CHAPTER 12

시리얼 통신

가. 레지스터 구조 이해하기
나. 하이퍼터미널을 이용하여 로봇 제어하기

시리얼 통신 (USART)

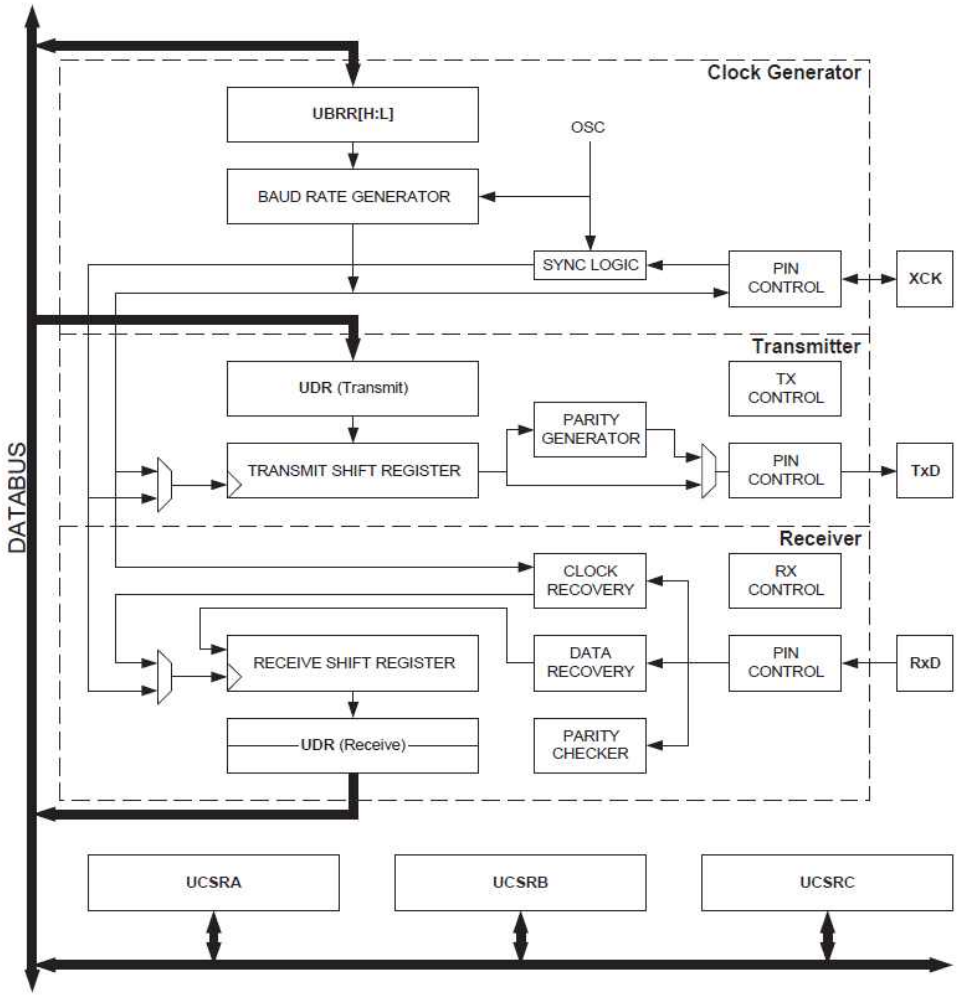
범용 동기 및 비동기 시리얼 수신기와 송신기 (USART)는 매우 유연한 시리얼 통신 장치이다. 주요 특징은 다음과 같다.

- 송수신 레지스터가 독립적으로 운용되는 전이중 방식.
- 비동기 또는 동기 동작.
- 마스터 또는 슬레이브 동기 동작.
- 고해상도 전송 속도 생성기.
- 5, 6, 7, 8, 또는 9 비트의 데이터와 1개 또는 2개의 정지 비트 직렬 프레임을 지원.
- 홀수 또는 짝수 패리티 비트 생성 및 하드웨어에 의한 패리티 검사 지원.
- 데이터 오버런(Overrun) 검출 장치.
- 오류 감지(Framing error)를 도용.
- 노이즈 필터링 검출 및 디지털 저역 통과 필터 지원.
- TX Complete, TX Data Register Empty, RX Complete 별도의 세 가지 인터럽트 지원.
- 멀티프로세서 통신 모드.
- 비동기 통신 모드에서의 보율 더블러(Baud rate doubler)

USART 송신기의 단순 블록 다이어그램은 그림 8.1에 나타낸다. CPU 접근 I/O 레지스터 및 I/O 핀은 굵은 글씨로 표시된다. 그림 8.1에서 점선으로 처리된 부분은 Clock generator, Transmitter, Receiver 세 부분으로 구성되어 있으며, 이 세 부분은 제어 레지스터를 공통으로 사용하고 있다.

Clock generator는 동기 슬레이브 동작에서 사용하는 외부 클록 입력과, 보율 발생 장치로 구성되어 있고, Transmitter는 단일 쓰기 버퍼, 시프트 레지스터, 패리티 발생기 등으로 이루어져 있으며, 단일 쓰기 버퍼는 프레임 간에 지연 시간 없이 연속적인 데이터를 보낼 수 있게 해준다.

Receiver는 USART 모듈의 가장 복잡한 구조이며, 클록 및 데이터 복구 장치로 이루어져 있다. 여기서 데이터 복구 장치는 비동기 데이터 수신에서 사용하고, 패리티 검사기와 제어로직, 시프트 레지스터와 두 개의 수신 버퍼(UDR)로 이루어져 있다.



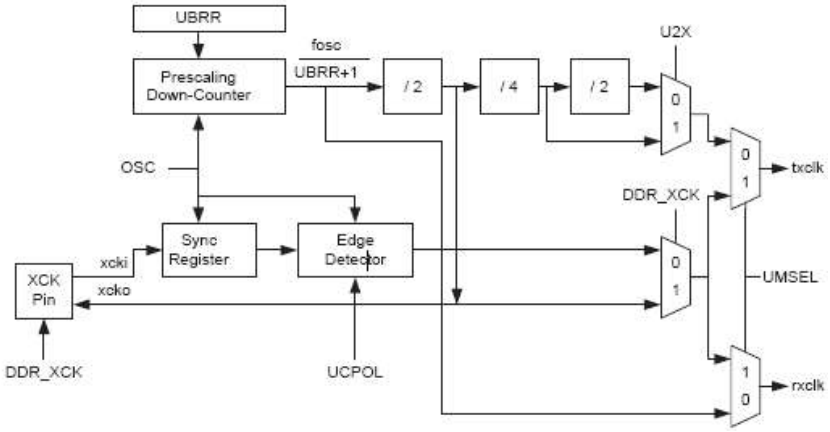
[그림 8.1] USART 블록 다이어그램

가. 레지스터 구조 이해하기

1) 클록 발생 장치

클록 발생 장치는 송신 및 수신의 가장 기본적인 클록을 만들어 낸다. USART 모듈은 Normal Asynchronous mode, Double Speed Astnchronous mode, Master Synchronous mode, Slave Synchronous mode 등 네 가지의 모드가 있다. UCSRC 레지스터의 UMSEL 비트를 설정하여 동기 및 비동기 방식을 선택할 수 있다. 비동기 모드에서만 지원되는 보율 더블러는 UCSRA 레지스터의 U2X를 설정하여 속도를 두 배로 증가할 수 있다.

동기 모드로 설정되었을 경우에(UMSEL=1), XCK 핀(DDR_XCK)의 데이터 방향 레지스터는 클록 소스가 내부(Master mode)에 있는지, 아니면 외부(Slave mode)에 있는지를 결정한다. 즉, XCK 핀은 동기 모드에서만 활용된다. 그림 8.2에 클록 발생 장치를 나타낸다.



[그림 8.2] 클록 발생 장치

그림 8.2의 신호에 대한 설명은 아래와 같다.

- txclk : 송신 클록(내부 신호).
- rxclk : 수신 클록(내부 신호).
- xcki : XCX 핀으로부터의 입력(내부 신호), Slave Synchronous mode에서 지원.
- xcko : XCK 핀에 대한 출력(내부 신호), Master Synchronous mode에서 지원.
- fosc : XTAL 핀 주파수(시스템 클록)

내부 클록 발생 장치는 Asynchronous 및 Synchronous Master mode에서 사용되며, USART 모듈의 보율 발생 레지스터(UBRR)는 다운카운터로 동작하며 프리스케일러 또는 보율 발생 장치의 기능을 한다. 시스템 클록(f_{OSC})과 동기해서 동작하는 다운 카운터는 UBRRL 레지스터에서 새로운 값을 쓰던지 아니면 카운터가 “0” 이 되면 값이 로드 된다. 이 클록이 바로 보율 발생 장치 클록의 출력이 된다. ($f_{OSC}/(UBRR+1)$) 송신기는 보율 발생 클록의 2, 4, 8 분주로, 수신기는 2, 8, 16 분주로 동작하며, UMSEL, U2X 그리고 DDR_XCK 비트의 설정에 따라서 달라진다. 그림 8.3에 보율 발생 계산식 및 각 보드에 따른 UBRR 값을 나타낸다.

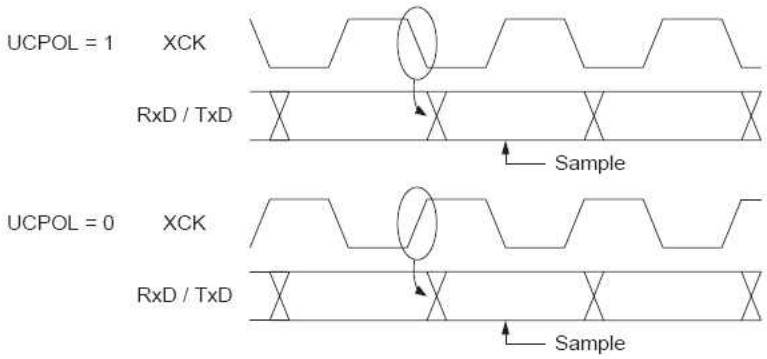
Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

[그림 8.3] Baud Rate Register 계산식

여기서, BAUD는 Baud rate(bps)이고, f_{OSC}는 시스템 클록을 의미하며, UBRR은 UBRRH와 UBRRL 레지스터를 의미한다. Synchronous Master Mode에서 지원하는 외부 클록은 XCK 핀으로 입력되는 외부 클록으로 송신 및 수신기로 사용될 때 에지 검출기에 의해서 검출된다. 이 과정에서 두 개의 CPU 클록 지연과 최대 외부 XCK 주파수는 다음 식으로 제한이 된다.

$$f_{XCK} < \frac{f_{OSC}}{4}$$

UMSEL=1로 설정하여 동기 모드인 경우에, XCK 핀은 클록 입력(Slave) 또는 클록 출력(Master)으로 사용된다. RXD의 데이터 입력은 XCK 클록 에지에서 샘플링 되며 이 과정을 그림 8.4에 나타낸다.



[그림 8.4] 동기 모드에서의 XCK 타이밍

UCSRC 레지스터의 UCPOL은 XCK 클록 에지가 어떤 데이터 샘플링에서 사용되는지 또는 데이터 변환에 사용되는지를 결정한다. 그림 118에 나타냈듯이 UCPOL이 “0” 이면, 데이터는 XCK의 상승 에지에서 변하고, XCK의 하강 에지에서 샘플링 된다. 반대로 UCPOL이 “1” 로 설정되면, 데이터는 XCK의 하강 에지에서 변하고 XCK의 상승 에지에서 샘플링 된다.

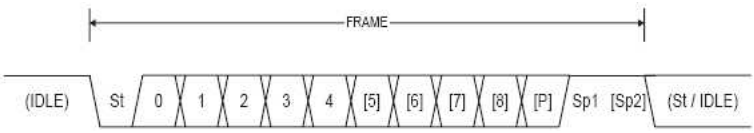
2) Frame format

직렬 데이터 프레임은 동기 비트인 시작과 정지 비트를 포함한 문자를 의미하는 것으로 여기에는 에러 검출을 위한 패리티 비트가 추가될 수도 있다. ATmega128의 USART의 프레임은 다음과 같다.

- 1 start bit
- 5, 6, 7, 9 data bit

- no, even or odd parity bit
- 1 or 2 stop bit

데이터 프레임은 start bit로 시작하여 LSB 비트 순서대로 전송하며, 전체 9비트인 MSB가 전송되면 완료된다. 패리티 비트가 있으면 MSB비트 다음에 위치하고 stop bit가 위치한다. 프레임 전체가 송신이 완료되면, 새로운 프레임이 오거나 idle 상태가 된다. 그림 8.5에 프레임 조압을 나타낸다.



[그림 8.5] 데이터 프레임

- St : start bit(Low)
- (n) : Data bit(0~8)
- P : Parity bit(even 또는 odd)
- Sp : Stop bit(High)
- IDLE : 데이터가 없는 경우로 언제나 High 상태가 된다.

패리티 비트는 모든 데이터 비트의 exclusive-or를 이용하여 계산하는데, odd parity가 설정되면 exclusive-or 결과 같의 역이 되고, 다음과 같은 식이 성립된다.

$$P_{even} = d_{n-1} \cdots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \cdots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

여기서, P_{even}은 even parity를 이용한 패리티 비트이고, P_{odd}는 odd parity를 이용한 패리티 비트이다.

3) USART 초기화

USART의 초기화는 통신이 시작되기 전에 설정되어야만 한다. 일반적인 초기화는 보율을 설정하고, 프레임 및 송수신을 결정하면 되며, 초기화 과정에는 인터럽트를 disable하여야 한다. 아래의 예제는 초기화 과정을 나타낸다.

```
void USART_Init(unsigned int baud)
```

```
{
  /* Set baud rate */
  UBRRH = (unsigned char)(baud>>8);
  UBRRH = (unsigned char)baud;
  /* Enable receiver and transmitter */
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* Set frame format : 8 bit data, 2stop bit */
  UCSRC = (1<<USBS)|(3<<UCSZ0);
}
```

4) USART의 송신 장치

USART의 송신기는 UCSRB 레지스터의 TXEN 비트를 "1"로 설정함으로써 enable된다. 송신기가 enable된 경우에는 TXD 핀의 범용 포트의 동작은 되지 않으며, USART의 데이터 출력용으로 이용된다. 보율, 동작모드, 데이터 프레임 등의 설정은 송신을 하기 전에 설정되어야만 하고, 동기 모드로 동작하는 경우에는 XCK의 클록은 송신 클록으로 사용된다. 데이터 송신은 전송할 데이터가 송신 버퍼에 로드되면 초기화된다. 그러면 CPU는 UDR에 값을 써서 송신 버퍼의 내용을 로드한다. 시프트 레지스터가 다음 데이터를 송신할 준비가 되어 있으면, 송신 버퍼에 있는 데이터가 시프트 레지스터로 이동한다. 다음 예제에 USART의 송신기가 Data Register Empty(UDRE) 플래그를 이용하여 송신하는 과정을 나타냈다.

```
void USART_Transmit(unsigned char data)
{
  /* Wait for empty transmit buffer */
  while (!(UCSRA & (1<<UDRE)));
  /* Put data into buffer, sends the data */
  UDR = data;
}
```

UCSZ=7인 경우, 즉 9비트 데이터인 경우에는 9번째 비트는 UCSRB 레지스터의 TXB8 비트가 "1"로 설정되어 있어야 한다. 다음 예제에 9번째 비트를 송신하는 과정을 나타낸다.

```
void USART_Transmit(unsigned char data)
{
```

```
  /* Wait for empty transmit buffer */
  while (!(UCSRA & (1<<UDRE)));
  /* Copy 9 th bit from r17 to TXB8 */
  UCSRB &= ~(1<<TXB8);
  if (data & 0x0100)
    UCSRB |= (1<<TXB8);
  /* Put LSB data(r16) into buffer, sends the data */
  UDR = data;
}
```

USART의 송신기는 USART Data Register Empty(UDRE), Transmit Complete(TXC) 등의 상태를 나타내는 2개의 플래그 레지스터가 있다. UDRE 플래그는 송신 버퍼가 새로운 데이터를 받을 준비가 되어 있다고 알려주는 역할을 하며, 송신 버퍼가 비어 있을 때 "1" 로 Set 되며, 아직 시프트 레지스터에 옮겨지지 않은 송신할 데이터가 있는 경우에는 "0" 으로 클리어 된다. UCSRB 레지스터의 Data Register Empty Interrupt Enable(UDRIE) 비트를 "1" 로 설정하면 UDRE 인터럽트가 수행된다. 물론 여기서 전체 인터럽트는 enable되어 있어야만 한다. UDRE 플래그는 UDR에 새로운 값을 쓰게 되면 "0" 으로 클리어 된다. 인터럽트에 의해서 데이터가 전송이 되면, UDR 인터럽트 서비스 루틴은 UDRE 플래그를 클리어하기 위해서 UDR에 새로운 데이터를 써 넣든지 아니면 UDR 인터럽트를 disable해야만 한다. 그렇지 않으면, 인터럽트가 종료한 후에도 새로운 인터럽트가 발생하게 된다.

Transmit Complete(TXC) 플래그는 송신 시프트 레지스터에 있는 전체 프레임이 시프트로 출력하고 송신버퍼에 현재 새로운 데이터가 존재하지 않으면, "1" 로 Set 된다. TXC플래그 비트는 송신 완료 인터럽트가 실행되거나, TXC 비트에 "1" 로 덮어쓰면 "0" 으로 클리어 된다. TXC 플래그가 RS485 방식과 같은 반이중 통신 방식에서 사용된다. UCSRB 레지스터의 Transmit Complete Interrupt Enable(TXCIE) 비트가 "1" 로 Set 되면, USART Transmit Complete 인터럽트가 실행된다. 물론 여기서 전체 인터럽트는 enable되어 있어야만 한다. USART Transmit Complete 인터럽트를 사용할 경우에는 인터럽트가 수행되고 나면 TXC 비트는 자동으로 클리어 되므로 TXC 플래그를 클리어 할 필요는 없다.

5) USART의 수신 장치

UASRT의 수신기는 UCSRB 레지스터의 Receive Enable(RXEN) 비트를 "1" 로 설정함으로써 enable된다. 수신기가 enable된 경우에는 RXD 핀의 범용 포트의 동작은 되지 않으며, USART의 데이터 입력용으로 이용된다. 보율, 동작모드, 데이터 프레임 등의 설정은 수신을 하기 전에 설정되어야만 하고, 동기 모드로 동작하는 경우

에 XCK의 클록은 송신 클록으로 사용된다. 수신기는 유효한 시작 비트가 입력되면 수신하기 시작하고, 각각의 데이터 비트는 보율 또는 XCK 핀에서 샘플링 하여 프레임의 철 번째 정지 비트가 수신될 때까지 수신 시프트 레지스터에 데이터를 시프트 한다. 첫 번째 정지 비트가 수신되면, 즉 전체 직렬 프레임이 수신 시프트 레지스터에 수신 완료되면, 수신 시프트 레지스터의 내용을 UDR 레지스터를 통하여 읽을 수 있다. 다음 예제는 Receive Complete(RXC) 플래그를 기초로 하여 간단한 USART 수신 기능을 나타낸다.

```
void USART_Transmit(unsigned char data)
{
  /* Wait for data to be received */
  while (!(UCSRA & (1<<RXC)));
  /* Get and return received data from buffer */
  return UDR;
}
```

UCSZ=7인 경우, 즉 9비트 데이터인 경우에는 9번째 비트는 UCSRB 레지스터의 RXB8 비트가 “1”로 설정되어 있어야 한다. 다음 예제는 9번째 비트를 수신하는 과정을 나타낸다.

```
unsigned int USART_Receive(void)
{
  unsigned char status, resh, resl;
  /* Wait for data to be received */
  while (!(UCSRA & (1<<RXC)));
  /* Get status and 9 th bit, then data */
  /* from buffer */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* If error, return -1 */
  if (status & (1<<FE)¦(1<<DOR)¦(1<<PE)
    return -1;
  /* Filter the 9 th bit, then return */
  resh = (resh>>1) & 0x01;
  return((resh<<8) ¦ resl);
```

```
}
```

USART 수신기는 수신의 상태를 나타내는 Receive Complete(RXC) 플래그가 있어서 수신 버퍼에 아직 읽지 않은 데이터가 있는지를 나타낸다. 아직 읽지 않은 데이터가 있으면 “1”로 표시되고, 수신 버퍼가 비어 있는 경우에는 “0”으로 표시된다. UCSRB 레지스터의 Receive Complete Interrupt Enable(RXCIE) 비트가 “1”로 Set 되면, USART Receive Complete 인터럽트가 수행된다. 물론 전체 인터럽트는 수행되어 있어야만 한다. 수신 완료 인터럽트가 수행되면, 수신완료 인터럽트 서비스 루틴에서 RXC 플래그를 클리어하기 위해서는 UDR에 있는 수신 데이터를 읽으면 된다. 그렇지 않으면 인터럽트가 종료하여도 새로운 인터럽트가 발생하게 된다.

USART의 수신기는 Frame Error(FE), Data OverRun(DOR), Parity Error(PE) 등의 세 가지 에러 플래그가 있으며, UCSRA를 읽어서 접근할 수 있다.

FE 플래그	수신 버퍼에 저장된 다음 프레임의 첫 정지 비트의 상태를 나타낸다. 정지 비트가 정상적으로 읽혀지면 FE 플래그는 “0”으로 된다.
PE 플래그	수신 버퍼에 있는 다음 프레임에 패리티 에러가 발생한 경우에 표시된다.
DOR 플래그	수신버퍼가 꽉 찬 경우에 데이터의 손실 상태를 나타낸다. DOR은 수신 버퍼가 가득 찬 경우, 수신 시프트 레지스터에 새로운 대기 문자가 있는 경우, 새로운 시작 비트가 검출된 경우에 발생된다.

6) 멀티프로세서 통신 모드

UCRSA 레지스터의 Multi-processor Communication Mode(MPCM) 비트를 “1”로 설정하여 멀티프로세서 기능을 사용할 수 있다. 멀티프로세서 통신 모드를 1개의 마스터 MCU가 여러 개의 슬래이브 MCU에게 특정 주소를 전송함으로써 1개의 슬레이브를 지정하여 데이터를 전송하는 동작 모드이다. 만약에 수신기가 5~8 비트로 설정된 데이터 프레임으로 설정되어 있으면, 첫 번째 정지 비트는 프레임이 데이터 정보인지 어드레스 정보인지를 가리킨다. 만약 수신기가 9비트로 설정되어 있으면, 9번째 비트(RXB8)는 어드레스 또는 데이터를 표시하게 된다. 프레임 지정 비트(첫 번째 비트 또는 9번째 비트)가 “1”로 설정되면 프레임은 어드레스 정보를 나타내고, “0”으로 설정하면 데이터 프레임이다. 다음 순서에 의해서 진행된다.

- 송신을 담당한 마스터 MCU는 특별한 모드 설정이 필요 없고, 여러 개의 수신을 하는 슬레이브 MCU는 UCSRA 레지스터의 MPCM 비트를 “1”로 설정하여 어드레스 프레임이 수신되기를 기다린다.

- 마스터 MCU는 8개의 데이터 비트에 지정하려고 하는 어드레스와 UCSRA 레지스터의 TXB8을 “1”로 만든 어드레스 프레임을 송신한다.
- 모 슬레이브 MCU는 이 어드레스 프레임을 수신하여 자신에게 해당되는 어드레스인지를 판별하, 이것이 자신의 어드레스인 것을 확인한 1개의 슬레이브 MCU만이 MPCM비트를 “0:으로 클리어 하여 이후에 전송될 데이터 프레임을 수신할 수 있도록 한다.
- 나머지의 선택되지 않은 슬레이브 MCU들은 계속하여 MPCM비트를 “1”로 유지하여 어드레스 프레임이 수신되기를 기다린다.
- 선택된 슬레이브 MCU는 데이터 프레임을 수신하고 이것이 완료되면 다시 MPCM비트를 “1”로 설정하여 어드레스 프레임의 수신 대기 상태로 진입하게 된다.

7) USARTn 레지스터

Atmega128에는 두 개의 Universal Synchronous and Asynchronous serial Receiver and Transmitter(USART)가 있다. 구분은 “0”과 “1”로 하고 있으며 각각의 USART는 독립적인 송수신 버퍼를 가지고 있고 하드웨어 적으로 분리 되어 있지만 UDR이라는 명칭은 동일하게 사용된다. 우선 USART에 대해 이해하기 전에 UART 비동기 통신에 대해서 이해하여야 한다. 일반적인 통신은 Serial방식과 Parallel방식의 통신 방식이 있다. 시간을 기준으로 데이터가 순차적으로 전송되면 Serial방식이고, 같은 시간에 한 번의 데이터가 여러 개의 전송선로를 통해서 전송되면 Parallel이다. 이때 어떤 것이 좋다고는 말할 수 없다. 일반적으로 PC에서 사용되는 USB COM 단자, SATA등과 같은 방식은 Serial방식이고 CPU에서 RAM과 같은 곳의 데이터를 읽기 위한 선로는 여러 개의 선로에 데이터가 동시에 전달되고, 일반적으로 부르는 BUS방식은 Parallel방식이다. 두 가지 통신방식 중 ATmega128에서 사용할 통신 방식은 Serial방식이다. 시리얼 통신방식에서도 두 가지의 방식이 세부적으로 나누어진다.

첫 번째 동기 통신과 비동기 통신으로 나누어지는데 동기 통신 방식은 데이터가 전송될 때 전기적으로 신호 선에 신호가 전달될 때는 Analog신호가 전송되는 것이 아니라 Digital신호를 전달하는 방식을 사용한다. 이때 On과 Off의 신호의 조합으로 8bit를 전송하여 데이터를 만드는 방법과 16bit의 데이터를 조합하여 데이터를 만드는 방법 등과 같은 여러 가지 방법적은 구분으로 나누어진다. 이때 On과 Off신호가 송신측과 수신측에서 언제 보내는지 확인 할 수 있도록 타이밍을 만들어 주는 선로가 존재하여 신호를 보내 주면 동기 방식의 통신이 되고, 수신측에서 언제 들어올지 모르는 데이터를 기다리며 송신측에서는 시간에 상관없이 데이터를 송신 하면

비동기 받는 측에서는 언제 올지 모르는 통신을 사용하게 된다.

비동기와 동기의 가장 큰 차이는 동기는 데이터 선외에 추가적으로 동기를 맞춰주는 전송 선로가 존재 하게 되고 일반적인 컴퓨터에서 사용하는 Dsub-9pin의 경우 동기를 맞춰주는 PIN이 할당 되어 있다. 그러면 비동기 통신 방법에서 의문이 발생될 것이다 언제 데이터가 수신될지 모르는 과정에서 어떻게 신호를 잡아 낼 수 있을까 항상 그 신호만 감지하고 있는 것이 아닌데 라고 생각이 들것이다. 이러한 에러를 피하기 위해서 소프트웨어 적인 방법이 아닌 하드웨어 적인 방법을 마이크로 컨트롤러에서는 사용된다.

일반적으로 비동기 통신 방식의 가장기본적인 방법으로 양방향통신 조건에서 Tx, Rx, GND 세 가지의 신호선 연결만으로 통신을 수행한다. 통신의 발전과 전송데이터 증가에 따른 신호선의 전압 레벨 및 조건이 변했을 뿐 기본적인 바탕은 위의 방법만으로 충분하다. 비동기 통신방법에서 가장 중요한 요소는 전송 통신의 규약이다. 일반적으로 사용하는 데이터 형태는 전송을 알리는 스타트 비트 1bit 데이터 8bit Stop비트, 1bit등과 같은 형태로 전송된다. 실제로 8bit를 전송하기 위해서 10비트를 사용하고 전송된 데이터의 결함을 검사하기 위해서 결함검사를 목적으로 한 패리티 비트를 추가적으로 넣는 경우, Stop비트는 2bit를 넣어 다양한 형태로 데이터 전송의 무 결성을 입증할 만한 정보들을 포함하여 보낸다.

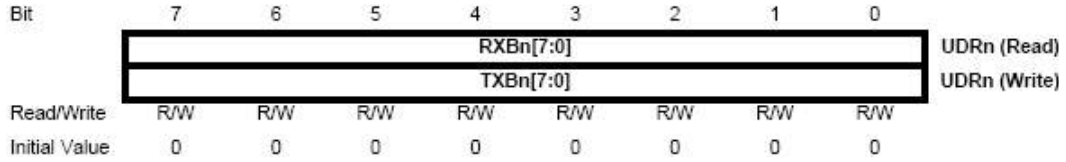
이 같은 이유는 고속으로 많은 데이터를 송수신 할 때 믿을만한 정보인가를 판단하여 사용유무를 검사하기 위함이다. 하지만 일반적으로 8bit를 한 개씩 전송하는 경우는 없다고 생각해도 무방하다. 시스템의 용도와 크기에 따라 다르지만 frame이라는 부르는 형태의 데이터 조합을 순차적으로 보내어 그 안에서 다시 시작을 알리는 바이트 형태의 데이터와 데이터 무 결성을 검사하기 위한 checksum, CRC코드 등을 추가 하여 무 결성을 검사 한다.

ATmega128에서 TxD핀에 출력을 내보내기 위해서는 전송 UDR 버퍼에 보내고자 하는 데이터를 쓰면 된다. 쉬프트레지스터에서 시간에 따라 자동적으로 신호를 전송해 준다. 이 같은 기능은 내부에 하드웨어 적으로 수행되며, 소프트웨어에서는 UDR에 단순히 보내고자 하는 데이터를 쓰기만 하면 된다. 이때 내부에서 패리티비트 제너레이터가 있어 설정에 따라 자동적으로 계산해서 출력을 구성한다. 반대로 수신할 때는 마이크로 컨트롤러가 기능을 수행해도 수신측에서 하드웨어 적으로 신호를 검출한다. 신호검출은 전송속도와 상관관계를 가지며, 수신측과 송신측은 송수신 속도를 알고 있어야만 정확한 데이터를 검출할 수 있다.

RxD포트를 통해서 신호(On, Off)가 들어오게 되고, 이때 스타트비트 데이터 패리티 비트등과 같은 신호를 검출하여 수신 쉬프트 레지스터로 한 비트 씩 전송하게 되고, 이때 규약에 따라 다르지만 8비트 설정 7비트 6비트 등과 같이 설정된 비트가 전송이 완료되면, 인터럽트가 발생되거나 수신완료 레지스터를 검색하여 사용자로 하여금 수신이 완료되었음을 확인 할 수 있다. 예제를 이용하여 사용될 통신 방식은 비동기, 양방향 통신을 사용하고, Start비트는 1비트 Stop비트, 1비트 패리티

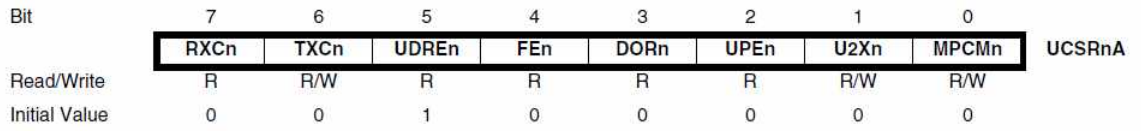
비트는 “0”, 데이터크기 8bit, 통신 속도는 115200bps으로 가장 범용 적으로 사용하는 통신 방식을 구현해보도록 하겠다.

우선 그림 8.6과 같이 레지스터를 살펴보면 UDRn 레지스터는 데이터를 송신 수신 할 때 사용되는 레지스터로 데이터를 입력 시키는데 사용된다.



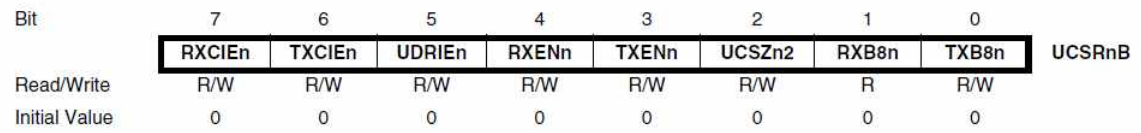
[그림 8.6] UDRn 레지스터

그림 8.7과 같이 UCSRnA 레지스터는 사용 중인 UART기능에서 버퍼의 송신 및 수신이 완료 되었는지에 대한 상태를 읽을 수 있다. 특히 인터럽트 방식을 쓰지 않고 수신 상태를 확인하기 위해서 반드시 확인할 수 있는 레지스터 이며, 송신 시 버퍼에 데이터가 시간차를 가지고 전송되어 지기 때문에 전송이 완료 되지 않은 상태에서 데이터를 다시 UDRn에 다시 쓰게 되면 버퍼에 이중으로 쓰이게 되고 전송 시 데이터가 제대로 전달되지 않기 때문에 송신 시에도 반드시 체크해야 되는 레지스터다.



[그림 8.7] UCSRnA 레지스터

그림 8.8과 같이 USCRnB 레지스터는 통신사용 시 발생되는 인터럽트를 제어할 수 있다. 송신완료 인터럽트, 수신완료 인터럽트를 설정할 수 있도록 비트 7번과 6번에 설정할 수 있고 설정 값은 “1”로 설정함으로서 사용가능 하다 반드시 인터럽트 사용을 설정해 놓으면 AVR Studio 4에서 지원하는 WinAVR에 “signal.h” 파일인 헤더 파일을 참조하여 인터럽트벡터 영역의 함수를 추가해야 한다. 만약 함수가 없을 경우 마이크로컨트롤러는 동작을 멈추고 정지 하게 된다.



[그림 8.8] UCSRnB 레지스터

TXENn와 RXENn비트는 송수신에 대해 각각 사용이 가능하게 할 것인가에 대한 사용 성을 결정하는 레지스터 비트이다. 송수신은 각각의 설정이 별개로 가능하기 때문에 송신만 쓰는 경우, 수신만 쓰는 경우와 두 가지를 모두 쓰는 경우 등에 대해서 설정할 수 있다.

그 외에 특별히 비트 1(RXB8n)은 9비트 송신 시 UDRn은 8비트 레지스터이기 때문에 8번째 비트를 읽어 낼 수 없다. 그래서 특별히 USCRnB의 비트 1(RXB8n)을 읽으면 수신된 8번째 비트를 읽어 낼 수 있다. 이와 반대로 0번 비트(TXB8n)는 9비트 전송 시 8번째 비트를 전송할 버퍼가 없기 때문에 반대로 USCRnB의 0번에 써주고 전송하면 된다. 이때 수신시 UDRn을 먼저 읽고 다음으로 USCRnB의 1번 비트를 읽어야 하며, 송신 시 USCRnB의 0번 비트에 데이터를 먼저 쓰고 UDRn에 써야 전송이 된다.

그림 8.9와 같이 UCSRnC 레지스터는 상태제어용 레지스터이다. 위에서 언급한 동기/비동기, Stop비트, 패리티 비트, 전송데이터 비트 수 등을 설정할 수 있다.

Bit	7	6	5	4	3	2	1	0	
	–	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

[그림 8.9] UCSRnC 레지스터

그림 8.10과 같이 UMSELn(USARTn Mode Select) 비트는 그림 8.6과 같이 “1” 이면 USARTn 모듈을 동기 전송 모드로 설정하고, “0” 이면 비동기 전송 모드로 설정한다.

UMSELn	Mode
0	Asynchronous Operation
1	Synchronous Operation

[그림 8.10] UMSELn Bit 설정

UPMn1, 0(Parity Mode) 비트는 그림 8.11과 같이 “1” 로 설정하면 패리티를 발생시키고 검사를 할 수 있다. 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 Set 된다.

UPMn1	UPMn0	Parity Mode
0	0	Disabled

0	1	(Reserved)
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

[그림 8.11] UPMn Bit 설정

USBSn(Stop Bit Select) 비트는 그림 8.12와 같이 "0" 이면 USARTn 모듈에서 데이터 포맷을 구성하는 Stop비트를 1개로 설정하고, "1" 이면 Stop비트를 2개로 설정한다.

USBSn	Stop Bit(s)
0	1-bit
1	2-bits

[그림 8.12] USBSn Bit 설정

UCPOLn 비트는 그림 8.13과 같이 설정한다.

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

[그림 8.13] UCPOLn Bit 설정

UCSZn1, 0(Chracter Size) 비트는 그림 8.14와 같이 UCSRnB 레지스터의 UCSZn2 비트와 함께 전송 문자의 데이터 비트 수를 설정하는데 사용된다.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

[그림 8.14] UCSZn Bit 설정

마지막으로 그림 8.15와 같이 UBRRnL, UBRRnH 레지스터이다. 비동기 통신에서 가장 중요한 요소를 설정하는 Baud Rate Register이다. 즉 통신 속도를 정의 하는 레지스터로 H는 상위 L은 하위를 나타내고 총 길이로 보면 16비트 레지스터라고 할 수 있다.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

[그림 8.15] UBRRnL, UBRRnH 레지스터

많은 사용자들이 이러한 기능을 먼저 설명하고 나면 복잡해지기 시작한다. 많은 기능 중에 사용자가 원하는 기능은 단순할 때가 많기 때문이다. 시리얼 통신도 이와 마찬가지 일 것이다. 일반적인 통신에서 Stop 1비트 패리티는 경우에 따라 쓰이는 곳과 쓰지 않는 곳이 있고, 데이터 크기는 대부분의 많은 프로그램에서 8비트의 데이터 크기를 가진다. 이때 자주 쓰이는 통신 속도는 9600bps와 38400bps, 115200bps 등과 같이 몇 가지만 주로 쓰이기 때문이다.

9600bps이하로 전송하면 통신의 속도가 느리기 때문에 자주 쓰이지 않는다. bps란 초당 보내는 비트수 이며, 가령 Start 1비트, Data 8비트, Stop 2비트, Parity 1비트를 합치면 12비트가 된다. 이때 9600bps로 전송하면 초당 800개의 8비트 데이터를 보낼 수 있다. 하지만 115200bps로 전송 시 9600개의 8비트 데이터를 전송한다. 최근에 쓰이는 통신 속도 및 데이터 크기에 비해서 작은 용량이지만 이정도로도 많은 시스템에서 필요한 정보는 전달 가능한 속도이다.

115200bps이상 전송속도를 증가하게 되면 전송선로 길이에 따른 감쇠에 따른 문제와 상관관계가 발생되지만, 데이터가 손실되거나 왜곡등과 같은 변형의 문제와 관련이 발생하여 통신상의 문제가 발생 할 수 있다.

자 그럼 본격적으로 역으로 프로그램을 작성해 보도록 하자. 설정은 아래와 같다.

- 시리얼통신 포트0 사용.
- 전송속도 115200bps.
- Stop비트 1비트.
- 패리티비트 없음.

- 데이터비트 8비트.
- 송수신 동시에 가능.
- 송신인터럽트 사용안함, 수신인터럽트 사용함.

위의 사실을 바탕으로 프로그램을 해보도록 하겠다.

우선 통신 속도 설정을 먼저 하도록 하자 통신 속도는 115200bps이다. 실제로 통신 속도 또한 메인 클록 주파수와 상관관계를 가지고 있다. 그만큼 메인 클록 주파수는 중요함을 한 번 더 상기 하도록 하자. 이때 계산방법은 그림 8.16과 같이 계산하는 방법이 있지만 특별한 경우 범용으로 사용하지 않는 속도의 경우 계산하여 연결해야 하지만 범용 적으로 사용하는 속도는 이미 다 계산을 해서 테이블 화해서 제공된다.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

[그림 8.16] 보율 설정 수식 표

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max [1]	62.5Kbps		125Kbps		115.2Kbps		230.4Kbps		125Kbps		250Kbps	

[그림 8.16.1] 자주 사용되는 주파수에 대한 UBRR 값

Baud Rate (bps)	f_{osc} = 3.6864MHz				f_{osc} = 4.0000MHz				f_{osc} = 7.3728MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max (1)	230.4Kbps		460.8Kbps		250Kbps		0.5Mbps		460.8Kbps		921.6Kbps	

[그림 8.16.2] 자주 사용되는 주파수에 대한 UBRR 값(1)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

[그림 8.16.3] 자주 사용되는 주파수에 대한 UBRR 값(2)

보율 설정은 메인 클록 주파수에 따라서 설정 값이 달라지며, Timer/Counter와 같이 계산하였을 때 카운터 값이 소수점 자리에 오차가 발생하였던 것과 동일하게 통신에서도 메인 클록 주파수에 따라 송수신간의 에러가 그림 8.14와 같이 존재한다. 통신의 에러가 발생하지 않도록 CRX10에서는 14.7456MHz라는 클록을 사용하게 된 이유이다.

그림 8.17에서 확인한 봐와 같이 115200bps 사용 시 16MHz에서도 -3.5%라는 에러가 존재한다. 통신에서 설계상의 에러 율은 데이터가 많으면 많을수록 더 많은 에러를 발생시킨다. 그렇기 때문에 완벽한 통신을 위해서는 Error가 0%인 영역에서 사용하는 것이 마이크로컨트롤러를 사용하여 통신을 하게 되는 기본동작에서 최종적으로 에러를 줄일 수 있는 요소로 작용한다.

	f_{osc} = 16.0000MHz			
	U2X = 0		U2X = 1	
Baud Rate (bps)	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max (1)	1Mbps		2Mbps	

[그림 8.17] 일반적으로 사용되는 발진기 주파수에 대한 UBRR 설정 값과 오차율

14.7456MHz에서 115200bps를 전송할 때 UCSRnA의 1번 비트(U2Xn) 설정 값에 따라서 UBRR의 값이 변하게 된다. U2Xn을 “0”으로 설정하면 UBRR에 “7”을 U2Xn에 “1”로 설정 시 UBRR에 “15”로 설정해야 한다. 테이블을 보고 간단히 값을 대입해서 입력만 시키면 된다. 이때 표의 값은 10진수이다. UCSR0A레지스터는 5번 비트(UDRE0)을 “1”로 설정해 주어야 한다. 기본적으로 리셋이후에 초기화되지만 확인차원에서 한 번 더 “1”로 설정해주자. UCSR0A 는 0x20으로 설정하고, UCSR0B는 아래와 같이 설정할 수 있다.

Bit7	RXCIEn : RX Complete Interrupt Enable 수신완료시 인터럽트 사용여부.
Bit6	TXCIEn : TX Complete Interrupt Enable 송신완료시 인터럽트 사용여부.
Bit4	RXENn : Receiver Enable 수신기능을 사용여부.
Bit3	TXENn : Transmitter Enable 송신기능 사용여부.

UCSR0B는 위 4가지 비트에 대해서 체크해 준다. 하지만 Bit6번은 송신 시 인터럽트는 사용하지 않는다고 했기 때문에 설정하지 않는다. UCSR0B은 0x98로 설정한다. 이제 마지막으로 UCSR0C을 설정하여야 한다. UCSR0C는 아래와 같이 설정할 수 있다.

bit6	UMSELn: USART Mode Select.
bit5~4	UPMn1:0 : Parity Mode.
bit3	USBSn : Stop Bit Select.
bit2~1	UCSZn1:0 : Character Size.

bit6의 경우 “0” 일 때 Asynchronous Operation 비동기 통신으로 동작하고 “1” 일 때 Synchronous Operation 동기 통신으로 동작한다. 이미 언급했듯이 비동기 통신으로 설정하기 때문에 “0” 으로 설정된다.

bit5번과 bit4번은 패리티비트를 체크 한다. 패리티 비트란 보내는 데이터에서 이진수로 계산하였을 때 1의 개수가 짝수나 홀수로 개수를 조정하여 마지막에 1bit를 더해서 전송하는 방식이다. 만약 0x0a를 전송하고 패리티를 짝수 패리티로 설정할 경우 0x0a를 이진수로 변경하면 0b00001010으로 1의 개수가 2개 이므로 짝수가 된다. 이때 패리티는 “0” 으로 설정된다. 만약 0x1a일 경우 이진수로 표기 하면 0b00011010 으로 1의 개수가 3개가 된다. 이때 패리티는 짝수를 맞추기 위해서 마지막에 1을 더 붙여서 보내게 된다. 현재에는 패리티를 쓰지 않기 때문에 bit5번과 bit4번은 각각 “0” 으로 설정한다.

bit3은 Stop비트와 관련된 비트이다. Stop비트는 데이터 전송이 완료된 후 끝에 1을 붙여서 보내는 통신 데이터 전송완료를 나타내는 비트로 1비트를 전송하는 방법과 2비트를 전송하는 방법 두 가지로 나누어진다. Stop비트는 1비트를 한다. 그러므로 “0” 이 된다. bit2~1은 UCSR0B의 bit2와 연결된다. 표를 보면 전송할 데이터의 비트수를 정의 할 수 있다. 특별한 경우를 제외 하고는 8bit를 거의 대부분에서 장치나 시스템에서 사용된다. UCSZ12에는 “0”, UCSZ11에는 “1”, UCSZ10에는 “1” 을 설정한다. 최종적으로 UCSR0C에 0x06으로 설정한다.

예제를 통해 실질적으로 동작해 본다. 아래와 같이 프로그램을 입력한다.

```
1. #include <avr/io.h>
2. #include <avr/iom128.h>
3. #include <avr/interrupt.h>
4. #include <avr/signal.h>
5. SIGNAL(SIG_UART0_RECV)
```

```
6.  {
7.    usinged char uart_buffer;
8.    uart_buffer = UDR0;
9.    PORTB = ~PORTB;
10. }
11. void main(void)
12. {
13.   cli();
14.   DDRE = 0x20;
15.   DDRB = 0x80;
16.   UBRR0H = 0;
17.   UBBR0L = 7;
18.   UCSR0A = 0x20;
19.   UCSR0B = 0x98;
20.   UCSR0C = 0x06;
21.   sei();
22.   while(1)
23.   {
24.     while((UCSR0A & 0x20) == 0x00);
25.     UDR0 = 0x30;
26.   }
27. }
```

설명

5. : 시리얼통신 데이터 수신이 완료되면 인터럽트가 요청되고 함수로 프로그램이 점프하여 { }안의 구문을 수행하게 된다.

7. : 캐릭터형의 uart_buffer의 지역변수를 할당한다.

8. : uart_buffer의 변수에 UDR0의 데이터를 옮긴다. 이때 수신완료와 관련된 레지스터들이 초기화 되어 새로운 데이터를 수신할 수 있다. 만약 이때 다른 데이터가 수신되어 옮기기 전에 데이터가 수신되면 데이터 Overflow 레지스터 비트가 Set 되어 검사를 할 수 있다. 인터럽트 방식을 사용하였기 때문에 수신이 완료된 상태에서 데이터를 바로 옮기기 때문에 검사를 할 필요가 없다.

9. : PORTB의 출력을 반전하여 출력한다.

14. : DDRE에 0x20을 설정하였다. USART포트 “RXD0”가 PORTE 0번에 “TXD0”가 PORTE 1번에 연결되어 있기 때문에 PORTE 0번은 입력으로 PORTE 1번은 출력으로 설정하였다.

15. : DDRB 에 0x80을 설정하여 PORTB 7번에 연결되어 있는 LED를 활성화 한다.

16, 17. : USART0를 사용하기 위하여 제일 먼저 Baud Rate를 설정한다. 설정 값은 14.7456MHz를 기준으로 UCSR0A의 1번 비트 U2X1의 설정 값을 “0” 으로 설정하였고, 표에 의해서 10진수 7이라는 값을 하위 바이트에 설정한다. 만약 설정 표에서 400이라는 값을 입력해야 한다면, 16진수로 바꾸어 상위 값은 UBBR0H에 하위 값은 UBBT0L에 설정해야 한다.

18. : UCSR0A에 0x20을 설정하여 UDRE0에 “1” 로 설정하여 UDR0에 데이터가 없음을 설정한다.

19. : UCSR0B에 0x98을 설정하여 수신완료시 인터럽트 사용, 송수신을 동시에 사용하겠다고 설정한다.

20. : UCSR0C에 0x06을 설정하여 비동기 통신, 패리티 비트 없음, Stop비트 1비트, 전송데이터 Size 8비트로 설정한다.

24. : while문을 사용하여 루프조건을 발생시킨다. 발생조건은 UCSR0A와 0x20을 엔드 연산하여 “0” 과 같으면 참인 조건으로 루프를 수행하게 되고 만약 참이 아니면 루프를 빠져 나오게 된다. 이렇게 조건을 걸어서 검사 하는 이유는 마이크로프로세서 수행시간이 빠르기 때문에 UDR0에 데이터를 중복해서 써버리면 데이터가 전송하기 전에 UDR0의 값을 바꿔 버리는 현상으로 데이터가 전송 중에 겹쳐 버리게 된다. 이러한 점을 없애기 위해서 사용자가 데이터를 검사하여 전송할 수 있도록 UCSR0A의 5번 비트가 전송이 완료되고 UDR0가 비어 있으면 “1” 로 설정 되게 한다. 바로 이점을 이용하여 UDR0에 데이터를 쓰고, 전송이 되기 전까지 기다리거나 프로그램에서 UDR0에 데이터를 써야할 때 다른 쪽 프로그램에서 만약 데이터를 전송하기 위해서 쓴 상태에 바로 써버리면 중복되어서 온전한 데이터 전송이 불가능하여 검사를 한다.

25. : 검사를 마치고 UDR0에 0x30이라는 데이터를 전송하라는 의미이다. 이때 UDR0에는 데이터가 존재 하고 전송 쉬프트 레지스터에서 전송을 메인 클록 동기에 맞춰서 스스로 전송하고 사용자의 프로그램은 다른 작업을 전송 중에 수행할 수 있으며, 전송이 완료되면 인터럽트를 발생시켜 수신과 마찬가지로 특정 함수로 빠뜨릴 수 있지만 여기에서는 인터럽트를 발생시키지 않고 전송할 때 검사를 하는 방법을 사용하여 전송하였다. 0x30은 ASCII코드에서 숫자 “0” 을 의미한다.

USART0을 이용하여 레지스터를 설정하고 동작시켜보았다. UDR0에 0x30이라는 데이터를 전송하였지만 전송 상대가 PC가 될 수도 있고, 다른 마이크로 컨트롤러가 될 수도 있다. 기본적으로 C언어는 ASCII코드를 이용하여 문자를 전송할 수 있는 시스템 구비 하였다. 만약 ASCII데이터 전송목적이 아니라면 사용자가 원하는 데이터를 전송하여도 상관없지만 문자를 전송해야 한다면, ASCII데이터를 전송하여 데이터를 보내야 한다. 일반적으로 PC에서 사용되는 방식은 ASCII코드 방식으로 텔넷이

나 모뎀등과 통신을 할 때 일반적인 16진수의 HEX데이터를 전송하면 이상한 문자로 깨져서 나오게 됨을 알 수 있다. ASCII코드는 부록을 참조 한다.

USART0번의 통신을 이용하여 컴퓨터에서 'b'를 입력 받을 때마다 LED를 On, Off를 반복하는 프로그램을 아래 조건에 맞추어 전 프로젝트의 소스를 주석 처리한 후에 작성한다.

- 시리얼통신 포트0 사용
- 전송속도 115200bps
- 스탑비트 1비트
- 패리티비트 없음
- 데이터비트 8비트
- 송수신 동시에 가능
- 송신인터럽트 사용안함, 수신인터럽트 사용함

```
1.  #include <avr/io.h>
2.  #include <avr/iom128.h>
3.  #include <avr/interrupt.h>
4.  #include <avr/signal.h>
5.  SIGNAL(SIG_UART0_RECV)
6.  {
7.     usinged char uart_buffer;
8.     uart_buffer = UDR0;
9.     if(uart_buffer == ' b')
10.    {
11.       PORTB = ~PORTB;
12.    }
13. }
14. void main(void)
15. {
16.    cli();
17.    DDRE = 0x20;
18.    DDRB = 0x80;
19.    UBRR0H = 0;
20.    UBBR0L = 7;
21.    UCSR0A = 0x20;
```

```
22.    UCSR0B = 0x98;
23.    UCSR0C = 0x06;
24.    sei();
25.    while(1)
26.    {
27.    }
28. }
```

설명

5. : USART0가 수신이 완료되어 인터럽트가 발생하면 이곳으로 점프하여 { }의 프로그램이 수행된다.
7. : 캐릭터형태의 변수 uart_buffer를 만든다.
8. : 만들어진 캐릭터 형 변수에 UDR0의 값을 이동시킨다.
9. : 만약 uart_buffer에 캐릭터 형 'b'와 같다면 아래{ }를 수행한다. 데이터 수신시 HEX값으로 8비트를 수신하지만 C언어에서 ASCII값을 받아들이고 문자를 쓸 수 있도록 ' '를 써서 문자를 검색하도록 하였다. 이때 b는 소문자 영어 b이므로 0x63이라는 숫자로 대입된다.
11. : PORTB의 출력을 반전 시킨다.
17. : DDRE에 0x20을 설정한다. USART포트 "RXD0"가 PORTE 0번에 "TXD0"가 PORTE 1번에 연결되어 있기 때문에 PORTE 0번은 입력으로 PORTE 1번은 출력으로 설정한다.
18. : DDRB 에 0x80을 설정하여 PORTB 7번에 연결되어 있는 LED를 활성화 한다.
19, 20. : USART0을 사용하기 위하여 제일 먼저 Baud Rate를 설정한다. 설정 값은 14.7456MHz를 기준으로 UCSR0A의 1번 비트 U2X1의 설정 값을 "0"으로 설정하였고, 표에 의해서 10진수 7이라는 값을 하위 바이트에 설정한다. 만약 설정 표에서 400이라는 값을 입력해야 한다면, 16진수로 바꾸어 상위 값은 UBBR0H에 하위 값은 UBBT0L에 설정해야 한다.
21. : UCSR0A에 0x20을 설정하여 UDRE0에 "1"로 설정하여 UDR0에 데이터가 없음을 설정한다.
22. : UCSR0B에 0x98을 설정하여 수신완료시 인터럽트 사용, 송수신을 동시에 사용하겠다고 설정한다.
23. : UCSR0C에 0x06을 설정하여 비동기 통신, 패리티 비트 없음, Stop비트 1비트, 전송데이터 Size 8비트로 설정하였다.

응용예제로 1초마다 동작하는 타이머를 Timer/Counter0 서비스를 이용해 만들고, 매 1초일 때 모니터 상에 "1second"라는 문구를 발생 시킨다. 이때 USART설정은 아래 기준에 따른다.

- 시리얼통신 포트0 사용

- 전송속도 115200bps
- 스탑비트 1비트
- 패리티비트 없음
- 데이터비트 8비트
- 송수신 동시에 가능
- 송신인터럽트 사용안함, 수신인터럽트 사용함

```
1.  #include <avr/io.h>
2.  #include <avr/iom128.h>
3.  #include <avr/interrupt.h>
4.  #include <avr/signal.h>
5.  unsigned int time_interval;
6.  SIGNAL(SIG_UART0_RECV)
7.  {
8.          unsigned char uart_buffer;
9.          uart_buffer = UDR0;
10. }
11. SIGNAL(SIG_OVERFLOW0)
12. {
13.         TCNT0 = 0xff - 115;
14.         time_interval++;
15. }
16. int main(void)
17. {
18.         cli();
19.         DDRE = 0x20;
20.         UBRR0H = 0;
21.         UBBR0L = 7;
22.         UCSR0A = 0x20;
23.         UCSR0B = 0x98;
24.         UCSR0C = 0x06;
25.         TCCR0 = 0x06;
26.         TCNT0 = 0xff - 115;
27.         sei();
28.         TIMSK = 0x01;
```

```
29.   while(1)
30.   {
31.         while(time_interval <= 500);
32.         while((UCSR0A & 0x20) == 0x00);
33.         UDR0 = '1'; //-15
34.         while((UCSR0A & 0x20) == 0x00);
35.         UDR0 = 's'; //-17
36.         while((UCSR0A & 0x20) == 0x00);
37.         UDR0 = 'e'; //-19
38.         while((UCSR0A & 0x20) == 0x00);
39.         UDR0 = 'c'; //-21
40.         while((UCSR0A & 0x20) == 0x00);
41.         UDR0 = 'o'; //-23
42.         while((UCSR0A & 0x20) == 0x00);
43.         UDR0 = 'n'; //-25
44.         while((UCSR0A & 0x20) == 0x00);
45.         UDR0 = 'd';
46.         time_interval = 0;
47.   }
48. }
```

설명

5. : time_interval이라는 int 형 변수를 만든다.
6. : USART0 수신완료 시 수신인터럽트 발생하고, 점프하여 { } 안의 수행한다.
11. : Timer/Counter0 서비스 인터럽트 발생 시 수행중문 작업을 중단하고 { }안을 수행한다.
14. : time_interval 값을 증가시킨다.
20~24. : 시리얼통신 비동기 115200bps의 속도 Stop비트1 비트 패리티 비트 없음 데이터 크기 8비트, 송수신 동시에 사용, 수신인터럽트 사용의 설정이다.
25~28. : Timer/Counter0을 256분주에 카운터 값 256으로 설정하여 2ms의 타이머 카운터 이벤트를 TIMSK에 0x01로 설정하여 활성화 한다.
31. : time_interval의 값이 500이하 일 때 참 조건이므로 루프를 수행한다. 하지만 500과 같거나 높아지면 거짓의 조건이 되므로 아래 프로그램을 수행한다. 500이라는 time_interval은 타이머서비스 인터럽트 루틴 발생시 11번으로 점프하고 14번에서 증가한다. 즉 타이머서비스 인터럽트 루틴에서는 증가만 시키고 메인함수 내의 while문에서 검사한다.
32. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다.
33. : UDR0에 '1' 이라는 ASCII값을 써준다. 코드 값은 0x31이다.
34. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다. 현재 조

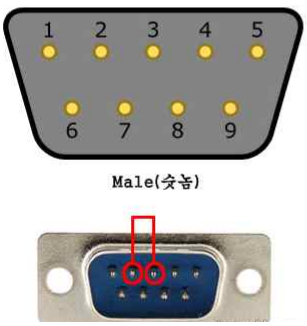
건에서는 33번에서 UDR0에 데이터를 쓰고 바로 내려 왔기 때문에 검사조건이 초기에는 참으로 루프에 머물게 될 것이다. 하지만 '1'의 전송이 완료되면 검사 조건이 거짓이 되면서 아래 구문을 수행한다.

35. : UDR0에 's'이라는 ASCII값을 써준다. 코드 값은 0x73이다.
36. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다.
37. : UDR0에 'e'이라는 ASCII값을 써준다. 코드 값은 0x65이다.
38. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다.
39. : UDR0에 'c'이라는 ASCII값을 써준다. 코드 값은 0x63이다.
40. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다.
41. : UDR0에 'o'이라는 ASCII값을 써준다. 코드 값은 0x6f이다.
42. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다.
43. : UDR0에 'n'이라는 ASCII값을 써준다. 코드 값은 0x6e이다.
44. : USART0의 전송 버퍼 UDR0의 데이터를 쓸 수 있는지 검사 한다.
45. : UDR0에 'n'이라는 ASCII값을 써준다. 코드 값은 0x64이다.
46. : time_interval값을 "0"으로 초기화 하여, 다음 또 메인함수의 루프가 돌 때 다시 1초를 만들어주기 위한 시간을 검사 할 수 있도록 시간을 초기화 해준다.

나. 하이퍼터미널을 이용하여 로봇 제어하기

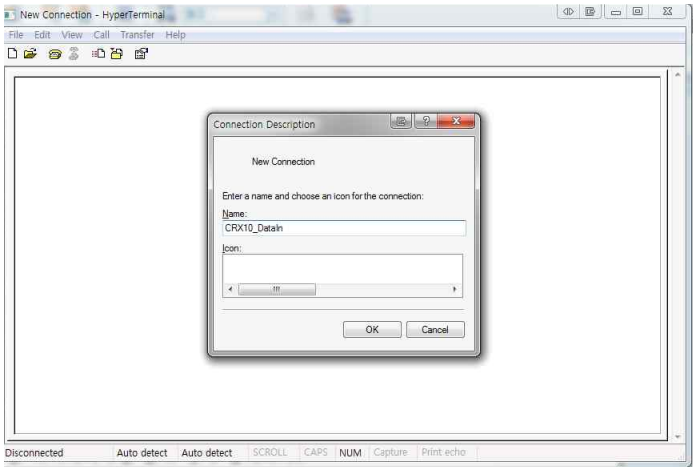
하이퍼터미널이란 프로그램은 Serial 통신을 지원하며 TCP/IP도 지원한다. 윈도우 기반의 C++ 또는 C#으로 통신프로그램을 개발하여 로봇과의 통신테스트를 진행하지만 먼저 간단한 기본적인 테스트는 보통 하이퍼터미널을 이용한다. Window XP OS는 하이퍼터미널이 기본프로그램으로 등록되어있어 바로 사용할 수 있지만 Window 7에서는 제공하지 않아 외부에서 다운받아 사용한다.

로봇을 사용하기에 앞서 먼저 준비해야할 것은 USB To Serial 케이블 또는 사용자의 컴퓨터에 Serial 단자가 존재한다면 Serial To Serial 케이블을 준비하여 로봇에 연결한다. 단, 그림 8.18과 같이 Serial 단자는 수놈이어야 한다.



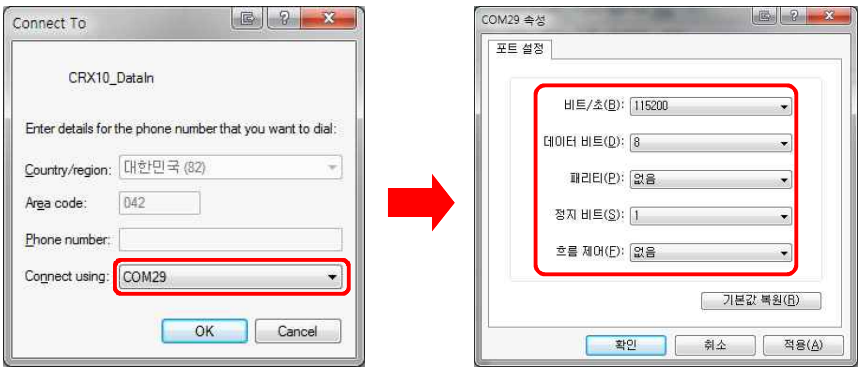
[그림 8.18] Serial 단자

하이퍼터미널 프로그램을 실행하면 그림 8.19와 같이 나타나며, 로봇과 커넥션하기 위해 커넥션 이름을 입력한다.



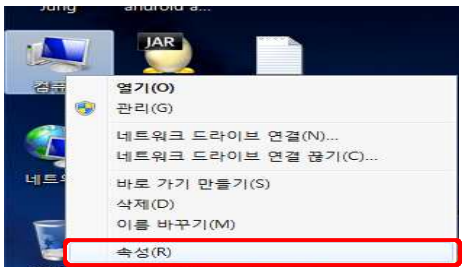
[그림 8.19] 하이퍼터미널 프로그램

그림 8.20과 같이 현재 연결된 Serial 케이블의 ComPort를 선택하고 통신 속도 외에 설정 값을 선택하여 “확인” 버튼을 클릭한다.



[그림 8.20] 하이퍼터미널 프로그램 커넥션 설정

만약 연결된 케이블의 ComPort를 모른다면 그림 8.21과 같이 오른쪽 클릭한다.

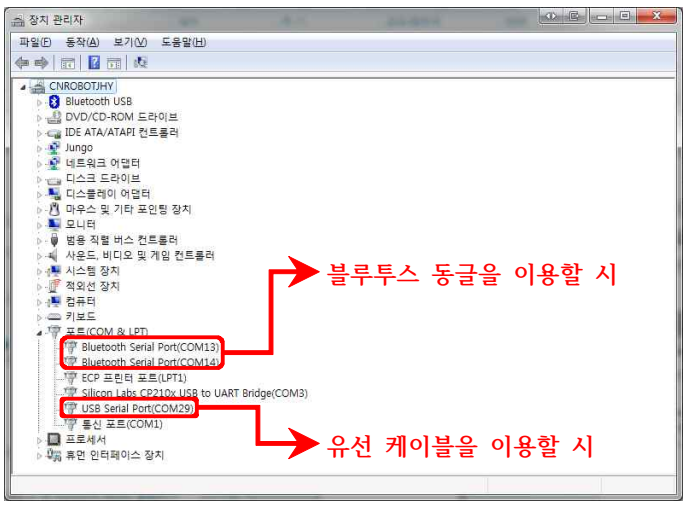


[그림 8.21] 내 컴퓨터 오른쪽 클릭

다음으로 그림 8.22와 같이 “장치 관리자”를 클릭한다.

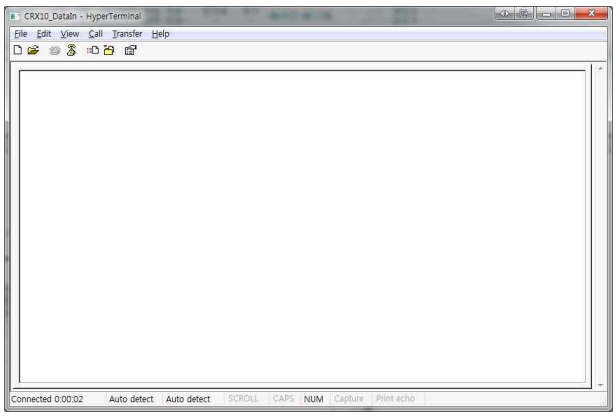
[그림 8.22] 내 컴퓨터 오른쪽 클릭

장치 관리자를 보면 포트(COM & LPT)가 있으며, 활성화 시키면 그림 8.23과 같이 나타난다. CRX10에는 블루투스가 장착되어있어 컴퓨터에 블루투스 동글을 연결하여 사용해도 무방하다.



[그림 8.23] 내 컴퓨터 오른쪽 클릭

그림 8.24와 같이 연결이 완료된 모습을 볼 수 있다.



[그림 8.24] 내 컴퓨터 오른쪽 클릭

간단한 Serial 통신 예제를 이용하여 로봇을 제어해 보자.

1) 활용 실습

Q : 8.1_ 문자 'A' 를 하이퍼터미널 출력 창에 출력하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

volatile unsigned char u8_make_time_var;
unsigned char u8_usart_receive_data;


SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
```

```
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}
void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
}

int main(void)
{
        unsigned char uartdata;
        cli();
        port_initial();
        basic_timer_initial();
        uart_initial();
        sei();
```

```
        make_time();
        make_time();
        make_time();
        uartdata='A';

        while(1)
        {
                usart_tx_data(uartdata);
                make_time();
        }
}
```

Q : 8.2_ 3개의 PSD센서의 데이터를 하이퍼 터미널에 출력하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

volatile unsigned char u8_make_time_var;
unsigned char u8_usart_receive_data;
unsigned char adc_result[3];

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}
```

```
void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void adc_initial(void)
{
        ADCSRA = 0x86;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
```

```
}
void adc_read(void)
{
        ADCSRA &= 0x7f;
        ADMUX = 0x20;
        ADCSRA |= 0xc0;
        while((ADCSRA & 0x10) != 0x10);
        adc_result[0] = ADCL;
        adc_result[0] = ADCH;
}

void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
}

int main(void)
{
        unsigned char uartdata;
        cli();
        port_initial();
        basic_timer_initial();
        uart_initial();
        sei();
        make_time();
        make_time();
        make_time();
        uartdata=',';

        while(1)
        {
                adc_read();
                usart_tx_data(adc_result[0]);
                usart_tx_data(uartdata);
                usart_tx_data(adc_result[1]);
                usart_tx_data(uartdata);
```

```
                usart_tx_data(adc_result[2]);
                usart_tx_data(uartdata);
                make_time();
        }
}
```

Q : 8.3_ PSD 센서의 데이터를 Cm로 변환하여 하이퍼 터미널 출력 창에 출력하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

volatile unsigned char u8_make_time_var;
unsigned char u8_usart_receive_data;
unsigned char adc_result[3];

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
```

```
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void adc_initial(void)
{
        ADCSRA = 0x86;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void adc_read(void)
{
        ADCSRA &= 0x7f;
```

```
        ADMUX = 0x20;
        ADCSRA |= 0xc0;
        while((ADCSRA & 0x10) != 0x10);
        adc_result[0] = ADCL;
        adc_result[0] = ADCH;

        ADCSRA &= 0x7f;
        ADMUX = 0x21;
        ADCSRA |= 0xc0;
        while((ADCSRA & 0x10) != 0x10);
        adc_result[1] = ADCL;
        adc_result[1] = ADCH;

        ADCSRA &= 0x7f;
        ADMUX = 0x22;
        ADCSRA |= 0xc0;
        while((ADCSRA & 0x10) != 0x10);
        adc_result[2] = ADCL;
        adc_result[2] = ADCH;
}

void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
}

double PSDADC(double adc)
{
    double k, returnk;
    int k1;
    k = 5 * adc / 255;
    if (k >= 1.3)
    {
        k1 = (int)(-(k - 3.87) * 14 / 1.8);
    }
    else if (k >= 0.9 && k < 1.3)
```

```
    {
        k1 = (int)(-(k - 2.1) * 10 / 0.4);
    }
    else if (k >= 0.5 && k < 0.9)
    {
        k1 = (int)(-(k - 1.3) * 30 / 0.4);
    }
    else if (k >= 0.4 && k < 0.5)
    {
        k1 = (int)(-(k - 0.95) * 20 / 0.15);
    }
    else
        k1 = 80;

    returnk = (double)k1;

    return returnk;
}

int main(void)
{
        unsigned char uartdata;
        cli();
        port_initial();
        basic_timer_initial();
        uart_initial();
        sei();
        make_time();
        make_time();
        make_time();
        uartdata=',';
        while(1)
        {
                adc_read();
                usart_tx_data((int)PSDADC(adc_result[0]));
                usart_tx_data(uartdata);
                usart_tx_data((int)PSDADC(adc_result[1]));
```

```
                usart_tx_data(uartdata);
                usart_tx_data((int)PSDADC(adc_result[2]));
                usart_tx_data(uartdata);
                make_time();
        }
}
```

Q : 8.4_ 로봇에 장착된 4개의 버튼을 ‘1’, ‘2’, ‘3’, ‘4’로 지정하여 버튼을 눌렀을 때 하이퍼터미널 출력 창에 출력하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

volatile unsigned char u8_make_time_var;
unsigned char u8_usart_receive_data;
unsigned char u8_io_read;

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
        DDRA = 0xff;
```

```
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void adc_initial(void)
{
        ADCSRA = 0x86;
}

void port_read(void)
{
        u8_io_read = PIND & 0xf0;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
```

```
        UDR0 = tx_data;
}
void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
}

int main(void)
{
        unsigned char uartdata;
        cli();
        port_initial();
        basic_timer_initial();
        uart_initial();
        sei();
        make_time();
        make_time();
        make_time();
        uartdata='A';

        while(1)
        {
                port_read();
                usart_tx_data(u8_io_read);
                make_time();
        }
}
```

Q : 8.5_ 키보드에서 ‘C’를 입력하며 로봇의 도트매트릭스에서 ‘C’를 출력하기

A :

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

```
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include<util/delay.h>

volatile unsigned char u8_make_time_var;
unsigned char u8_usart_receive_data;
unsigned char u8_io_read;

#define dot_O 0
#define dot_X 1
#define dot_C 2

unsigned char ccc[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
unsigned char row[3][8] = {{0x3C,0x7E,0xC3,0xC3,0xC3,0xC3,0x7E,0x3C},
                           {0x81,0xC3,0x66,0x3C,0x3C,0x66,0xC3,0x81},
                           {0x3C,0x7E,0xC3,0xC3,0xC3,0xC3,0x66,0x66} };

void print(int n,int time){
        int i,l;
        for(l=0;l<time;l++){
                for(i=0;i<8;i++){
                        PORTA=ccc[i];
                        PORTC=row[n][i];
                        _delay_ms(1);}
        }
}

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
```

```
}

void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}
void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void adc_initial(void)
{
        ADCSRA = 0x86;
}

void port_read(void)
{
        u8_io_read = PIND & 0xf0;
}
```

```
void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
}

int main(void)
{
        unsigned char uartdata;
        cli();
        port_initial();
        basic_timer_initial();
        uart_initial();
        sei();
        make_time();
        make_time();
        make_time();
        uartdata='A';

        while(1)
        {
                if(u8_usart_receive_data=='C')
                {
                        print( dot_C, 500);

                }
                make_time();
        }
}
```

Q : 8.6_ 키보드의 방향키 중 ‘W’를 누르면 로봇이 전진하게 하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include<util/delay.h>
volatile unsigned char u8_make_time_var;
volatile unsigned int cnt;
unsigned char adc_result[8];
unsigned char u8_usart_receive_data;

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}
void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}
```

```
void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void pwm_initial(void)
{
        TCCR1A = 0xa2;
        TCCR1B = 0x19;
        TCCR1C = 0x00;
        ICR1 = 200;
        OCR1A = 0;
        OCR1B = 0;
        OCR1C = 0;

        TCCR3A = 0xc2;
        TCCR3B = 0x1a;
        TCCR3C = 0x00;
        ICR3 = 1000;
        OCR3A = 500;
        OCR3B = 100;
        OCR3C = 0;
}

void usart_tx_data(unsigned char tx_data)
{
```

```
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void motor_r_foward(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x80;
}

void motor_r_back(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x40;
}

void motor_l_foward(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x08;
}

void motor_l_back(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x10;
}

void Motor_control_Forwards(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_foward();
        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Backwards(unsigned char right, unsigned char left)
```

```
{
	motor_r_back();
	motor_l_back();

	OCR1A = right;
	OCR1B = left;
}

void Motor_control_Left(unsigned char right, unsigned char left)
{
	motor_r_foward();
	motor_l_back();

	OCR1A = right;
	OCR1B = left;
}

void Motor_control_Right(unsigned char right, unsigned char left)
{
	motor_r_back();
	motor_l_foward();

	OCR1A = right;
	OCR1B = left;
}

void make_time(void)
{
	while(u8_make_time_var <= 50){}
	u8_make_time_var = 0;
	cnt++;
}

int main(void)
{
	cli();
	port_initial();
```

```
	basic_timer_initial();
	pwm_initial();
	uart_initial();
	sei();
	make_time();
	make_time();


	while(1)
	{
		if(u8_usart_receive_data=='W')
		{
			Motor_control_Forwards(130,130);

		}
		else
		{
			Motor_control_Forwards(0,0);
		}

		make_time();
	}
}
```

Q : 8.7_ 키보드의 방향키 중 'S'를 누르면 로봇이 후진하게 하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include<util/delay.h>
volatile unsigned char u8_make_time_var;
volatile unsigned int cnt;
unsigned char adc_result[8];
unsigned char u8_usart_receive_data;
```

```
SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}
```

```
void pwm_initial(void)
{
        TCCR1A = 0xa2;
        TCCR1B = 0x19;
        TCCR1C = 0x00;
        ICR1 = 200;
        OCR1A = 0;
        OCR1B = 0;
        OCR1C = 0;

        TCCR3A = 0xc2;
        TCCR3B = 0x1a;
        TCCR3C = 0x00;
        ICR3 = 1000;
        OCR3A = 500;
        OCR3B = 100;
        OCR3C = 0;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void motor_r_foward(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x80;
}

void motor_r_back(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x40;
}
```

```
void motor_l_foward(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x08;
}
void motor_l_back(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x10;
}

void Motor_control_Forwards(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_foward();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Backwards(unsigned char right, unsigned char left)
{
        motor_r_back();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Left(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
```

```
}

void Motor_control_Right(unsigned char right, unsigned char left)
{
        motor_r_back();
        motor_l_foward();

        OCR1A = right;
        OCR1B = left;
}

void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
        cnt++;
}

int main(void)
{
        cli();
        port_initial();
        basic_timer_initial();
        pwm_initial();
        uart_initial();
        sei();
        make_time();
        make_time();

        while(1)
        {
                if(u8_usart_receive_data=='S')
                {
                        Motor_control_Backwards(130,130);

                }
                else
```

```
                {
                        Motor_control_Forwards(0,0);
                }

                make_time();
        }
}
```

Q : 8.8_ 키보드의 방향키 중 'A'를 누르면 로봇이 좌로 회전하게 하기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include<util/delay.h>
volatile unsigned char u8_make_time_var;
volatile unsigned int cnt;
unsigned char adc_result[8];
unsigned char u8_usart_receive_data;

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
```

```
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void pwm_initial(void)
{
        TCCR1A = 0xa2;
        TCCR1B = 0x19;
        TCCR1C = 0x00;
        ICR1 = 200;

        OCR1A = 0;
        OCR1B = 0;
        OCR1C = 0;

        TCCR3A = 0xc2;
```

```
        TCCR3B = 0x1a;
        TCCR3C = 0x00;
        ICR3 = 1000;
        OCR3A = 500;
        OCR3B = 100;
        OCR3C = 0;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void motor_r_foward(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x80;
}

void motor_r_back(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x40;
}

void motor_l_foward(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x08;
}

void motor_l_back(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x10;
}
```

```
void Motor_control_Forwards(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_foward();
        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Backwards(unsigned char right, unsigned char left)
{
        motor_r_back();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Left(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Right(unsigned char right, unsigned char left)
{
        motor_r_back();
        motor_l_foward();

        OCR1A = right;
        OCR1B = left;
}
void make_time(void)
{
```

```
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
        cnt++;
}

int main(void)
{
        cli();
        port_initial();
        basic_timer_initial();
        pwm_initial();
        uart_initial();
        sei();
        make_time();
        make_time();

        while(1)
        {
                if(u8_usart_receive_data=='A')
                {
                        Motor_control_Left(130,130);
                }
                else
                {
                        Motor_control_Forwards(0,0);
                }
                make_time();
        }
}
```

Q : 8.9_ 키보드의 방향키 중 'D'를 누르면 로봇이 우로 회전하게 하기

A :

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

```
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include<util/delay.h>
volatile unsigned char u8_make_time_var;
volatile unsigned int cnt;
unsigned char adc_result[8];
unsigned char u8_usart_receive_data;

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}
```

```
void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}

void pwm_initial(void)
{
        TCCR1A = 0xa2;
        TCCR1B = 0x19;
        TCCR1C = 0x00;
        ICR1 = 200;
        OCR1A = 0;
        OCR1B = 0;
        OCR1C = 0;

        TCCR3A = 0xc2;
        TCCR3B = 0x1a;
        TCCR3C = 0x00;
        ICR3 = 1000;
        OCR3A = 500;
        OCR3B = 100;
        OCR3C = 0;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void motor_r_foward(void)
{
```

```
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x80;
}

void motor_r_back(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x40;
}

void motor_l_foward(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x08;
}
void motor_l_back(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x10;
}
void Motor_control_Forwards(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_foward();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Backwards(unsigned char right, unsigned char left)
{
        motor_r_back();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
}
```

```
void Motor_control_Left(unsigned char right, unsigned char left)
{
	motor_r_foward();
	motor_l_back();

	OCR1A = right;
	OCR1B = left;
}

void Motor_control_Right(unsigned char right, unsigned char left)
{
	motor_r_back();
	motor_l_foward();

	OCR1A = right;
	OCR1B = left;
}

void make_time(void)
{
	while(u8_make_time_var <= 50){}
	u8_make_time_var = 0;
	cnt++;
}

int main(void)
{
	cli();
	port_initial();
	basic_timer_initial();
	pwm_initial();
	uart_initial();
	sei();
	make_time();
	make_time();
```

```
        while(1)
        {
                if(u8_usart_receive_data=='D')
                {
                        Motor_control_Right(130,130);
                }
                else
                {
                        Motor_control_Forwards(0,0);
                }

                make_time();
        }
}
```

Q : 8.10_ 키보드의 방향키를 조이스틱으로 활용하여 로봇을 자유롭게 움직이기

A :

```
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include<util/delay.h>
volatile unsigned char u8_make_time_var;
volatile unsigned int cnt;
unsigned char adc_result[8];
unsigned char u8_usart_receive_data;

SIGNAL(SIG_OVERFLOW0)
{
        TCNT0 = 0xff - 115;
        u8_make_time_var++;
```

```
}

SIGNAL(SIG_UART0_RECV)
{
        u8_usart_receive_data = UDR0;
}

void port_initial()
{
        DDRA = 0xff;
        DDRB = 0xf7;
        DDRC = 0xff;
        DDRD = 0x00;
        DDRE = 0xfb;
        DDRF = 0x00;
        DDRG = 0x00;
}

void basic_timer_initial(void)
{
        TCCR0 = 0x06;
        TCNT0 = 0xff - 115;
        TIMSK = 0x01;
}

void uart_initial(void)
{
        UBRR0H = 0;
        UBRR0L = 7;
        UCSR0A = (0<<RXC0) | (1<<UDRE0);
        UCSR0B = 0x98;
        UCSR0C = 0x06;
}
void pwm_initial(void)
{
        TCCR1A = 0xa2;
        TCCR1B = 0x19;
```

```
        TCCR1C = 0x00;
        ICR1 = 200;
        OCR1A = 0;
        OCR1B = 0;
        OCR1C = 0;

        TCCR3A = 0xc2;
        TCCR3B = 0x1a;
        TCCR3C = 0x00;
        ICR3 = 1000;
        OCR3A = 500;
        OCR3B = 100;
        OCR3C = 0;
}

void usart_tx_data(unsigned char tx_data)
{
        while((UCSR0A & 0x20) == 0x00);
        UDR0 = tx_data;
}

void motor_r_foward(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x80;
}

void motor_r_back(void)
{
        PORTE = PORTE & 0x3f;
        PORTE = PORTE ¦ 0x40;
}

void motor_l_foward(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x08;
```

```
}

void motor_l_back(void)
{
        PORTG = PORTG & 0xe7;
        PORTG = PORTG ¦ 0x10;
}
void Motor_control_Forwards(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_foward();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Backwards(unsigned char right, unsigned char left)
{
        motor_r_back();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Left(unsigned char right, unsigned char left)
{
        motor_r_foward();
        motor_l_back();

        OCR1A = right;
        OCR1B = left;
}

void Motor_control_Right(unsigned char right, unsigned char left)
{
        motor_r_back();
```

```
        motor_l_foward();

        OCR1A = right;
        OCR1B = left;
}
void make_time(void)
{
        while(u8_make_time_var <= 50){}
        u8_make_time_var = 0;
        cnt++;
}
```

2) 응용문제

① 시리얼통신을 이용하여 두 개의 숫자를 PC에서 전송하고, 마이크로 컨트롤러는 두 개의 수신 데이터의 차를 구해서 다시 PC에 전송하기

② 로봇에 장착된 4개의 버튼을 '1', '2', '3', '4'로 지정하여 버튼을 눌렀을 때 하이퍼터미널 출력 창에 출력하기

③ 키보드에서 'C'를 입력하며 로봇의 도트매트릭스에서 'C'를 출력하기

④ 키보드의 'W'를 누르면 로봇이 전진하게 하기

⑤ 키보드의 'S'를 누르면 로봇이 후진하게 하기

⑥ 키보드의 'A'를 누르면 로봇이 좌로 회전하게 하기

⑦ 키보드의 'D'를 누르면 로봇이 우로 회전하게 하기

⑧ 키보드의 활용하여 로봇을 자유롭게 움직이기