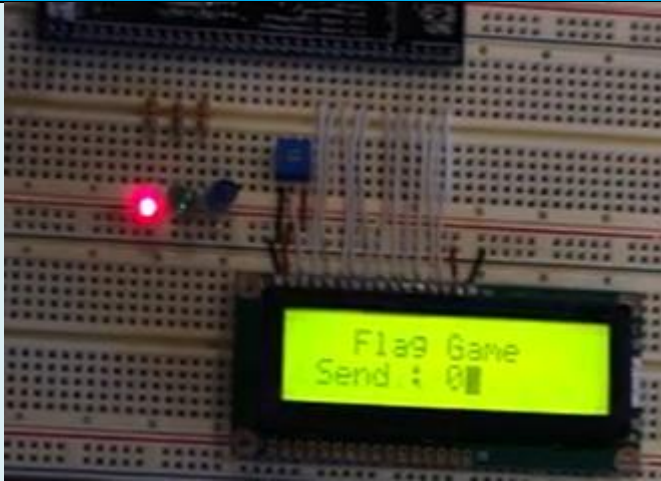
else sprintf(lcd\_string[1], " Lose "); // 틀리면 lcd\_string[1]에 졌다고 저장

LCD\_str\_write(1, 0, lcd\_string[1]); // 위에서 저장한 String을 LCD에 출력

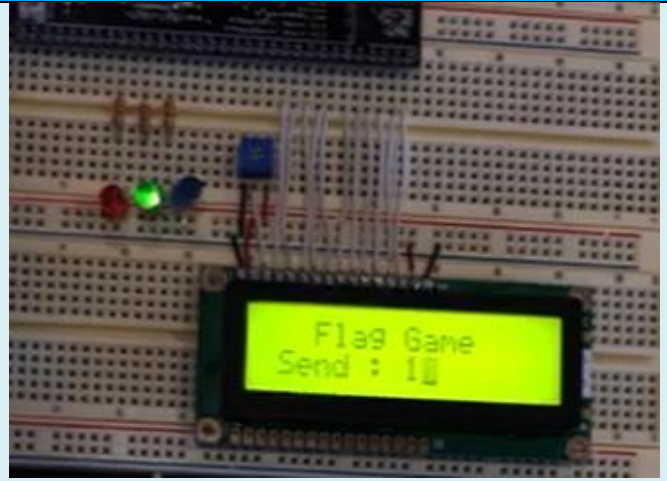
return 0; //종료

}

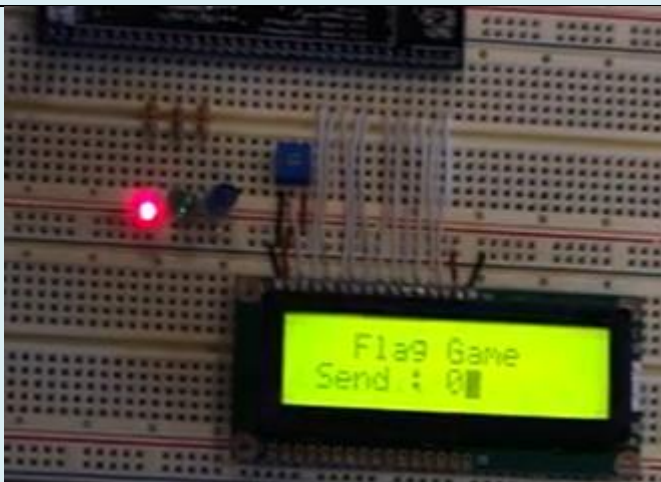
## 송신부



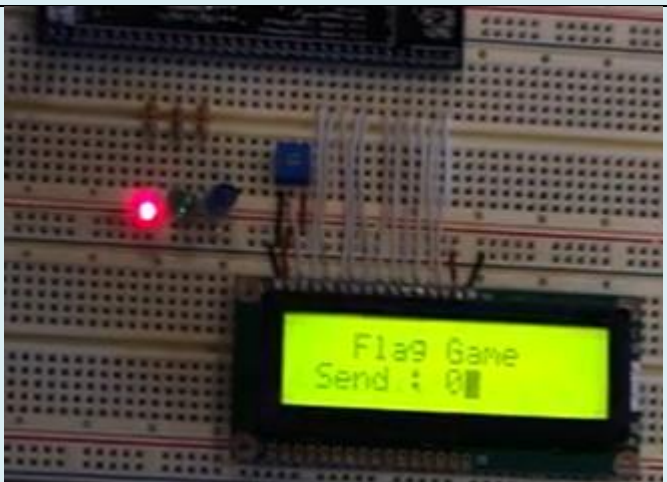
첫 번째 문제 송신 : 백기 == 붉은색 LED == 0



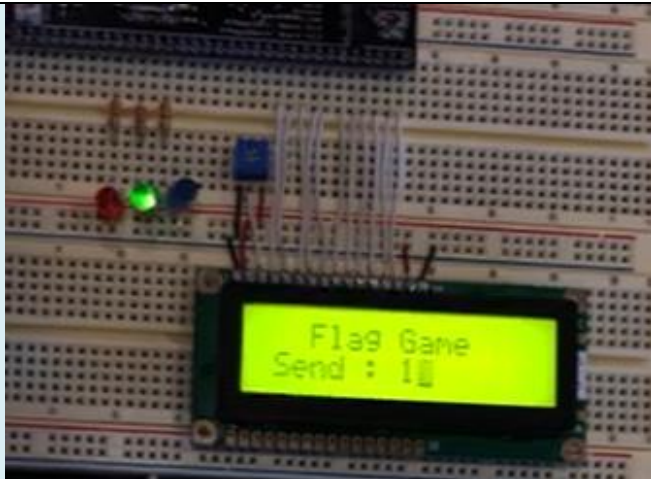
두 번째 문제 송신 : 청기 == 초록색 LED == 1



세 번째 문제 송신 : 백기 == 붉은색 LED == 0



네 번째 문제 송신 : 백기 == 붉은색 LED == 0

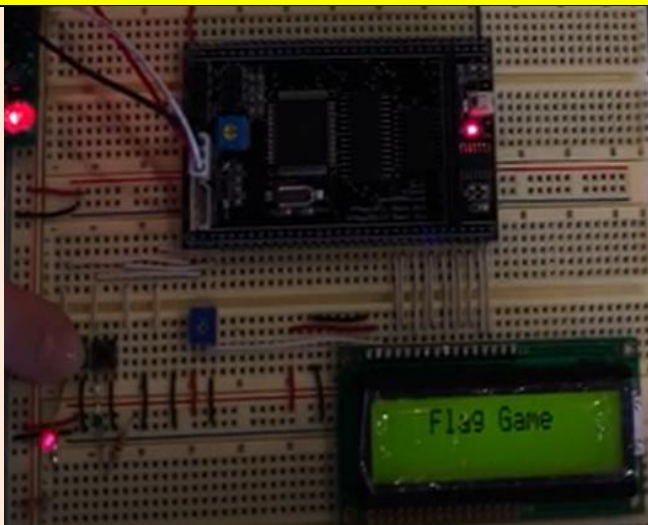


다섯 번째 문제 송신 : 청기 == 초록색 LED ==1

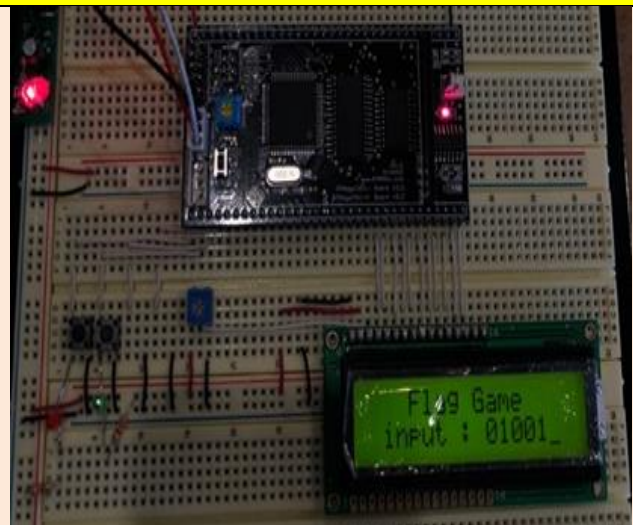


모두 송신 완료 후, 송신한 숫자들을 모두 합쳐 출력함

### 수신부



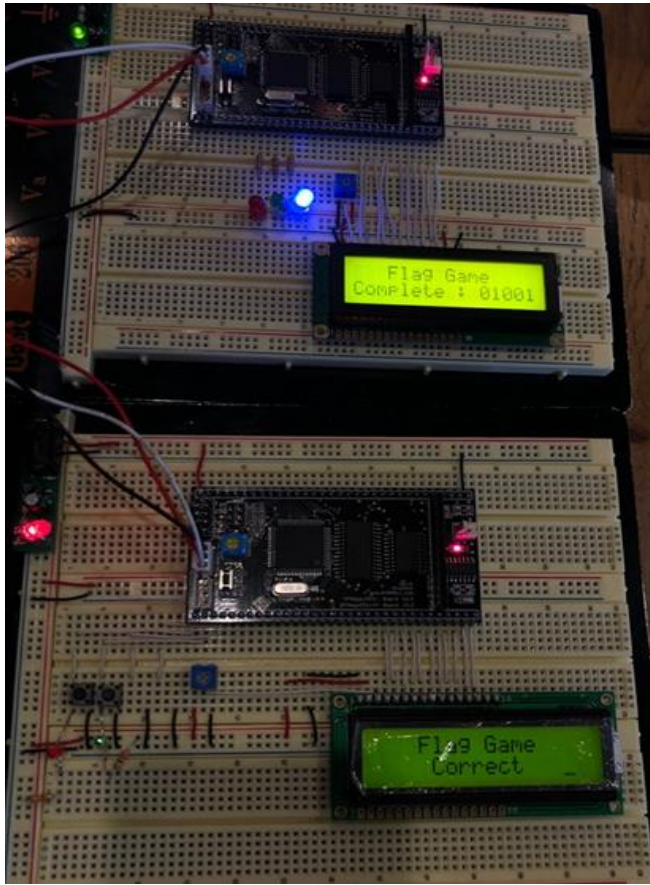
수신 완료 후 스위치(청기, 백기)를 눌러 문제를 푸는 모습



문제를 모두 풀 후, 제시한 답을 출력함



게임에서 이겼을 경우 : 모든 문제를 맞춘 경우



게임에서 졌을 경우 : 단 하나의 문제라도 틀린 경우

