

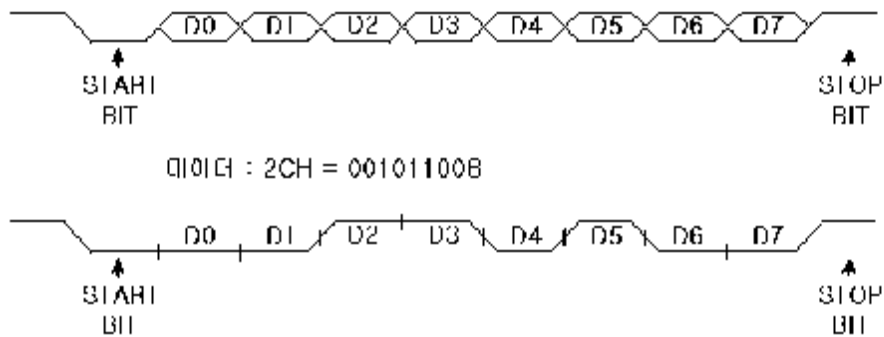
8. UART 직렬통신

학습 목표
1. 시리얼 통신의 개념을 이해한다.
2. AVR에서 시리얼 통신을 통해 컴퓨터와 데이터를 주고 받을 수 있다.
3. AVR의 UART를 이용해 PC와 데이터를 주고받을 수 있다. - UART와 관련된 레지스터를 학습한다. - 프로토콜에 대해 학습한다. - UART로 데이터를 주고 받는 방법을 학습한다.

8.1 직렬 통신의 개요

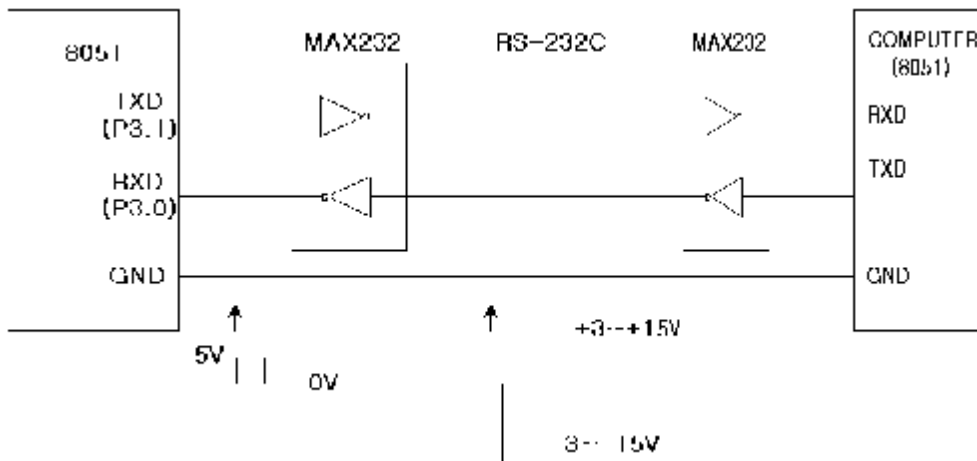
USART(Universal Synchronous and Asynchronous serial Receiver and Transmitter)는 동기식, 비동기식 직렬통신 장치이다. 동기식 통신은 고속 전송이 가능하지만 동기를 맞추기 위한 하드웨어가 필요하고 제어가 비동기식에 비해 어렵다. 비동기식 통신은 **RS-232** 통신방식을 사용한다. 보통 **UART**(비동기식 통신)라고 한다. **ATmega128**에는 2개의 시리얼 통신(**USART0**, **USART1**)이 있다. **USART0**와 **USART1**은 서로 다른 입출력(I/O) 레지스터를 갖고 있다.

8.1.1 시리얼 통신과 시리얼 통신 인터페이스 규격



Serial interface 통신 규격

- RS-232C, RS-422, RS-423, RS-485...



<RS232C 예>

Serial 통신 이점

- 적은 수의 통신 라인 사용과 먼 거리 전송

Serial 통신 전송 방식

- Simplex : 단방향 전송 방식 - 라디오
- half duplex : 반이중 전송 방식(서로 다른 시간 양방향 전송 방식)- 무전기
- full duplex : 전이중 전송 방식(동시 양방향 전송 방식) - 8051

속도

- bps(bits per second) : 1초당 전송되는 비트의 수
- baud rate : 1초당 전송되는 변조된 신호의 수

8.2 UART 통신 관련 레지스터와 사용법

이 장의 설명은 UART 통신에 Baudrate은 9600, 데이터 비트는 8비트, 스톱비트는 1비트, 패리티 비트는 사용 안 하고, 수신의 경우에만 인터럽트를 사용하는 것을 가정한다.

UART 사용 방법

◎ UART 설정

1. UBRRL과 UBRRH에 Baudrate을 설정한다.
 - $UBRR = F_CPU / (16 * Baudrate) - 1$
2. UCSRB의 RXCIE(수신 인터럽트 설정)와 RXEN(받기 사용), TXEN(보내기 사용)을 1로 설정한다.
3. UCSRC의 URSEL, UCSZ1, UCSZ0을 1로 설정한다. (반드시 UCSEL을 1로 하고 UCSRC에 값을 써넣어야 함)
 - 데이터 비트 8비트, 스톱 비트 1비트, 패리티 비트 없음 설정

◎ 한 바이트 전송하기

1. UCSRA의 UDRE가 1이 될 때까지 기다린다.
2. UCR에 보낼 데이터를 쓴다.

◎ 인터럽트로 전송된 한 바이트 읽기

1. ISR(USART_RX_vect)에서 UDR에서 전송된 한 바이트를 읽는다.

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR(Read)
	TXB[7:0]								UDR(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	

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
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	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	01	0	0	0	0	1	1	0	

Bit	7	6	5	4	3	2	1	0	
	URSEL	-	-	-	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
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	

UCSRnA(USARTn Control & Status Register A)

- BIT 7 : RXCn(USART Receive Complete)

RXCn(USARTn Receive Complete) 비트는 수신버퍼에 읽혀지지 않은 수신 문자가 들어있으면 "1"로 셋 되고 CPU가 이를 읽어서 수신 버퍼가 비어있는 상태라면 "0" 으로 클리어 되었음을 나타내는 상태 플래그이다. RXCn 비트는 수신 완료 인터럽트를 발생시킬 때 사용한다.

- BIT 6 : RxCn(USARTn Transmit Complete)

TXCn(USARTn Transmit Complete) 비트는 송신 시 포트 레지스터에 있는 송신 데이터가 모두 송신되고 UDRn의 송신 버퍼에 아직 새로운 송신 데이터가 라이트 되지 않은 상태 이면 "1"로 셋 되는 상태를 지시 하는 플래그이다. TXCn 비트는 송신 완료 인터럽트를 발생 시킬 때 사용된다

- BIT 5 : UDREn(USARTn Data Register Empty)

UDREn(USARTn Data Register Empty)비트는 UDRn의 송신 버퍼에 새로운 송신 데이터를 받을 준비가 되어 있으면 "1"로 셋 되는 상태 플래그이다. UDREn 비트는 UDARTn Data Register Empty 인터럽트를 발생할 때 사용한다.

- BIT 4 : FEn(Frame Error)

FEn(USARTn Frame Error)비트는 UDRn의 수신 버퍼에 현재 저장 되어있는 데이터를 수신 하는 동안 프레임 에러가 발생 하였음을 나타내는 상태 플래그로, 프레임에러는 수신 문자의 첫 번째 스톱비트가 "0"으로 검출 되면 발생(1로 셋 되고)하고 UCSRAn 레지스터를 라이트하면 이 비트는 "0"으로 클리어된다

- BIT 3 : DORn*Data OverRun

DORn(USARTn Data Overrun Error) 비트는 수신 동작에서 오버런 에러가 발생하였음을 나타내는 상태 플래그로 OverRun Error는 UDRn의 수신 버퍼에 현재 읽지 않은 수신 문자가 들어있는 상태에서, 수신 시 포트레지스터에 새로운 문자가 수신 완료되면, 다시 그 다음 수신 데이터인 3번째 문자의 스타트비트가 검출되면 발생한다. UCSRnA 레지스터를 라이트 하면 이 비트는 "0"으로 클리어된다.

- BIT 2 : PEn(Parity Error)

PEn(Parity Error)비트는 UDR의 수신 버퍼에 현재 저장 되어 있는 데이터를 수신 하는 동안 패리티에러가 발생 하였음을 나타내는 상태플래그로 패리티에러는 UCSRnC 레지스터의 UPMn1 비트를 "1" 설정하여 패리티 비트를 사용하도록 설정 한 경우에만 발생할 수 있다. UCSRnA 레지스터를 라이트 하면 이 비트는 "0"으로 클리어된다

- BIT 1 : UCXn(Double the USARTn Transmission Speed)

U2Xn(Double the USART Transmission Speed) 비트는 비동기모드에서만 사용가능 한 것으로서 클록의 n 분주비를 16에서 8로 1/2만큼 낮추어 전송 속도를 2배 높이는 기능을 한다

- BIT 0 : MPCMn(USART Multi-Processor Communication Mode)

비트는 USARTn을 멀티프로세서 통신 모드로 설정하고 멀티프로세서통신

어드레스정보를 포함 하지 않는 모든 수신 데이터는 수신부에 의하여 무시 된다.

UCSRnB

- BIT 7 : RXCIEn(RX Complete Interrupt Enable)

이 비트에 1을 쓰면 수신완료 인터럽트가 가능해 진다. 수신완료 인터럽트가 발생하려면 RXCIE 비트가 1이고, SREG 레지스터의 I-비트가 1이고, UCSRA 레지스터의 RXC 비트가 1이 되어야 한다.

- BIT 6 : TXCIEn(TX Complete Interrupt Enable)

이 비트에 1을 쓰면 송신완료 인터럽트가 가능해 진다. 송신완료 인터럽트가 발생하려면 TXCIE 비트가 1이고, SREG 레지스터의 I-비트가 1이고, UCSRA 레지스터의 TXC 비트가 1이 되어야 한다.

- BIT 5 : UDRIEn(USART Data Register Empty Interrupt Enable)

이 비트에 1을 쓰면 데이터 레지스터 빈(Empty) 인터럽트가 가능해진다. 데이터 레지스터 빈 인터럽트가 발생하려면 UDRIE 비트가 1이고, SREG 레지스터의 I-비트가 1이고, UCSRA 레지스터의 UDRE 비트가 1이 되어야 한다.

- BIT 4 : RXENn(RX Enable)

이 비트에 1을 써야만 수신 가능해 진다.

- BIT 3 : TXENn(TX Enable)

이 비트에 1을 써야만 송신이 가능해 진다.

- BIT 2 : UCSZn2(USART Character Size)

이 비트는 UCSRC 레지스터에 있는 UCSZ1:0과 결합해서 (3비트로) 데이터 길이(캐릭터 사이즈)를 결정한다.

- BIT 1 : RXB8n(RX data Bit 8)

데이터 수신 9번째 비트로 멀티 프로세서 직렬통신 시 사용한다.

- BIT 0 : TXB8n(TX data Bit 8)

데이터 송신 9번째 비트로 멀티 프로세서 직렬통신 시 사용한다.

UCSRnC(USARTTn Control & Status Register C)

- BIT 7 - 예약

- BIT 6

UMSELn(USARTn Mode Select) UMSELn(USARTn Mode Select) 비트는 "1"이면 USARTn 모듈을 동기 전송 모드로 설정 하고 "0"이면 비동기 전송 모드로

설정한다

- BIT5,4 : UPMn1, 0(Parity Mode)

이 비트를 "1"로 설정 하면 패리티를 발생 시키고 검사를 할 수 있고 송신기는 자동적으로 각 프레임의 송신 데이터에 패리티 비트를 더하여 송신한다. 수신기는 UPM0 비트와 수신된 데이터를 비교한다. 만약에 오류가 발생하면 UCSRnA 레지스터의 PE 플래그가 "1"로 셋된다

- BIT 3 : USBn(Stop Bit Select)

USBn(Stop Bit Select) 비트가 "0" 이면 USARTn 모듈에서 데이터 포맷을 구성하는 스톱 비트를 1개로 설정하고 "1"이면 스톱 비트를 2개로 설정한다.

- BIT 2,1 : UCSZn1,0(Character Size) - 데이터비트수를설정
- BIT 0 : UCPOLn

이 비트는 동기식 통신 모드에서만 사용한다. 비동기식 통신 모드에서는 0 을 써 넣는다.

UMSELn 비트 셋팅

UMSELn	모드
0	비동기 모드
1	동기 모드

UPMn 비트 셋팅

UPMn1	UPMn0	패리티 모드
0	0	패리티 체크 기능을 사용하지 않음
0	1	reserved
1	0	짝수 패리티 방식을 사용
1	1	홀수 패리티 방식을 사용

UCSZn 비트 셋팅

UCSZn2	UCSZn1	UCSZn0	전송 데이터 길이
0	0	0	5비트
0	0	1	6비트
0	1	0	7비트
0	1	1	8비트
1	0	0	reserved
1	0	1	reserved
1	1	0	reserved
1	1	1	9비트

USBSn 비트 셋팅

USBSn	스톱 비트
0	1비트
1	2비트

UBRR0H = 0x00;

- UBRnH/L(USARTTn Baud Rate Register) 레지스터는 16비트 중에서 12비트만 사용하여 USARTTn 모듈의 송·수신 속도를 설정하는 기능을 한다.

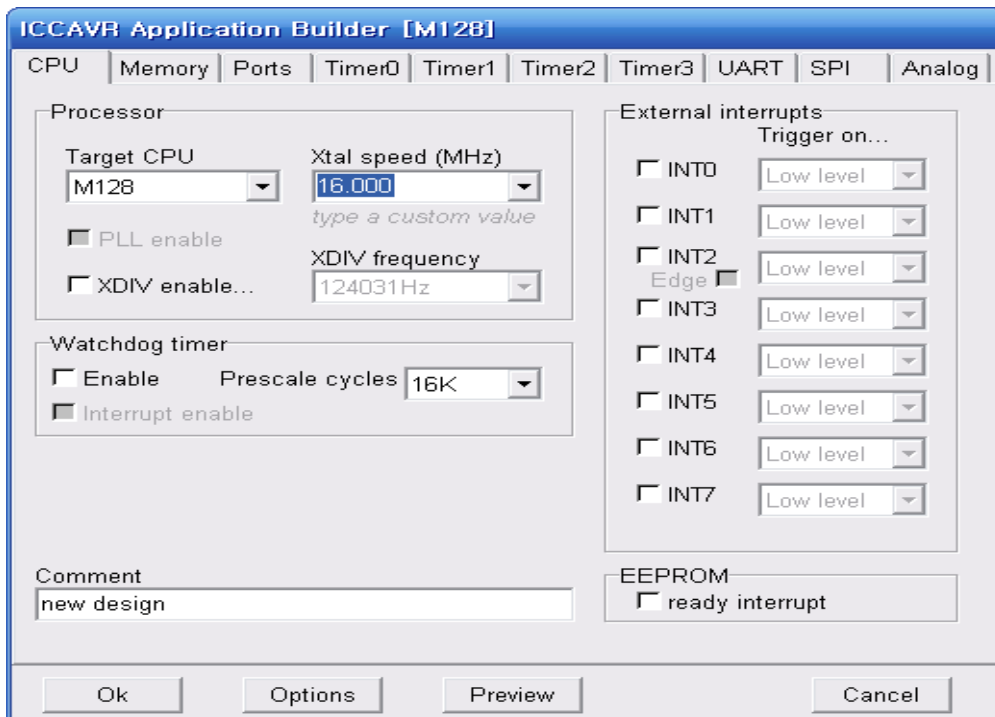
- BIT 15~23 : 예약비트

- BIT 11~0 : UBRRn11~0(USARTn Baud Rate Register)

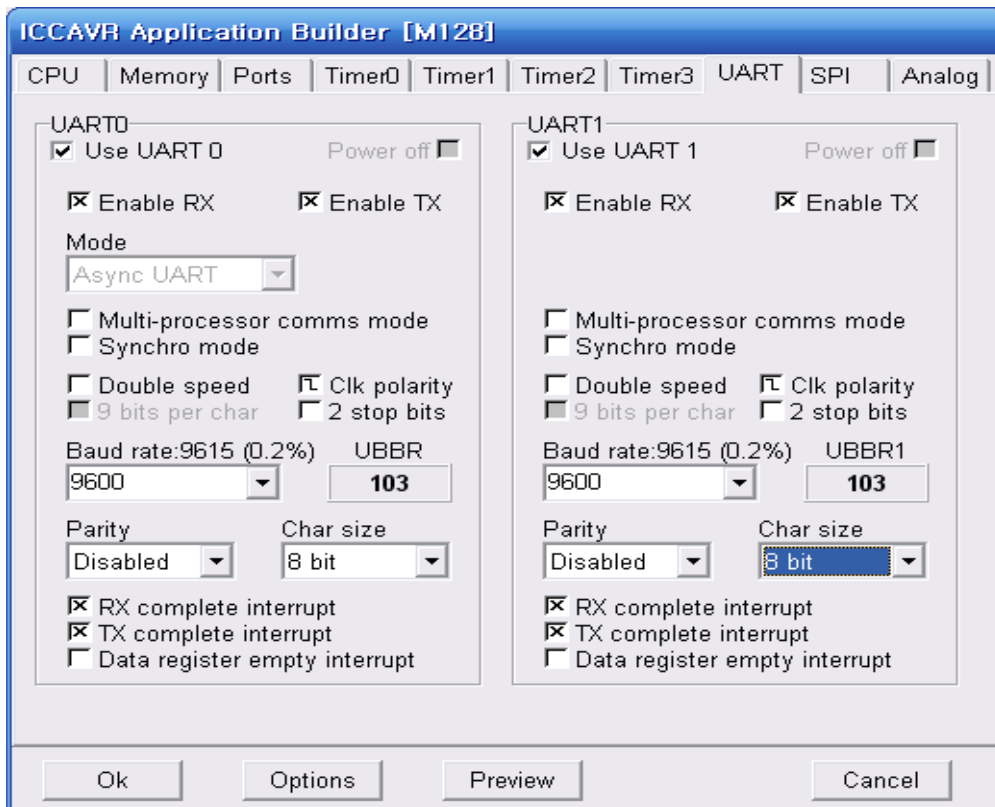
- 2비트를 이용하여 USARTn의 보율(baud rate)을 결정 하는데 UBRRnH의 4비트와 UBRRnL의 8비트가 조합을 이루고 있다

8.3 시리얼 통신 만들기 - ICCAVR

ICCAVR을 실행시켜 프로젝트를 만든후, Tools >> Application Builder를 Click



Target CPU 에 M128, Xtal speed에 16.000을 선택하고 UART탭을 클릭하여 다음 그림과 같이 셋팅한다.

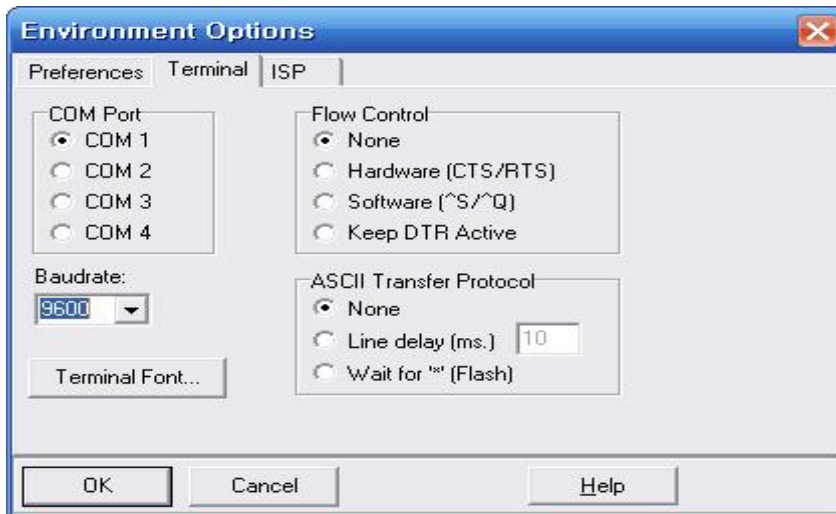


위와 같이 셋팅이 끝났으면 Options에서 Include "main()"을 클릭한후 OK 버튼을 클릭한다. 그러면 UART를 사용하기 위한 기본 셋팅이 나온다.

```
★ #pragma interrupt_handler uart0_rx_isr:19
void uart0_rx_isr(void)
{
    //uart has received a character in UDR
    UDR0 = UDR0;
} // 컴파일 후 ATmega128에 포팅한다.

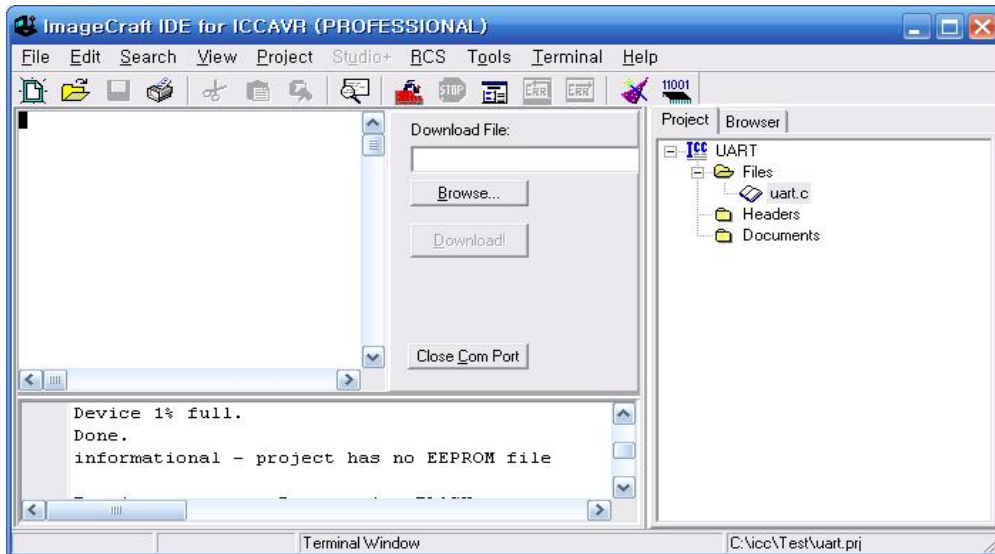
void main() {
    init_devices();
    printf("Serial Forwarder v0.1 \r \n");
    while(1) { ;
        }
    } // main문에 위의 내용을 추가한후.
```

Tools >> Environment Options를 Click 한 후 아래와 같이 COM Port를 COM1, Baudrate를 9600으로 맞춘후 OK Click.

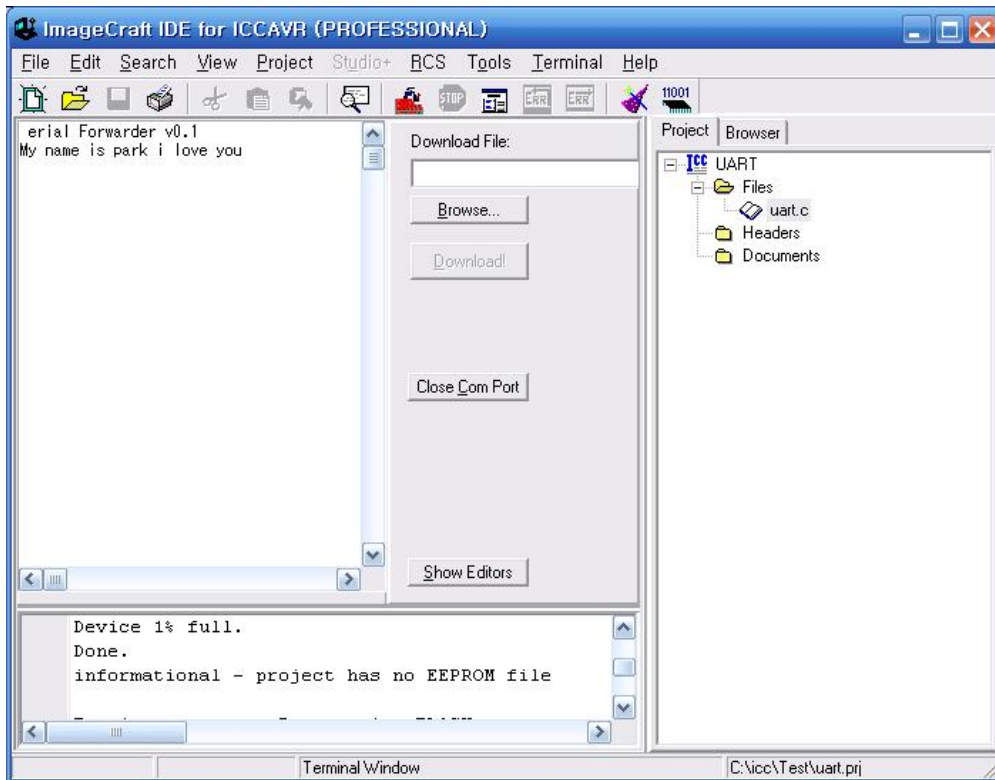


파일을 저장시켜 프로젝트에 추가(ADD)한후 Project >> Make Project 또는 F9키를 눌러 라이트 프로그램한다.

Terminal >> Show Terminal Window 를 Click 하면 아래의 창이 뜬다.



키보드를 눌러도 아무런 변화가 없을 것이다. 오른쪽 창에 Open Com Port 버튼을 누른다.



그러면 키보드로 치는 것들이 화면에 뜰 것이다. UART 성공....
그러면 다음은 실습을 통해 좀 다양한 UART 통신을 활용해 보자.

8.4 UART 심화 실습

UART를 이용한 통신 실습 - ICCAVR

```
#include <iom128v.h>
#include <macros.h>
//UART0 initialize // desired baud rate: 9600
// actual: baud rate:9615 (0.2%)
// char size: 8 bit
// parity: Disabled
```

UART을 이용한 통신 실습 계속 - **ICCAVR**

```

void uart0_init(void)
{
    UCSRB = 0x00; //disable while setting baud rate
    UCSRA = 0x00;
    UCSRC = 0x06;
    UBRR0L = 0x67; //set baud rate lo
    UBRR0H = 0x00; //set baud rate hi
    UCSRB = 0xF8;
}
void main(void){
    char in;
    init_devices();
    while(1){

        printf("\r\nPress Character A~Z");
        scanf("%c", &in);
        switch(in){
            case 'A' : printf("\r\nPressed A");
                        break;
            case 'B' : printf("\r\nPressed B");
                        break;
            case 'C' : printf("\r\nPressed C");
                        break;
            case 'D' : printf("\r\nPressed D");
                        break;
            case 'E' : printf("\r\nPressed E");
                        break;
            default  : printf("\r\nPressed F~Z or other");
                        break;
        }
    }
}

```

UART을 이용한 통신 실습 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
void uart0_init(void)
{
    UCSR0B = 0x00; //disable while setting baud rate
    UCSR0A = 0x00;
    UCSR0C = 0x06;
    UBRR0L = 0x67; //set baud rate lo
    UBRR0H = 0x00; //set baud rate hi
    UCSR0B = 0x18;
}
void init_devices(void)
{
    //stop errant interrupts until set up
    CLI(); //disable all interrupts
    XDIV = 0x00; //xtal divider
    XMCRA = 0x00; //external memory
    uart0_init();
    MCUCR = 0x00;
    EICRA = 0x00; //extended ext ints
    EICRB = 0x00; //extended ext ints
    EIMSK = 0x00;
    TIMSK = 0x00; //timer interrupt sources
    ETIMSK = 0x00; //extended timer interrupt sources
    SEI(); //re-enable interrupts
}
char getch(void)
{
    while (!(UCSR0A & 0x80)); // UCSR0A 레지스터의 RXC0 레지스터가
        return UDR0;          // 1이면 즉, 읽지 않은 데이터가 수신버퍼에
                                //있으면, 데이터 수신
}

```

UART을 이용한 통신 실습 계속 - ICCAVR

```

int putch(char data)
{
    while (!(UCSR0A & 0x20)) ; //UCSR0A 레지스터의 UDRE0 레지스터가
        UDR0 = data;          //1이면 즉, 버퍼가 비어 있으면 데이터 송신
    }
void main(void)
{
    char rx;
    init_devices();
    putch('A'); // 'A'를 송신(테스터용)

    while(1){ //키보드로 입력(수신)된 데이터를 터미널에 출력(송신)
        rx = getch();
        putch(rx);
    }
}

```

-이로써 UART 직렬통신을 마무리하겠습니다. 직렬통신은 아주 광범위하게 응용되는 분야이기에 직렬통신을 이용한 많은 프로그램을 짜보는 것이 왕도라 생각합니다. 공부하시다가 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다.

9. DC MOTOR 실습

9.1 DC Motor의 개요

- 고정자로 영구자석을 사용
- 회전자(전기자)로 코일을 사용하여 구성한 것
- 전기자에 흐르는 전류의 방향을 전환함으로써 자력의 반발
- 흡인력으로 회전력을 생성시키는 모터
- 모형 자동차, 무선조정용 장난감 등을 비롯하여 여러 - 방면에서 가장 널리 사용되고 있는 모터

9.1.1 DC 모터의 특성

- 기동 토크가 크다
- 인가전압에 대하여 회전특성이 직선적으로 비례한다
- 입력전류에 대하여 출력 토크가 직선적으로 비례하며, 또한 출력 효율이 양호하다
- 가격이 저렴하다

9.1.2 제어성의 장점과 실제 특성

- T-I 특성(토크 대 전류)
 흘린 전류에 대해 깨끗하게 직선적으로 토크가 비례한다.
 큰 힘이 필요한 때는 전류를 많이 흘리면 되는 것이다.
- T-N 특성(토크 대 회전수)
 토크에 대하여 회전수는 직선적으로 반비례한다.
 이것에 의하면 무거운 것을 돌릴 때는 천천히 회전시키게 되고, 이것을 빨리 회전시키기 위해서는 전류를 많이 흘리게 된다.
- 인가전압에 대해서도 비례한다.

9.1.3 DC 모터의 결점

- DC 모터의 가장 큰 결점으로는 그 구조상 브러시(bush)와 정류자(commutator)에 의한 기계식 접점이 있다는 점이다.
- 이것에 의한 영향은 전류(轉流)시의 전기불꽃(spark), 회전 소음, 수명이라는 형태로 나타난다.
- 마이크로컴퓨터 제어를 하려는 경우는 "노이즈"가 발생하게 된다. 따라서 이 노이즈 대책이 유일한 과제가 될 수 있다.

- 이 노이즈 대책을 위해서는 아랫 그림과 같이 각 단자와 케이스 사이에 $0.01\mu\text{F} \sim 0.1\mu\text{F}$ 정도의 세라믹 콘덴서를 직접 부착한다. 이것으로 정류자에서 발생하는 전기불꽃을 흡수하여 노이즈를 억제할 수 있다.

- DC 모터의 노이즈 대책에는 콘덴서를 케이스와 단자간에 직접 부착한다. 콘덴서의 리드는 가급적 짧게 한다

9.2 DC 모터의 가변속 제어법

9.2.1 아날로그 방식의 가변속 제어

- 직접 구동전압 그 자체를 변화시키는 것으로, 트랜지스터로 전압 Dropper를 구성하고 컬렉터 이미터간의 드롭 전압을 바꿈으로써 모터에 가해지는 구동전압을 가변으로 한다.

- 기본원리에 의해 드롭퍼 전압이 그대로 열로 되어 손실로 되며, 특히 저속으로 할 때 전력 사용 효율이 나빠지고 만다.

- 손실로 인해 발생하는 열대책을 위해, 큰 방 열판을 필요로 하기 때문에 전체가 대형으로 되고 만다.

- 소형 모터이고 게다가 속도의 가변폭이 작아도 좋은 경우에는 손실을 작게 할 수 있다는 점과 제어회로가 간단하기 때문에 흔히 사용되고 있다.

9.2.2 펄스폭 변조(PWM) 방식

- PWM 방식은 결과적으로는 구동전압을 바꾸고 있는 것과 같은 효과를 내고 있지만, 그 방법이 펄스폭에 따르고 있으므로 펄스폭 변조(PWM:Puls-e Width Modulation)라 부르고 있다.

- 구체적으로는 모터 구동전원을 일정 주기로 On/Off 하는 펄스 형상으로 하고, 그 펄스의 Duty비(On 시간과 Off 시간의 비)를 바꿈으로써 실현하고 있다.

- DC 모터가 빠른 주파수의 변화에는 기계 반응을 하지 않는다는 것을 이용하고 있다. 트랜지스터를 일정시간 간격으로 On/Off 하면 구동전원이 On/Off 되는 것이다.

- 전압을 생각하면 외관상 구동전압이 변화하고 있는 것이다. 여기서 중요한 기능을 담당하고 있는 것이 위의 회로도에 있는 다이오드이며, 일반적인 전원용 다이오드를 사용하지만 그 동작 기능에 의해 flywheel diode라 부르고 있다. 즉, 트랜지스터가 Off로 되어 있는 동안 모터의 코일에 축적된 에너지를 전류로 흘리는 작용을 한다(회생전류라 부른다).

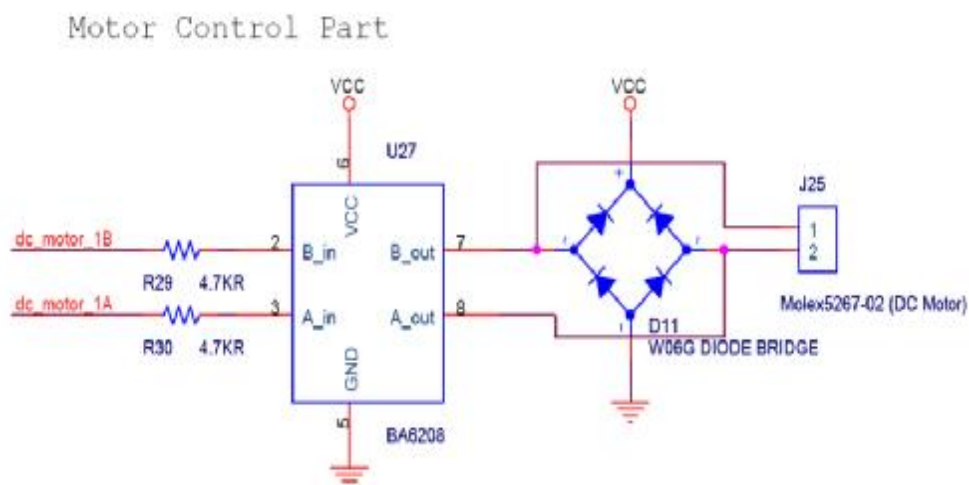
9.3 DC 모터의 위치 및 속도 제어

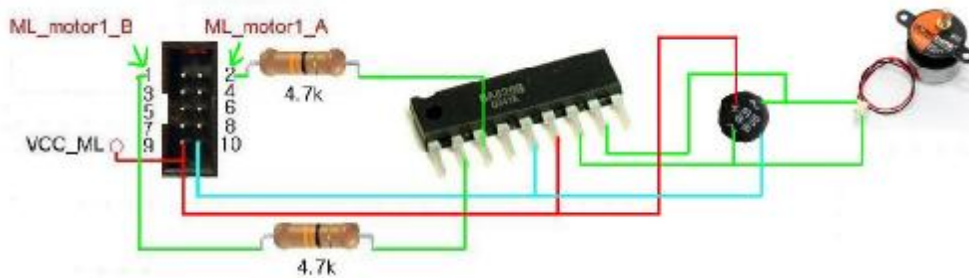
9.3.1 DC 모터 제어기술

- DC 모터 제어란 모터의 속도 및 위치를 원하는 대로 조정하는 것을 의미한다.
- DC 모터 제어를 위해서는 모터 축 또는 기어 축의 위치(회전 각도)를 측정할 수 있는 엔코더가 부착되어 있어야 한다.
- 제어기는 이 엔코더로부터 들어오는 정보로부터 모터의 속도 및 회전각을 계산하고 이를 바탕으로 모터에 가해지는 전압을 조절하게 된다.
- 모터 제어부는 그림 8과 같이 엔코더의 펄스를 세어 12비트 또는 16비트의 펄스 수로 바꾸어 주는 계수기(Counter)와 이 계수기(Counter)값으로 속도 및 위치를 계산하여 모터에 가할 전압을 결정하는 논리회로 그리고 모터로 가하는 전압에 충분한 전류를 공급하여 모터가 큰 토크로 구동할 수 있도록 해주는 driver로 구성된다.
- 제어부를 구성하는 방법은 크게 두 가지로 첫 번째는 각각의 부품을 따로 따로 구입하여 하나의 제어부를 구성하는 방법과 제어부 전체를 담당하는 하나의 칩을 구입하여 구성하는 방법이다.

9.3.2 엔코더

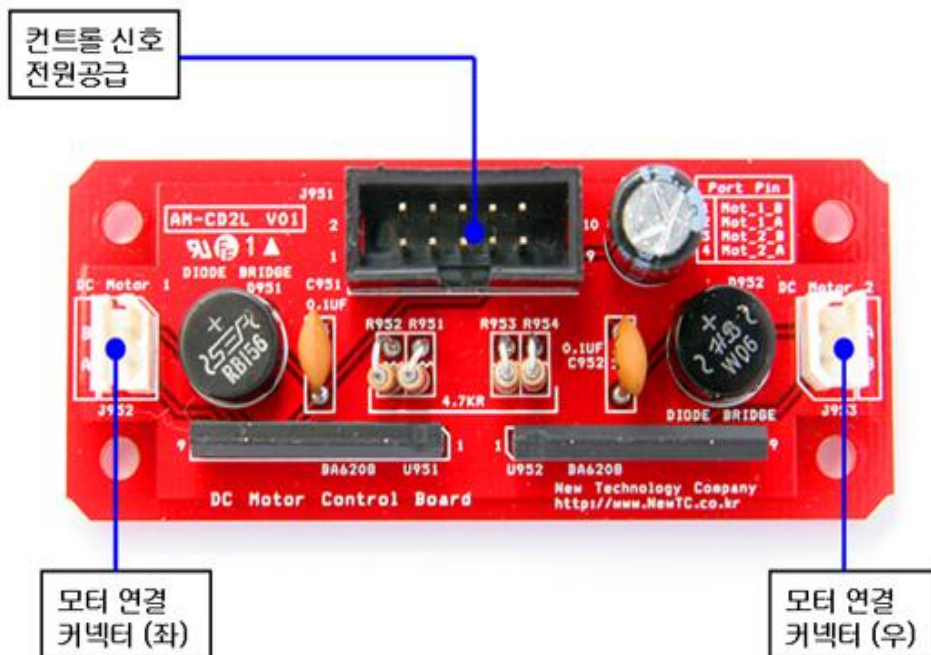
- 엔코더는 회전 위치 및 속도를 검출하는 목적으로 서보모터를 사용한 고속회전 위치 제어에 사용된다. 엔코더는 수십khz 정도까지 응답하는 고속성과 비접촉 동작에 의한 높은 신뢰성을 가지고 있다. 로터리 엔코더에는 절대값형(Absoulte)과 증분형(Incremental)이 있다.



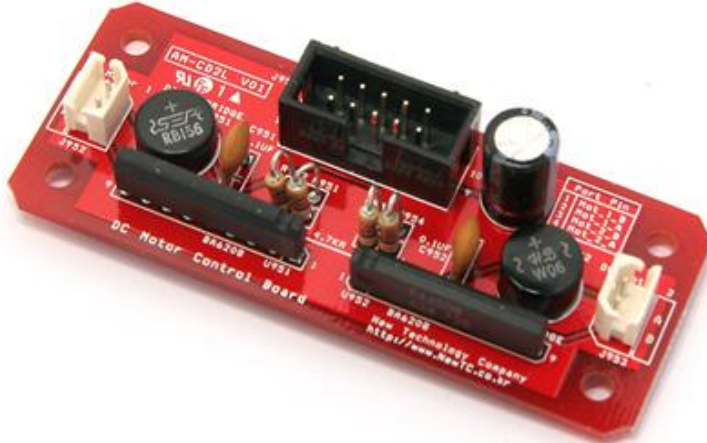


9.4 소형 DC 모터 구동보드 (Model : AM-CD2L)

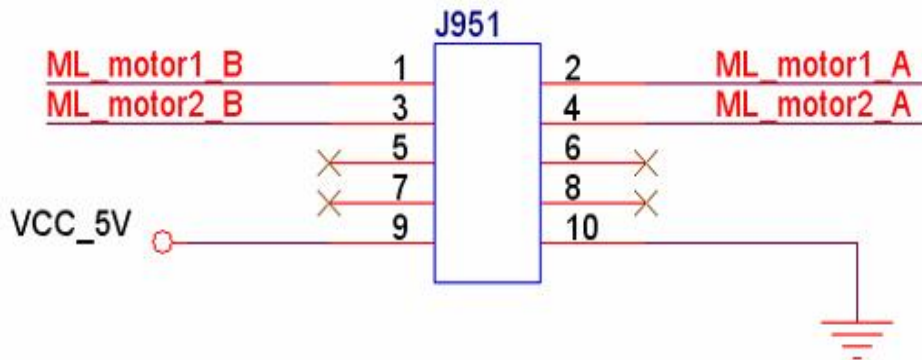
9.4.1 소형 DC 모터 구동보드 소개



- ◆ 소형 DC 모터 구동보드 (모터 1개당 최대 허용 전류 100mA)
- ◆ 소형 DC 모터 2개 구동
- ◆ 10Pin Cable 로 컨트롤 신호와 구동 전원을 공급 받음
- ◆ 정방향, 역방향으로 구동 가능



소형 DC 모터 구동보드 AM-CD2L



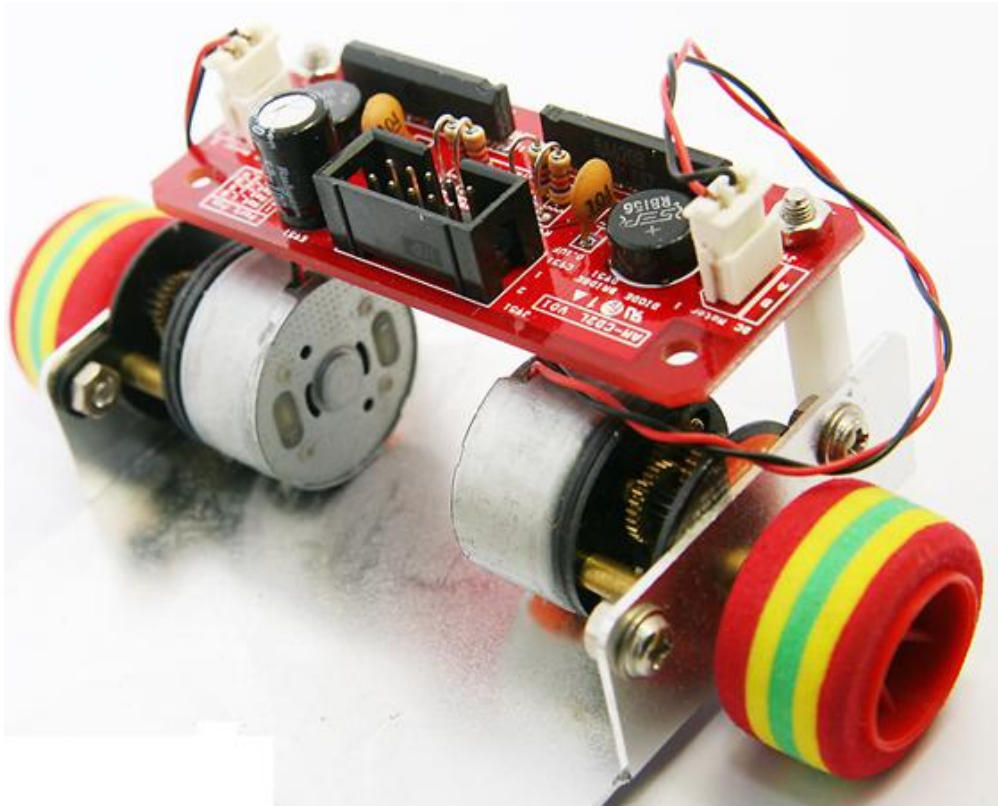
1 번	3 번	5 번	7 번	9 번
왼쪽 모터 B상	오른쪽 모터 B상			전원 5V
2 번	4 번	6 번	8 번	10 번
왼쪽 모터 A상	오른쪽 모터 A상			전원 GND

AM-CD2L DC 모터 구동 모듈 사용하기

모터 제어 신호인 A상과 B상을 마이크로 컨트롤러의 포트 입력과 연결한다.

	CCW	CW	Stop	Stop
A상	1	0	1	0
B상	0	1	1	0

모터 제어를 위한 입력신호는 위와 같다. A상에 1, B상에 0의 입력을 인가하면 모터는 CCW 로 돌게 된다. 반대로 A상에 0, B상에 1의 입력을 인가하면 모터는 CW 로 돌게 된다. 모터를 멈추기 위해서는 A상과 B상에 같은 입력을 넣으면 된다.



DC Motor 제어 실습 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
#define MOTORPORT1 PORTC
#define ON          1
#define OFF         0
#define CW          1
#define CCW         0
void port_init(void)
{
    DDRC = 0xFF; //모터 포트를 출력으로 설정
}
// Motor Control
void M1A(int onoff){ //왼쪽 모터 B상
    if(onoff==ON)
        MOTORPORT1 = MOTORPORT1|0x01;
    else
        MOTORPORT1 = MOTORPORT1&0xFE;
}
void M1A_(int onoff){ //왼쪽 모터 A상
    if(onoff==ON)
        MOTORPORT1 = MOTORPORT1|0x02;
    else
        MOTORPORT1 = MOTORPORT1&0xFD;
}
void M1B(int onoff){ //오른쪽 모터 B상
    if(onoff==ON)
        MOTORPORT1 = MOTORPORT1|0x04;
    else
        MOTORPORT1 = MOTORPORT1&0xFB;
}

```

DC Motor 제어 실습 계속 - ICCAVR

```

void M1B_(int onoff){ //오른쪽 모터 A상
    if(onoff==ON)
        MOTORPORT1 = MOTORPORT1|0x08;
    else
        MOTORPORT1 = MOTORPORT1&0xF7;
}

void Motor1(int CWCCW){
    if(CWCCW==CW){
        M1A(ON);
        M1A_(OFF);
    }else{
        M1A(OFF);
        M1A_(ON);
    }
}

void Motor2(int CWCCW){
    if(CWCCW==CW){
        M1B(ON);
        M1B_(OFF);
    }else{
        M1B(OFF);
        M1B_(ON);
    }
}

void delay(int delaytime){
    int i,j;
    for(i=0;i<1000;i++){
        for(j=0;j<delaytime;j++){
        }
    }
}

```

DC Motor 제어 실습 계속 - ICCAVR

```

void main(void)
{
    init_devices();
    //insert your functional code here...
    printf("TEST for DC motor control");

    while(1){
        printf("\r\nClockWise");
        Motor1(0);
        Motor2(0);
        delay(1000);
        printf("\r\nCounter ClockWise");
        Motor1(1);
        Motor2(1);
        delay(1000);
    }
}

```

이상으로 DC모터에 관한 기초 사항들을 마무리 하겠습니다. 보다 많은 소스는 <http://www.NTC.co.kr>에 오시면 Q&A에 등록되어 있습니다. 많은 소스를 접하다보면 자신도 모르게 발전하는 자신을 느낄수 있을것 같습니다. 공부하시다가 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다.

10. STEPPING MOTOR

10.1 스테핑 모터란?

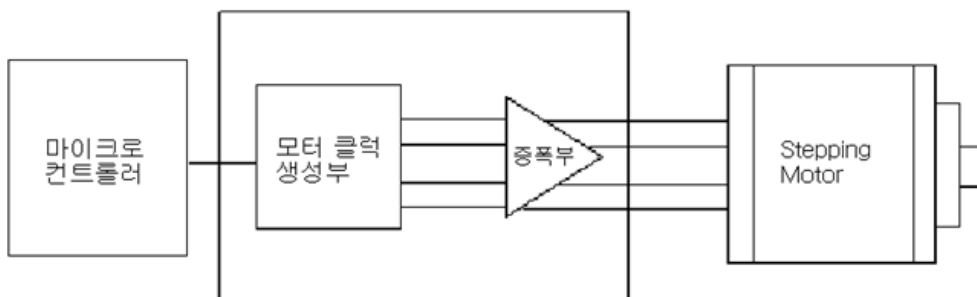
- 스테핑 모터란 Step 에 의해 구동되는 모터이다. 4개의 모터상이 있어서 한 스텝 한 스텝 진행하면서 모터가 회전하는 것이다. 한 스텝 당 보통 1.8° 씩 회전한다.
- 스테핑 모터는 AC servo, DC servo 모터에 비하여 값이 싸고 정확한 각도 제어에 유리하여 우리 주위에서 쉽게 접할 수 있다.
- 스테핑 모터는 pulse에 의해 digital적으로 제어하는 것이 가능하므로 마이크로 컨트롤러에서 사용하기에 적합한 모터이다.
- 스테핑 모터는 다른 모터와 달리 모터의 위치를 검출하기 위한 feedback없이, 정해진 각도를 회전하고, 상당히 높은 정확도로 정지할 수 있다.

10.2 스테핑 모터 구동하기

- DC모터는 회로는 간단하지만 제어하는 방법이 복잡하다. 반면 스테핑 모터는 구동회로는 상대적으로 복잡하지만 제어는 훨씬 쉽다.
- 스테핑 모터를 구동하기 위해서는 L297과 SLA7024를 같이 사용한다. L297은 스테핑 모터 구동에 필요한 상(클럭)을 만들어 주기 위한 것이고 SLA7024는 만들어진 상(클럭)의 전압과 전류를 증폭시키는 역할을 합니다

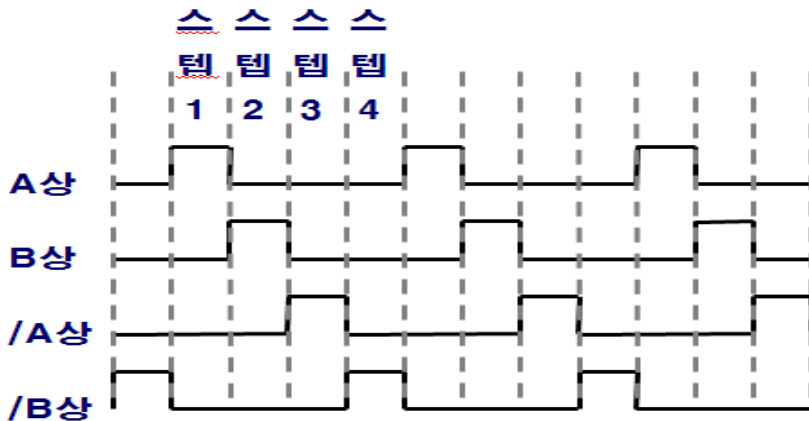
10.3 스테핑 모터의 구동부 구성

- 모터클럭 생성부 : Stepping motor의 여자 상(Step)을 만든다. (소프트웨어)
- 증폭부 : 모터 입력 신호를 증폭하여 Stepping motor를 구동한다.



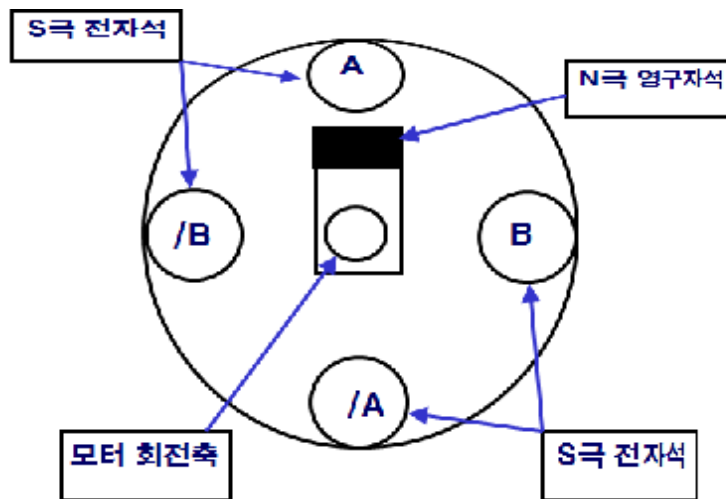
10.4 구동방식의 종류

- 4상 motor 의 1상 여자 동작



위 그림은 1상 여자 방식의 motor의 동작을 위한 신호를 나타낸다

10.4.1 스테핑 모터의 기본 구조



- 스테핑 모터의 기본 구조는 위 그림과 같다. 위와 같은 A, B, /A, /B의 전자석이 연속해서 배치 되어 회전축이 더 정밀하게 회전을 할 수 있다.

- A, B, /A, /B 4개의 전자석중 하나에 전류가 흘러 S 극 전자석으로 동작하면 모터 회전축에 연결되어 있는 N극 영구자석이 전류가 흐르는 전자석 방향으로 회전을 하게 되는데 이러한 구조를 이용하여 모터를 회전 하도록 신호를 인가한다.

- A, B, /A, /B 상에 차례로 전류가 흐르게 하면 다음 그림과 같이 모터축이 회전을 하게 된다.

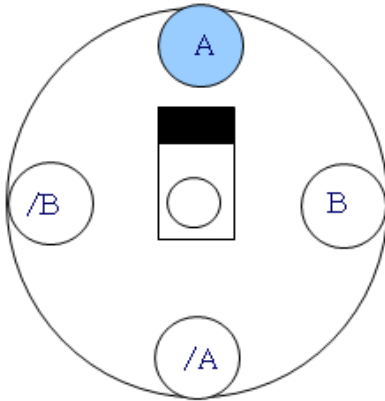


그림 1.
A상만 on되고 나머지는 off

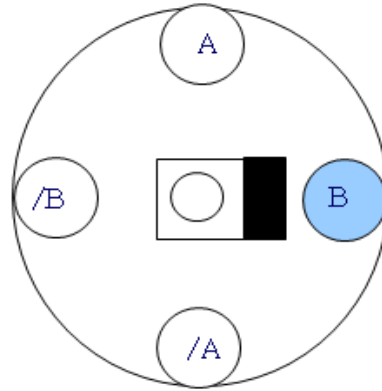


그림 2.
B상만 on되고 나머지는 off(90도회전)

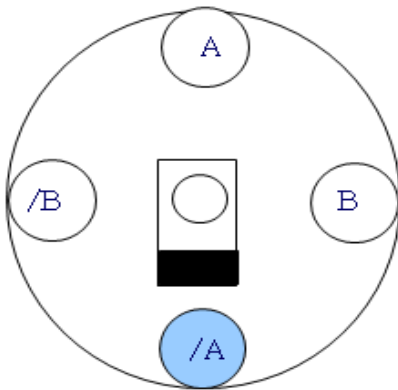


그림 3
/A상만 on되고 나머지는 off(90도회전)

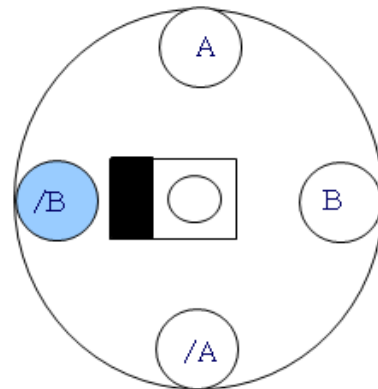


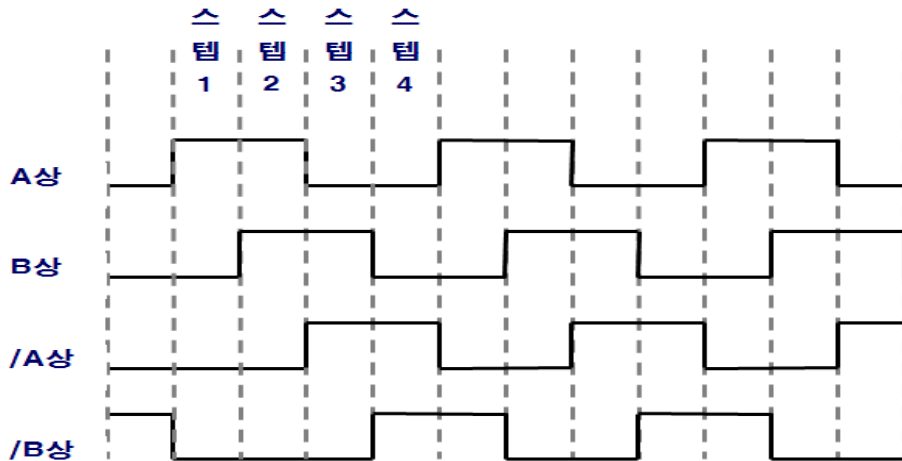
그림 4

Step 1 A상에 연결된 TR이 On 이 되어 A->Com로 전류가 흐른다. coil에 전류가 흐름으로써 고정자의 상은 N,S극으로 여자된다. 이때 B 쪽에는 전류가 흐르고 있지 않으므로 B 쪽의 상은 비여자이지만, A쪽의 여자에 동반해서 회전자들의 영구자석은 각각 N 과 S , S 와 N 이 결합해서 안정한 위치에 정지한다.

다음에 step 2로 진행하면 먼저 on 하고 있던 A상에 연결된 TR 은 off가 되고, 대신에 B상에 연결된 TR 가 on 이된다. B상에 연결된 TR이 on이 되면 B ->

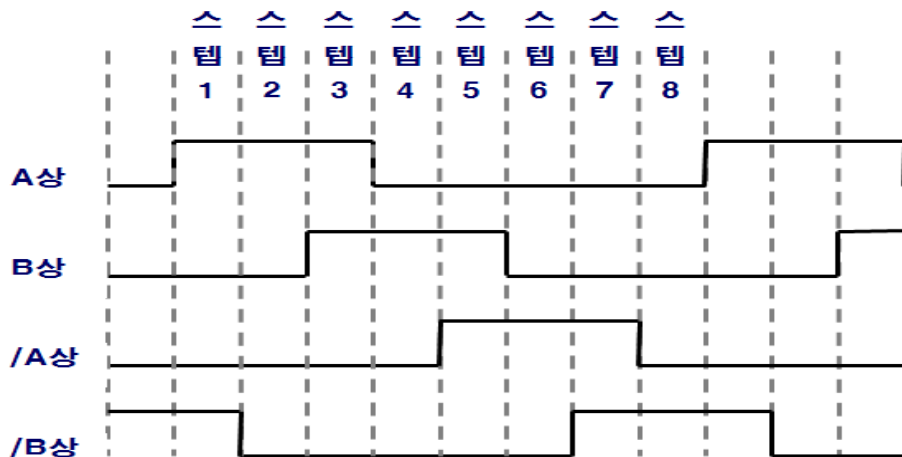
Com의 coil에 전류가 흘러 이번에는 90°씩 어긋나고 있는 고정자가 여자가 된다. 그리고 여자위치가 이동한 것으로 회전자도 시계 방향으로 당겨져 90° 회전하게 된다. 같은 방법으로 step 3 과 step 4 의 동작을 함으로써 모터는 각 step 당 90°씩 진행시켜 회전시킬 수 있다.

- 4상 motor 의 2상 여자 동작



기본적인 동작은 1상 여자 방식과 동일하다. 차이점은 토크를 좋게 하기 위해 2개의 코일을 동시에 동작시킨다는 것이다. 1상 여자에 비하면 전류가 많이 흐르는 단점이 있지만 정지상의 오버슈터나 언더슈터가 작고 과도 특성이 좋아진다.

- 4상 motor 의 1-2 상 여자 동작

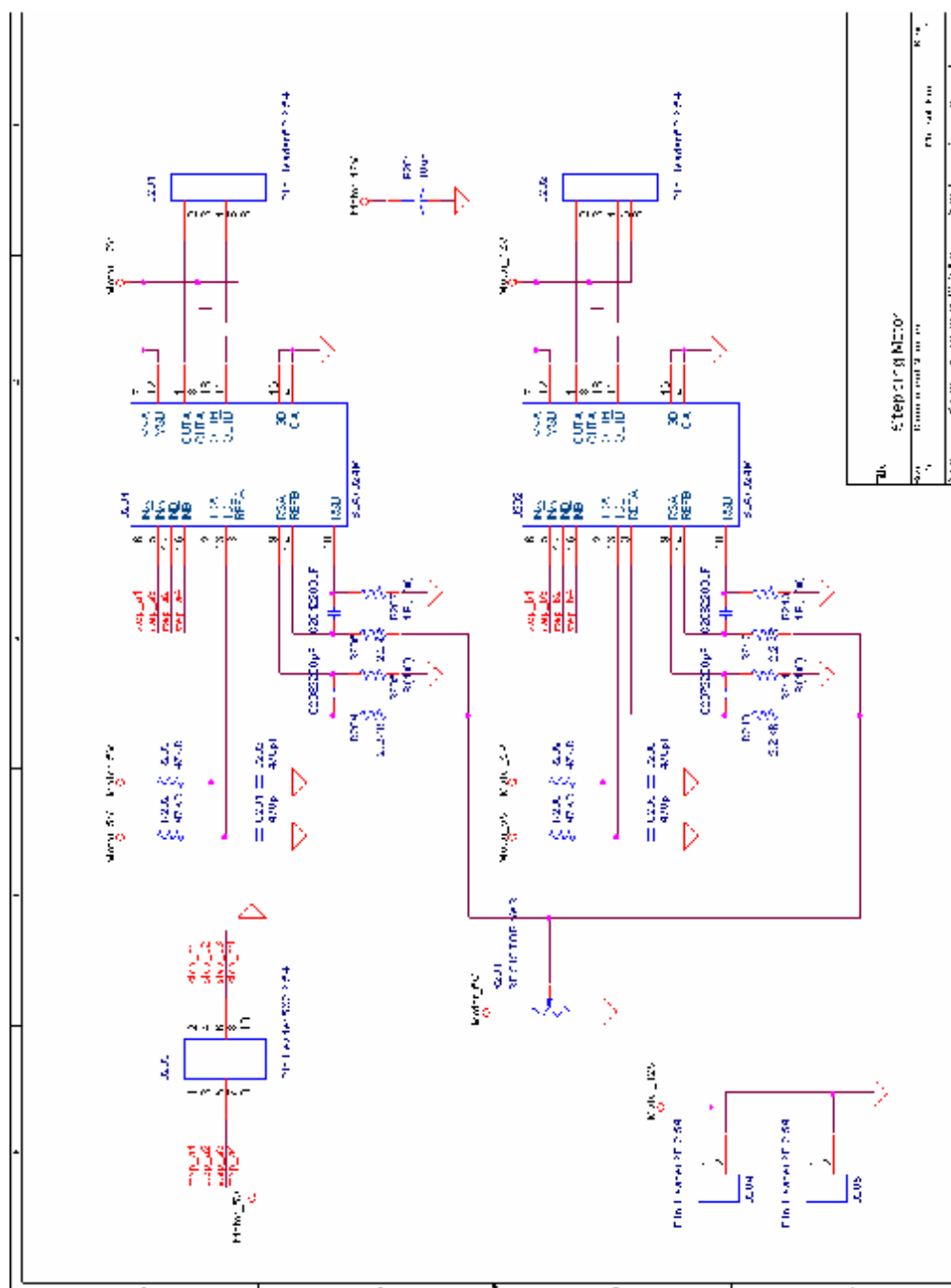


위 그림의 1-2 상 여자는 전술한 1상 여자와 2상 여자가 교대로 반복하는 것이다. 따라서 회전자는 step 마다 45° 회전한다. 즉 step 각은 maker 가 표시하는 각도의 1/2 가 된다. 1-2상 여자는 1상 여자와 2상 여자의 특성을 갖고 있으므로 step rate는 배가된다. Step 의 각도가 다른 동작법에 비해 1/2 만큼 작기 때문에 3가지 방식 중 가장 성능이 좋다. 단점은 다른 여자방식과 같은 속도를 내려면 마이크로 컨트롤러에서 클럭을 2배로 만들어야 한다는 것이다.

10.5 구동 함수

```
// step      : 모터의 현재 출력값
// next: 모터의 다음 출력 할 값
char nextMotorCLK_1 (char step)
{
    char next;
    switch(step) {
        case 0x01 : next = 0x02; break;
        case 0x02 : next = 0x04; break;
        case 0x04 : next = 0x08; break;
        case 0x08 : next = 0x01; break;
        default  : next = 0x01; break;
    }
    return next;
}

void main (void)
{
    init_devices();
    char step = 0;
    while(1) {
        step = nextMotorCLK_1 (step);    // 위의 함수를 호출
        PORTD = step;    // PORTD 에 출력 신호가 연결 되어있다고 가정한다.
        Delay(100);
    }
}
```



- 회로 설명

J203 은 모터 제어신호가 입력되는 커넥터이다. 마이크로 컨트롤러에서 생성

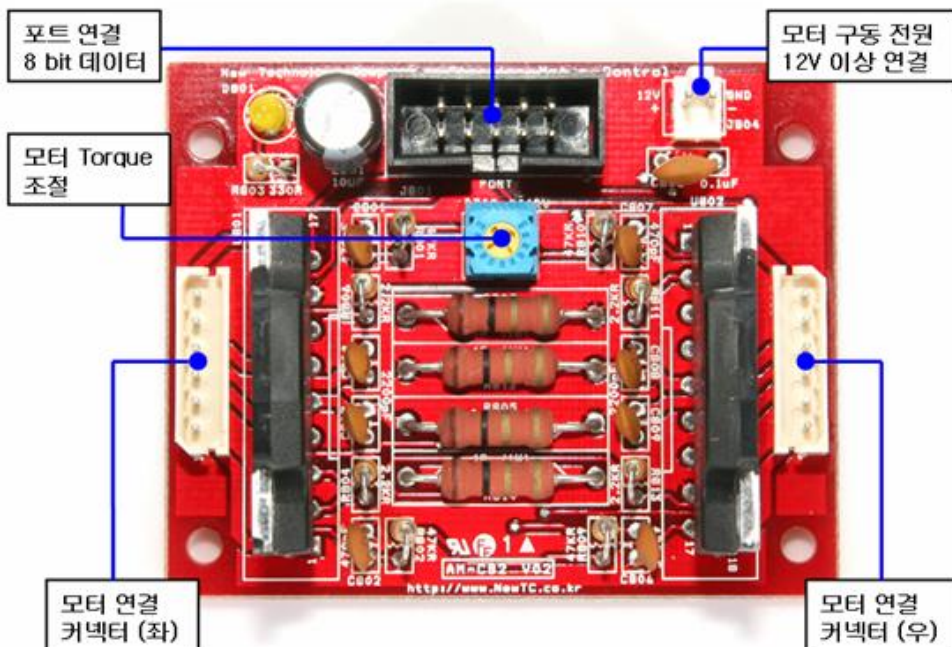
한 모터 제어신호가 들어온다. 제어신호는 U201, U202 SLA7024(7026)으로 입력되어 이 칩에서 모터 동작을 위한 전압으로 증폭된다. 증폭된 신호는 모터 연결 커넥터인 J201과 J202 로 연결되어 모터로 전달된다.

R201 의 가변 저항은 모터의 전류를 조절해 주는 가변 저항이다. 적당히 돌려 모터의 토크를 결정할 수 있다.

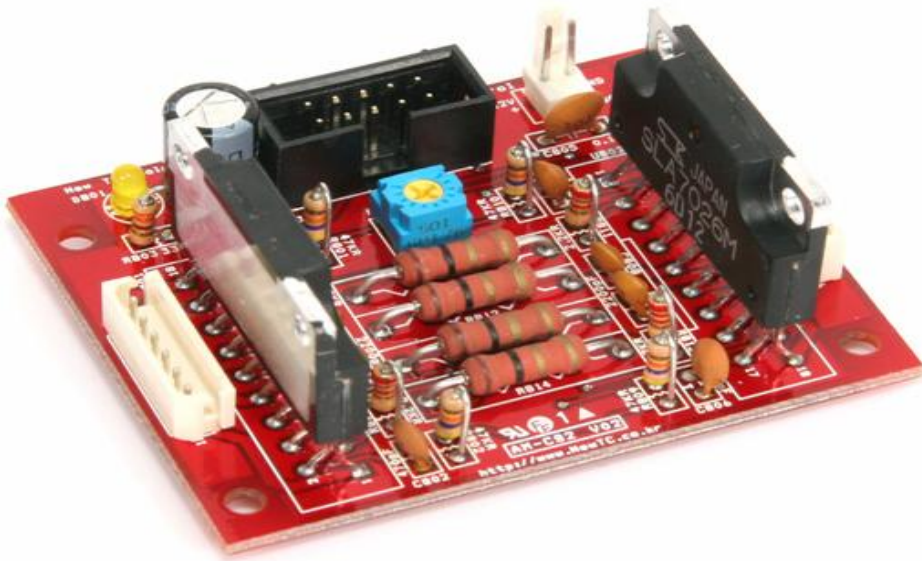
나머지 저항과 캐패시터는 SLA7024 의 기본 회로이다.

예전에는 모터클럭 생성부와 증폭부 모두를 하드웨어 적으로 구성하였으나 모터클럭 생성부의 경우 마이크로 컨트롤러에서 소프트웨어적으로 구현이 가능하기 때문에 여기서는 증폭부만을 하드웨어로 구성하고 아래 MCU F/W 파트에서 모터구성을위한 신호를 소프트웨어적으로 구현하도록 한다.

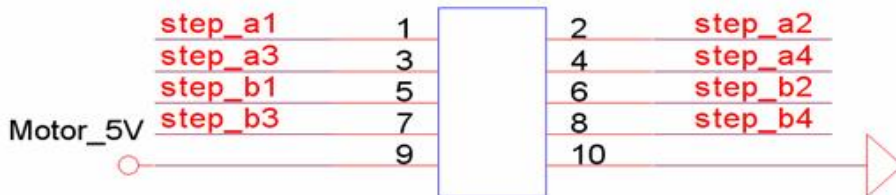
10.6 AM-CS2P(스테핑모터 구동 드라이브)



- ◆ 라인트레이서용 스텝핑 모터 구동보드
- ◆ 스텝핑 모터 2개 구동
- ◆ 소프트웨어적으로 A, /A, B, /B 신호를 인가하여 제어
(1상, 2상 1-2상 제어방식 으로 제어할 수 있다)
- ◆ 10Pin Cable 과 12V 전원 공급 커넥터 연결
- ◆ 모터에 흐르는 전류량을 조절할 수 있음 (가변저항 사용)
- ◆ 외관 크기 63.3 X 50.6 mm



스텝핑 모터 구동 모듈 AM-CS2P



1 번	3 번	5 번	7 번	9 번
왼쪽 모터 A	왼쪽 모터 /A	오른쪽 모터 A	오른쪽 모터 /A	전원 5V
2 번	4 번	6 번	8 번	10 번
왼쪽 모터 B	왼쪽 모터 /B	오른쪽 모터 B	오른쪽 모터 /B	전원 GND

스텝 모터 구동 실습 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
#define MOTOR_PORT      PORTC
#define MOTOR_PORT_DDR  DDRC
void delay(int n)
{ int i,j;
  for(i=1;i<n;i++)
    for(j=1;j<600;j++) ;}
void main(void)
{
  init_devices();
  MOTOR_PORT_DDR = 0xff;
  while(1){
    delay(10); // 1상 여자 방식
    MOTOR_PORT = 0x11;
    delay(10);
    MOTOR_PORT = 0x22;
    delay(10);
    MOTOR_PORT = 0x44;
    delay(10);
    MOTOR_PORT = 0x88;
    /* delay(10); //2상 여자 방식
    MOTOR_PORT = 0x33;
    delay(10);
    MOTOR_PORT = 0x66;
    delay(10);
    MOTOR_PORT = 0xcc;
    delay(10);
    MOTOR_PORT = 0x99; */
  }
}

```

스텝 모터 구동 실습 1,2상 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
#define MOTOR_PORT      PORTC
#define MOTOR_PORT_DDR  DDRC
unsigned char  RIGHTmotorOneClock(unsigned char step)
{
    step = step & 0xf0;
    switch(step){
        case 0x90: step=0x10; break;
        case 0x10: step=0x30; break;
        case 0x30: step=0x20; break;
        case 0x20: step=0x60; break;
        case 0x60: step=0x40; break;
        case 0x40: step=0xc0; break;
        case 0xc0: step=0x80; break;
        case 0x80: step=0x90; break;
        default: step=0xc0; break;
    }
    return step;
}

unsigned char  LEFTmotorOneClock(unsigned char step)
{
    step = step & 0x0f;
    switch(step){
        case 0x09: step=0x01; break;
        case 0x01: step=0x03; break;
        case 0x03: step=0x02; break;
        case 0x02: step=0x06; break;
        case 0x06: step=0x04; break;
        case 0x04: step=0x0c; break;
        case 0x0c: step=0x08; break;
        case 0x08: step=0x09; break;
        default: step=0x0c; break;
    }
    return step;
}

```

스텝핑 모터 구동 실습 1,2상 - ICCAVR

```

void delay(int n)
{
    int i,j;
    for(i=1;i<n;i++)
        for(j=1;j<600;j++) ;
}
void main(void)
{
    unsigned char stepleft=0, stepright=0;
    init_devices();
    MOTOR_PORT_DDR = 0xff;
    while(1){
        stepright = RIGHTmotorOneClock(stepright);
        stepleft = LEFTmotorOneClock(stepleft);
        delay(10);
        MOTOR_PORT = stepleft|stepright;
    }
}

```

이상으로 스텝핑 모터에 관한 기초 사항들을 마무리 하겠습니다. 보다 많은 소스는 <http://www.NTC.co.kr>에 오시면 Q&A에 등록되어 있습니다. 많은 소스를 접하다보면 자신도 모르게 발전하는 자신을 느낄수 있을것 같습니다. 공부하시다가 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다.

11. SENSOR AND ADC 실습

학습 목표

1. Sensor에 대한 이해
2. MCU가 센서의 아날로그 데이터를 어떻게 받을까?
3. VR을 이용한 ADC 값 측정
4. CDS를 이용한 ADC 값 측정
5. IR(AM-ARS)를 이용한 실습 및 응용
6. AVR의 ADC(Analog-to-Digital Converter)를 사용할 수 있다.
 - ADC와 관련된 레지스터를 사용할 수 있다.
 - MUX의 역할을 설명할 수 있다.
 - AD 변환을 위한 순서를 설명할 수 있다.