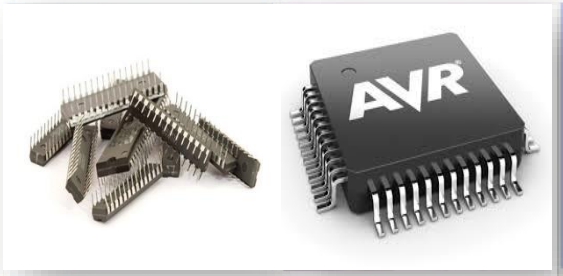
기업을 품는 산학융합 선도대학
한국산업기술대학교



2022
June

03

직렬 통신 포트의 동작

목차

- 직렬 통신(RS232C)의 개요
- ATmega128 직렬 포트의 개요
- 직렬 포트 제어용 레지스터
- USARTn의 동작
- 다중 프로세서 통신
- USARTn의 초기화 및 액세스
- USARTn 활용 실험

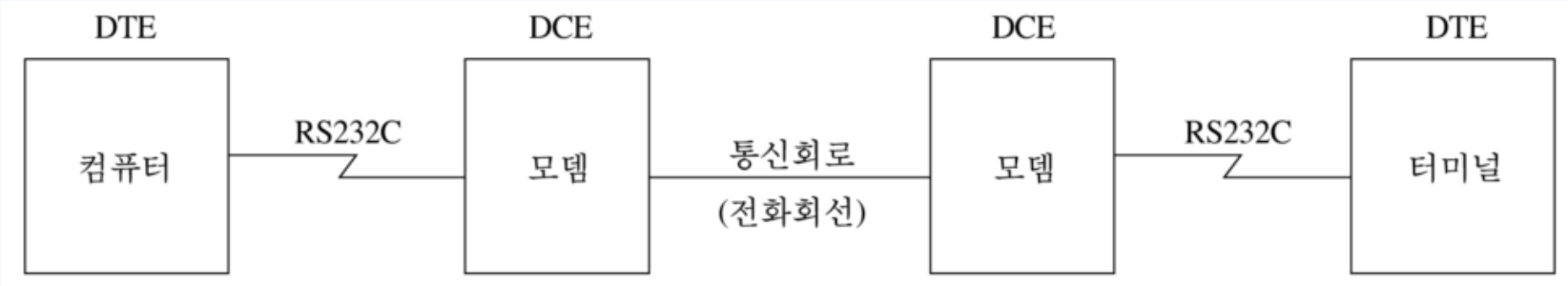
학습목표

- 직렬 통신(RS232C) 필요성 및 활용 방안의 이해
- ATmega128에서의 직렬 통신(RS232C)활용
- ATmega128에 내장된 USART 사용을 위한 레지스터의 이해를 통한 통신 기법 이해
- 외부 기기(PC 등)와의 데이터 송수신 기법(프로토콜 적용)

직렬통신(RS232C)의 개요

직렬 통신 방식

- 전송 방식에 따라 동기식과 비동기식으로 구분됨.
  - 동기식 통신 방식 : 기준 클럭에 동기를 맞추어 데이터를 순차적으로 전송하는 방식
  - 비동기식 통신 방식 : 동기 클럭없이 데이터의 전송 속도를 서로 정하여 데이터를 순차적으로 전송하는 방식
- RS232C : 1969년 미국 EIA에 의해 정해진 표준 인터페이스, 직렬 2진 패킷을 활용, 전기적인 특성, 기계적인 특성(커넥터 사양), 인터페이스 등으로 규정하고 있어서 현재 모뎀과 컴퓨터 주변장치와의 입/출력 인터페이스로서 널리 사용함.



〈 모뎀을 이용한 컴퓨터와 터미널의 접속 〉

직렬통신(RS232C)의 개요

RS232C 규격(25pin 규격 기준)

RS232C 신호 및 기능

핀 번호	명 칭	신호방향 DTE - DCE	기호
1	보안용 접지 (Frame Ground)	—	FG
2	송신 데이터(Transmitted Data)	→	TxD
3	수신 데이터(Receive Data)	←	RxD
4	송신 요구(Request to Send)	→	$\overline{RTS}$
5	송신 허가(Clear to Send)	←	$\overline{CTS}$
6	통신기기 세트 준비(Data Set Ready)	←	$\overline{DTR}$
7	신호용 접지(Signal Ground)	—	SG
8	캐리어 검출(Data Carrier Detect)	←	$\overline{DCD}$
20	데이터 단말 준비 (Data Terminal Ready)	→	$\overline{DSR}$
22	Ring Indicator	←	$\overline{RI}$

직렬통신(RS232C)의 개요

특징 및 사양

- 데이터단말기(DTE)와 데이터통신기(DCE) 사이의 인터페이스에 대한 규정
- 디지털논리 레벨로 원격에 떨어져 있는 장치로 데이터 를 정확하게 전송하기에는 전압 강하와 노이즈로 인하여 어렵다. 따라서 RS232C에서는 "1"의 값을 +12V를 "0"의 값으로 -12V를 사용 하여 신호간의 전압차를 24V로 만듬. 즉, 양단의 장치들 사이에서의 데이터 교환을 위해 전압을 레벨을 이용하는 데 TTL 레벨은 전압 편차가 작아 장거리 송/수신 시 노이즈에 취약 함.
- 통신거리 : 약 15m, 최대 1km(지원 케이블 사용시)
- 통신속도 : 최근 2Mbps 가능하나, 보편적으로 9.6Kbps,19.2Kbps,115.2Kbps를 사용함.

< RS232C의 신호 레벨 >

상 태	"L"	"H"
전압범위	-25V ~ -3V	+3V ~ +25V
논리	"1"	"0"
명칭	마크	스페이스

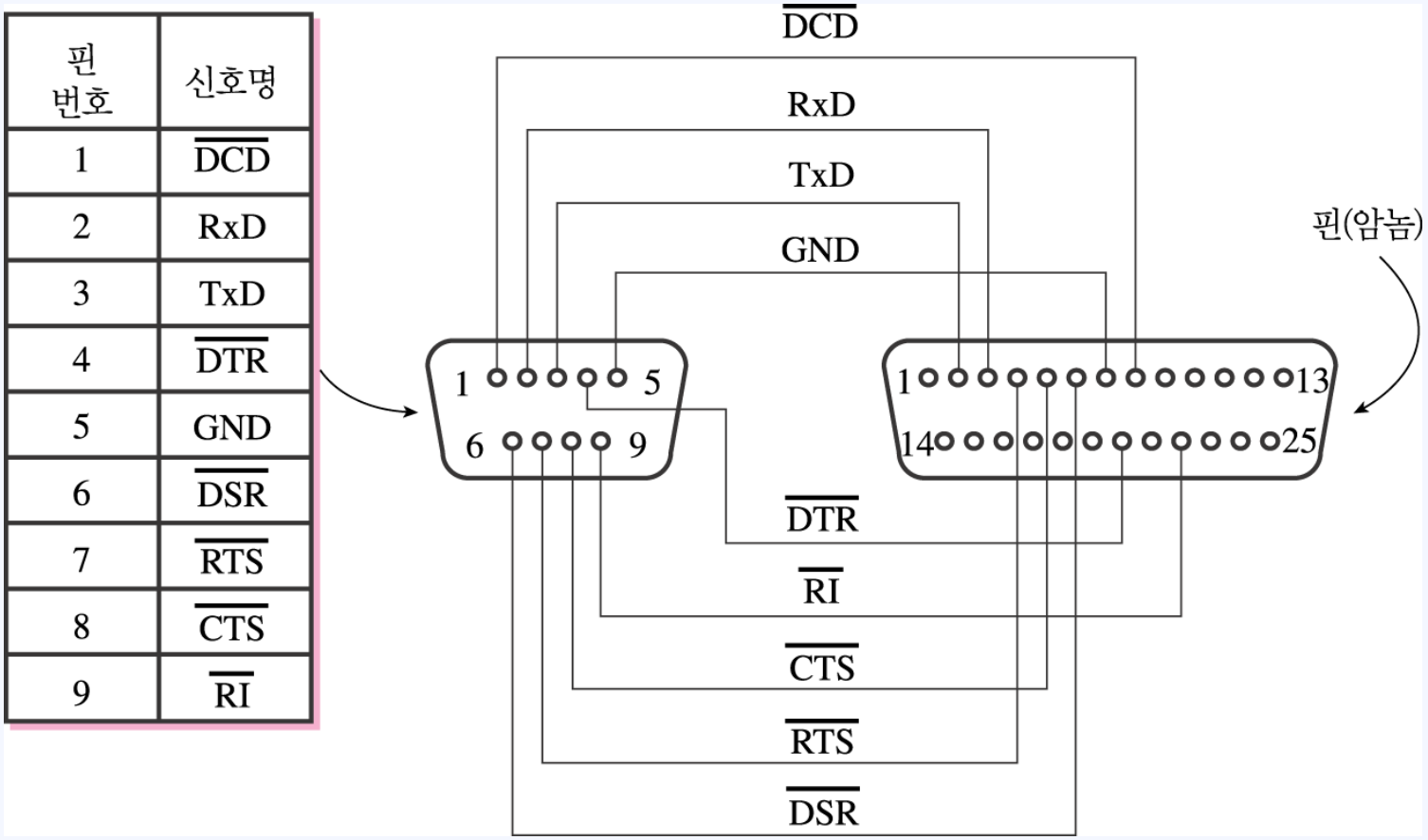
<RS232C의 전송 신호의 예>

DTE (Data Terminal Equipment)
DCE (Data Communications Equipment)

직렬통신(RS232C)의 개요

커넥터 및 신호선

핀 번호	신호명
1	$\overline{DCD}$
2	RxD
3	TxD
4	$\overline{DTR}$
5	GND
6	$\overline{DSR}$
7	$\overline{RTS}$
8	$\overline{CTS}$
9	$\overline{RI}$

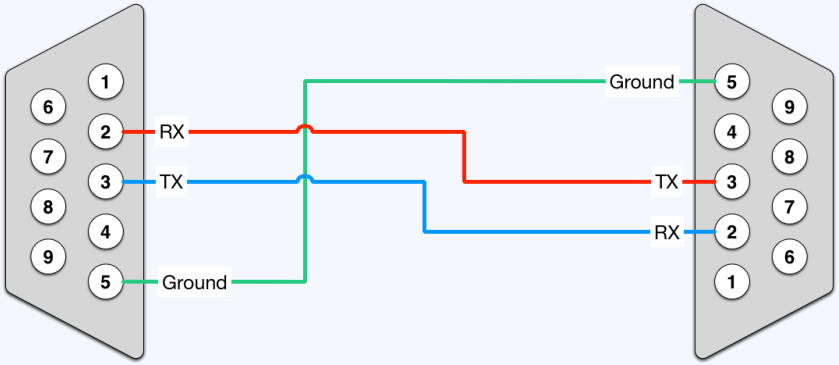


<9핀과 25핀의 신호 대응 관계>

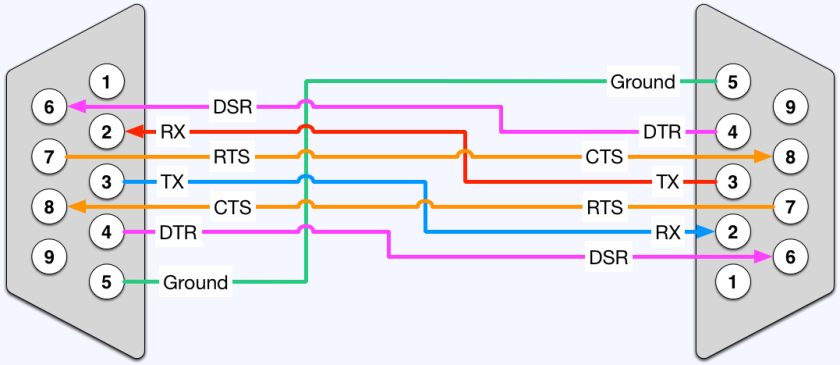
직렬통신(RS232C)의 개요

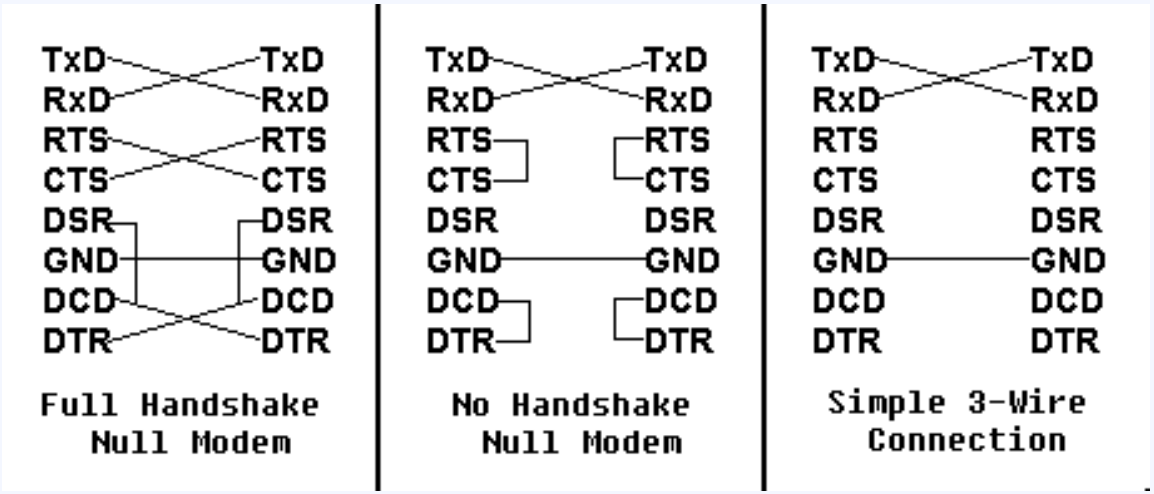
컴퓨터와의 신호 연결 - 9pin

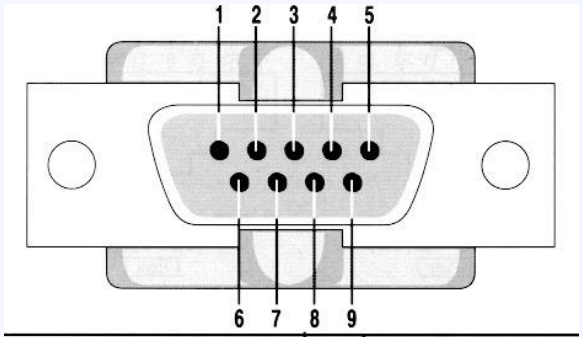
Simple Null Modem Cable



Null Modem Cable with Handshake







Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

직렬통신(RS232C)의 개요

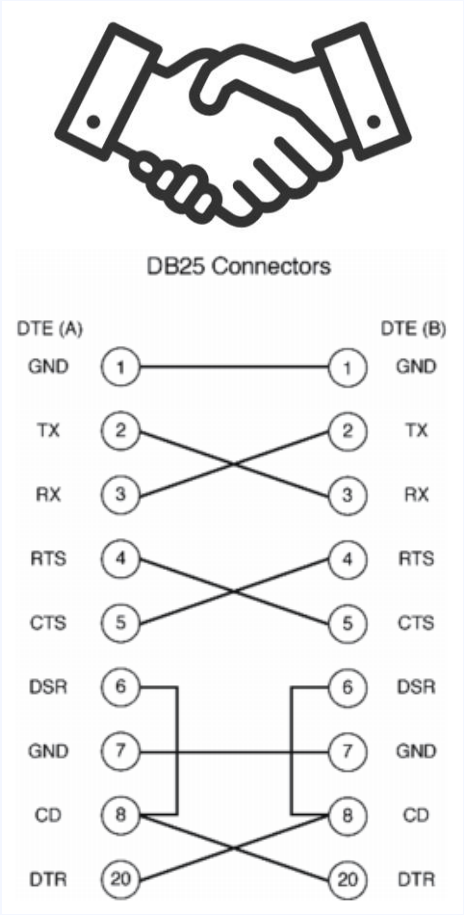
컴퓨터와의 신호 연결 (handshaking) - 25pin

의미 : RS232 통신시 상대 장비의 상태를 확인하고 상대 장비가 수신 가능한 상태일 경우를 확인 후 데이터를 전송 하는 방식

이유 : 수신측의 버퍼에서 처리 가능한 데이터 량 보다 많이 전송 시 수신 측 버퍼의 데이터 유실 및 지연을 방지 하기 위해

신호명	핀 번호	핀 번호	신호명
보안용 접지	①	①	보안용 접지
송신 데이터	②	②	송신 데이터
수신 데이터	③	③	수신 데이터
송신 요구	④	④	송신 요구
송신 허가	⑤	⑤	송신 허가
데이터 세트 준비	⑥	⑥	데이터 세트 준비
신호용 접지	⑦	⑦	신호용 접지
데이터 단말 준비	⑳	⑳	데이터 단말 준비

<핸드세이크 방식을 이용한 접속 방식>

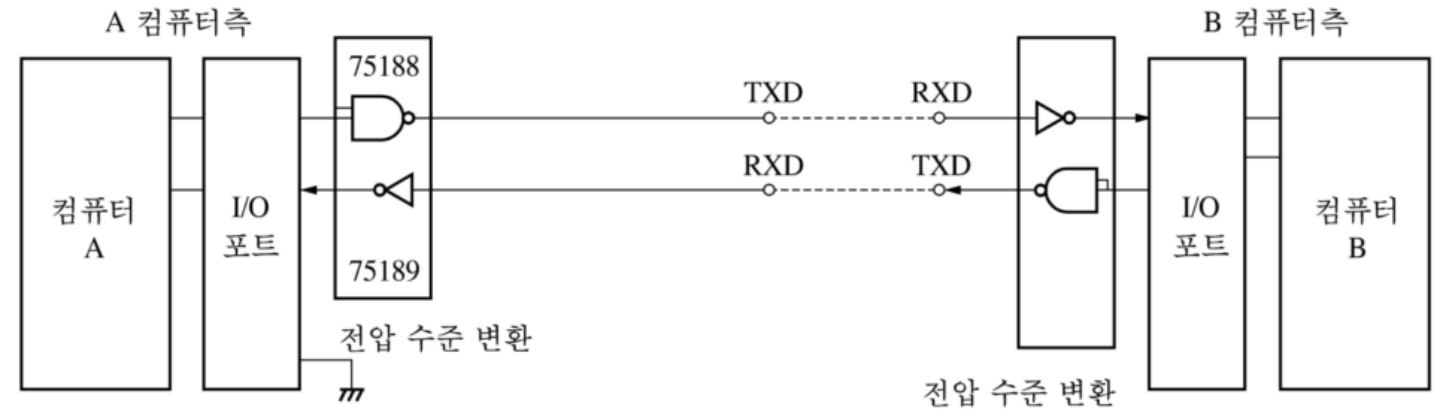


- **소프트웨어**의 제어 코드를 사용하여 전송 데이터의 **흐름을 제어하는 방식으로 보편적으로 XON/XOFF 제어 방식 사용**
  - **XON : 전송 시작을 의미하며 <Ctrl>+<S> 또는 ASCII 코드 0x11를 사용**
  - **XOFF : 전송 정지를 의미하며 <Ctrl>+<Q> 또는 ASCII 코드 0x13을 사용**

직렬통신(RS232C)의 개요

인터페이스 IC(1)

- TTL 신호를 EIA 레벨(RS232C 레벨)로 변환하기 위해 라인 드라이버 Line Receiver와 Line Driver IC 세트로 사용
  - SN75188, MC1488과 SN75189, MC1489
  - ±12V 전원 필요
- MAX232 : RS232C 신호를 송수신하기 위한 전용 Transceiver IC
  - +5V 전원만으로 사용

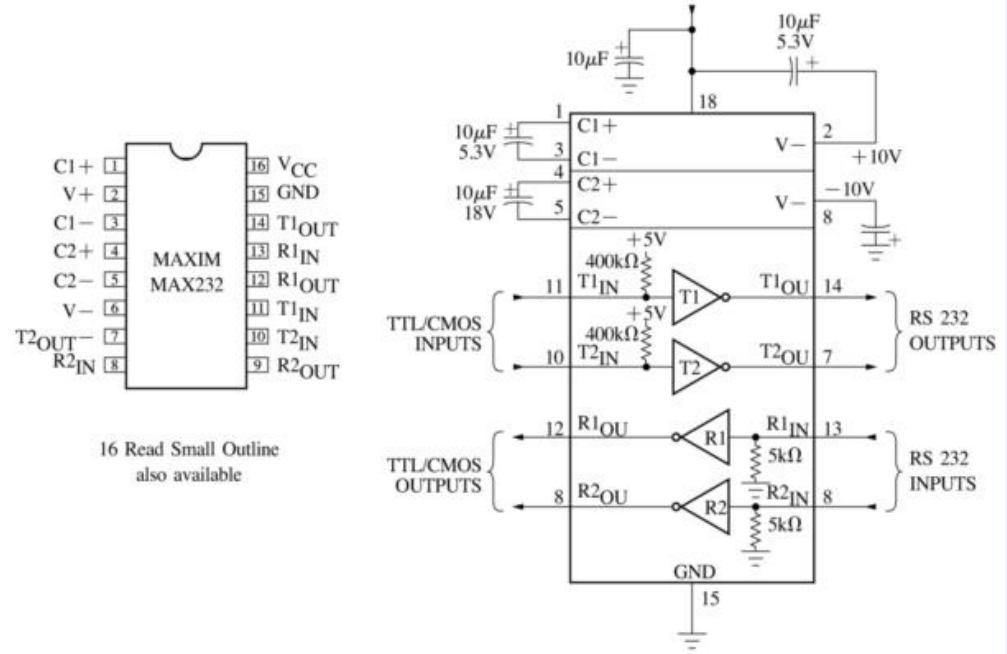


<기존의 인터페이스 IC를 이용한 컴퓨터 인터페이스 예>

직렬통신(RS232C)의 개요

인터페이스 IC(2)

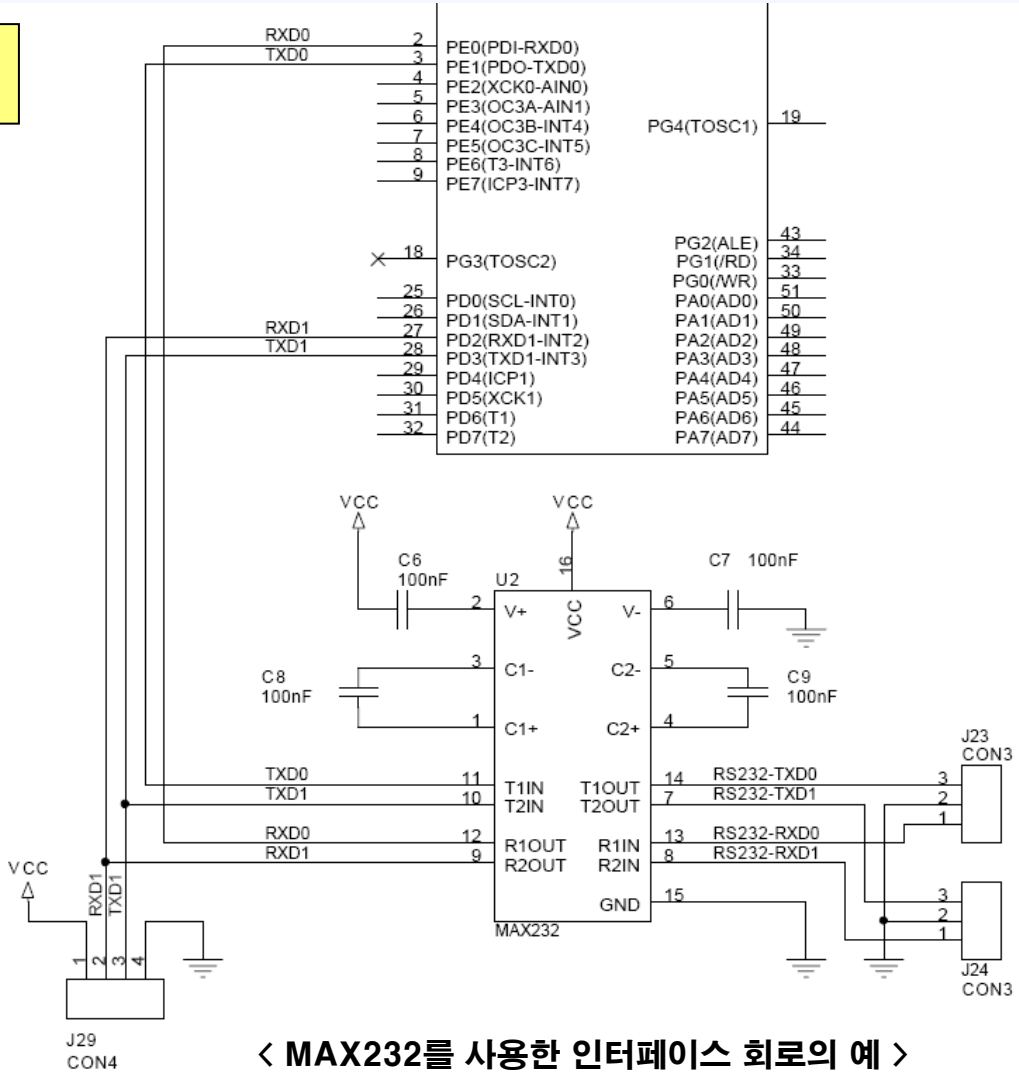
- 현재 EIA-232용의 신호 변환기로 MAX232가 주로 사용되고 있는데, 75188과 75189을 사용하는 대신에 MAX232을 사용하는 장점은 RS232C 신호를 송수신하는데 하나의 IC만을 사용한다는 것과 기존의 방식은 ±12V 전원을 별도로 필요로 하였는데 MAX232는 +5V 전원만으로 사용할 수 있음



<MAX232 외부 구조 및 내부 구조>

직렬통신(RS232C)의 개요

실험 회로

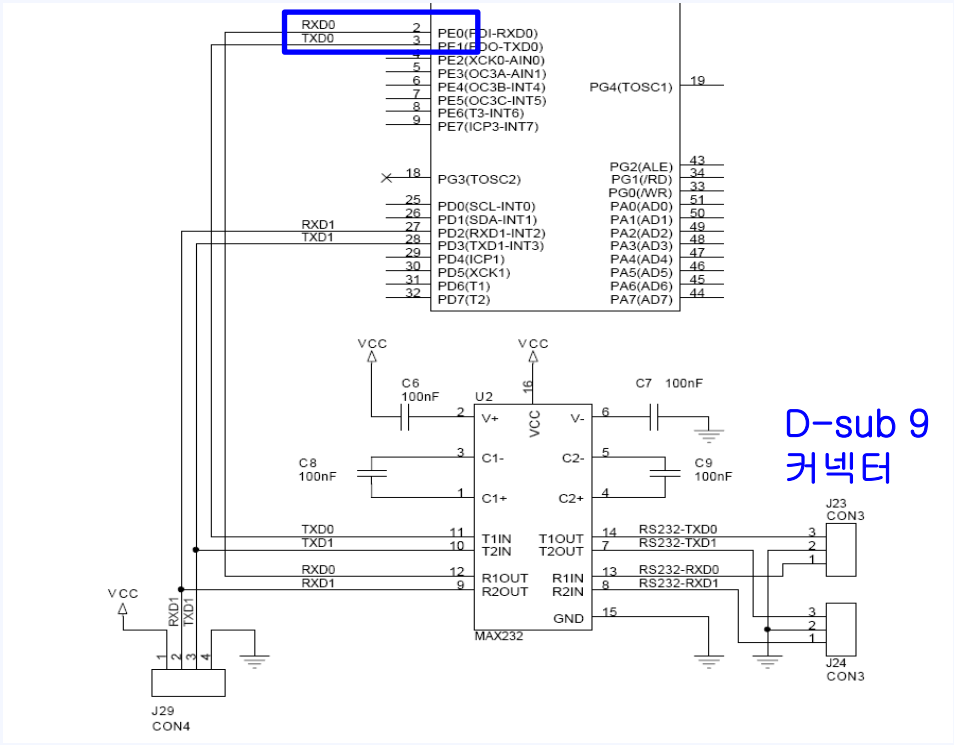


< MAX232를 사용한 인터페이스 회로의 예 >

교육용 보드의 USART 회로

USART0

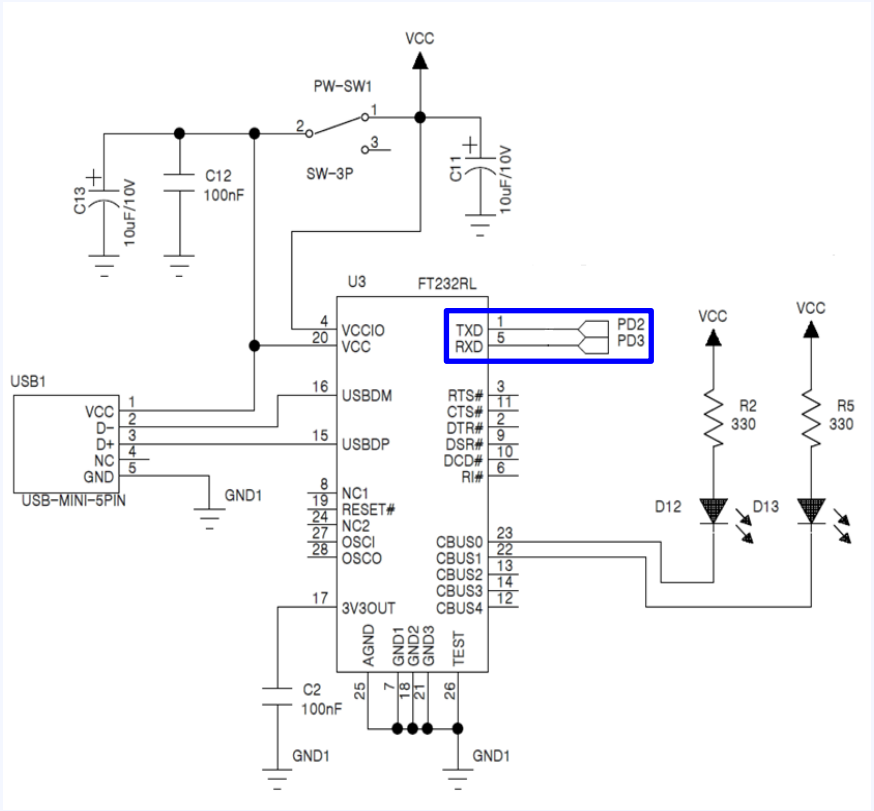
- 외부 RS232 인터페이스 회로



D-sub 9 커넥터

USART1

- USART to FT232 인터페이스 회로



직렬통신(RS232C)의 개요

기타 직렬 통신 방식의 비교

특 성	RS232C	RS422	RS485
동작 모드	single end	differential	differential
접속 가능 대수	1:1 통신	1:N(N=10) 통신	N:N(N=32) 통신 멀티드롭 통신 방식
최대 선로 길이	15[m]	1.2[㎞]	1.2[㎞]
최대 전송 속도	2×10^4[bps]	10^6[bps]	10^6[bps]
드라이버 출력 전압	±5V ~ ±15V	±2V ~ ±6V	±1.5V ~ ±6V
리시버 입력 전압	±3V ~ ±25V	±0.2V ~ ±6V	±0.2V ~ ±6V
데이터 “1” (마크)	-3V ~ -25V	V_A-V_B= -0.2V ~ -6V	V_A-V_B= -0.2V ~ -6V
데이터 “0” (스페이스)	+3V ~ +25V	V_A-V_B= +0.2V ~ +V	V_A-V_B= +0.2V ~ +V
최소 수신 전압	1.5[V](절대값)	100[㎷](차동값)	100[㎷](차동값)

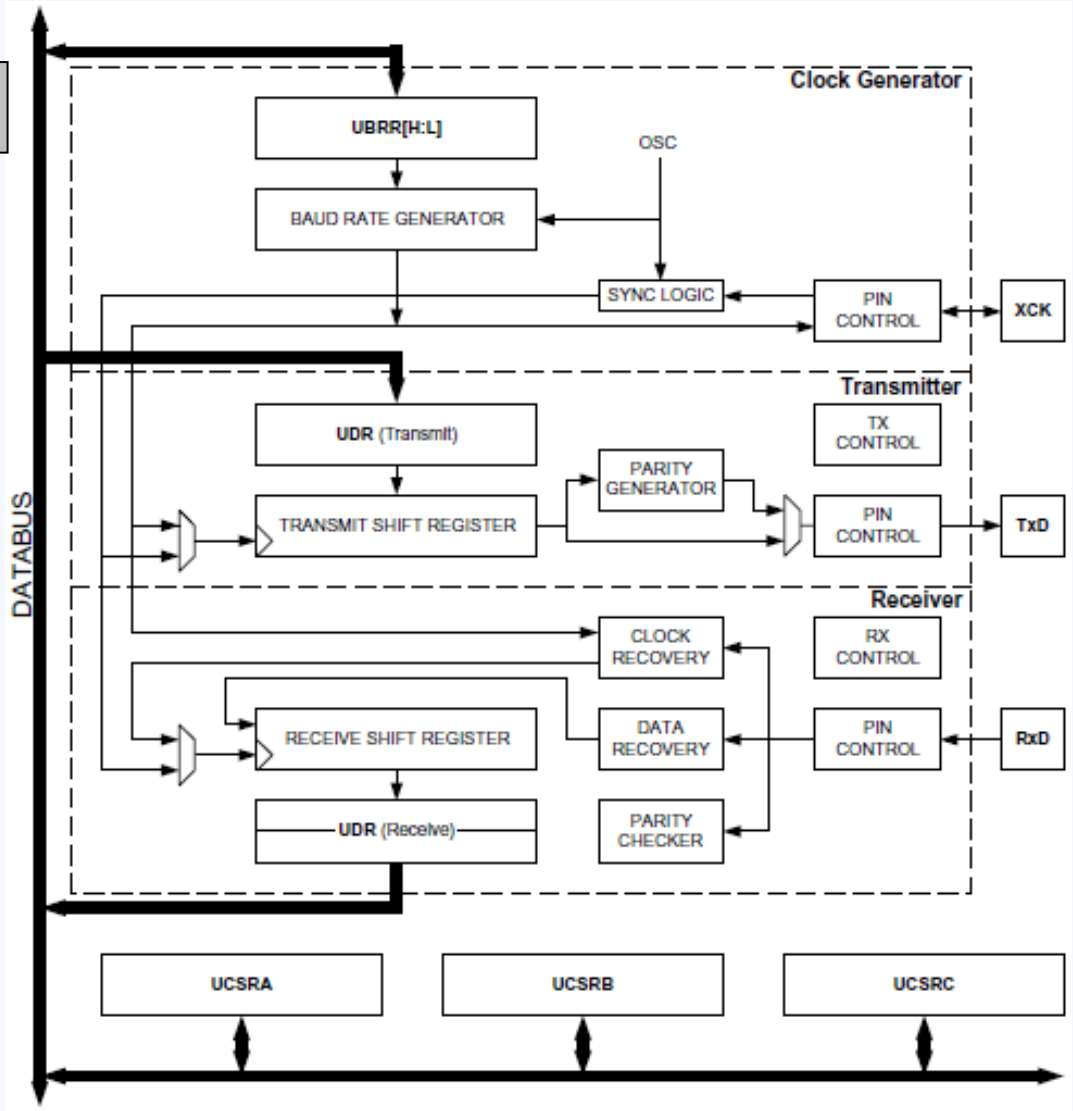
ATmega128 직렬 포트의 개요

USART의 특징

- 전이중 방식의 통신 모드 지원
- 비동기식 또는 동기식의 통신 모드를 지원
- 동기식으로 동작하는 마스터 또는 슬레이브 모드 지원
- 고 분해능의 보레이트 발진기 내장
- 다양한 직렬 통신 프레임
  - 5~9 비트의 data 비트와 1~2 비트의 stop 비트 제공.
- 오류 정정을 위한 짝수 또는 홀수 parity 발생/검사 기능을 하드웨어로 지원
- 데이터 오버런/프레임 오류 검출 기능을 내장
- 데이터의 신뢰성 향상을 위해 시작 비트 검출과 디지털 저대역 필터 등과 같은 잡음 제거 기능을 내장
- 송신 완료(TXCIEn), 송신 데이터 준비 완료(UDREn), 수신 완료(RXCIEn) 등의 3가지 인터럽트를 지원
- 다중 프로세서 통신 모드 지원
- 비동기 2배속 통신 모드 지원

ATmega128 직렬 포트의 개요

USART의 내부 구조

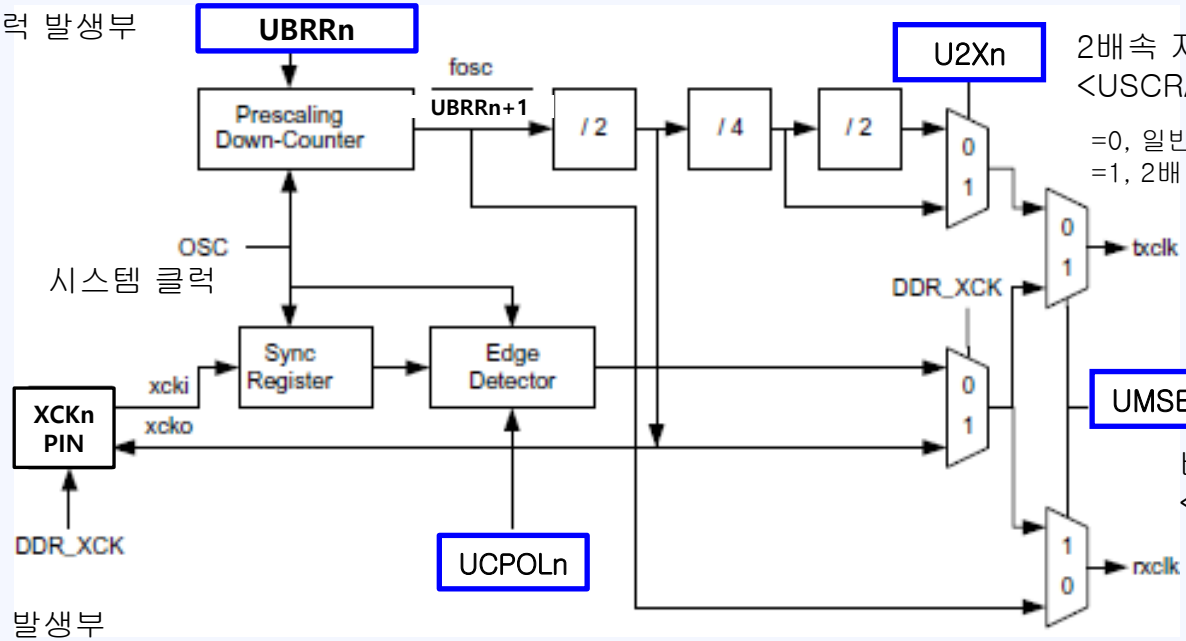


ATmega128 직렬 포트의 개요

클럭 발생부

- 송신/수신에 사용되는 클럭을 만드는 회로부
- 비동기 일반 모드, 비동기 2배속 모드, 동기 마스터 모드와 동기 슬레이브 모드에서 필요한 클럭을 발생

① 비동기 클럭 발생부



② 동기 클럭 발생부

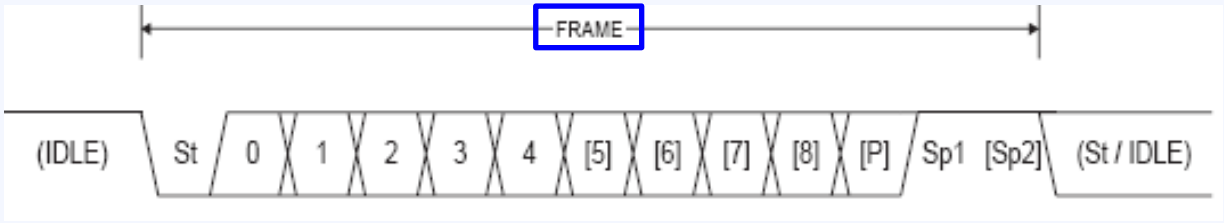
- ✓ 동기 방식의 경우, 마스터-슬레이브 모드로 동작
- ✓ 마스터 DDR_XCK -1, 슬레이브 = 0

ATmega128 직렬 포트의 개요

전송 데이터 형식

- 1 START 비트
- 5,6,7,8 또는 9 Data 비트
- 1 Parity 비트 (no, 짝수, 홀수)
- 1 또는 2 Stop 비트

☞ 직렬 데이터의 전송 형식



St : 시작 비트, L 상태
(n) : 데이터 비트(0~8)
P : 패리티 비트(짝수 또는 홀수 패리티)
S_P : 정지 비트, H 상태
IDLE : RxD 또는 TxD의 전송선에서 데이터의 비전송 상태, H 상태

직렬 포트 제어용 레지스터

USART 제어용 레지스터

USART 레지스터	설 명
UDRn	USART 입출력 데이터 레지스터
UCSRnA	USART 제어 및 상태 레지스터 A
UCSRnB	USART 제어 및 상태 레지스터 B
UCSRnC	USART 제어 및 상태 레지스터 C
UBRRnH/L	USART 보오 레이트 레지스터

USART 입출력 데이터 레지스터

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

- USART의 송수신 데이터 버퍼의 기능을 수행하는 레지스터

직렬 포트 제어용 레지스터

- USART 송신 버퍼 레지스터와 수신 버퍼 레지스터는 같은 이름의 레지스터 공유
- UDRn 레지스터는 동일한 I/O 번지를 갖지만, 내부적으로는 서로 분리되어 있음.
- 송신할 데이터를 UDRn 레지스터에 쓰면 TXBn 레지스터에 저장되고, 수신된 데이터를 UDRn로 읽으면 RXBn 레지스터의 내용이 읽혀진다.
- 송신 버퍼는 USCRnA 레지스터의 UDREn 플래그가 1로 설정되었을 경우에만 쓰기가 가능하며, UDREn 플래그가 0으로 되어 있는 경우에는 UDR 레지스터에 송신할 데이터를 쓰더라도 무시된다.
- 수신 버퍼는 2단계의 FIFO로 구성(이중 버퍼 구조)되어 있기 때문에, 이 수신 버퍼가 읽혀질 때마다 이 FIFO의 상태는 변화하게 된다.

USART 제어 상태 레지스터 A

- USART의 송수신 동작을 제어하거나 송수신 상태를 저장하는 기능을 수행하는 레지스터

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	

송수신 상태 통신 모드 설정용

직렬 포트 제어용 레지스터

- 비트 7 : RXCn (USART 수신완료, USART Receive Complete)
  - 데이터의 수신이 완료되어 버퍼에 문자가 들어와 있으면 1로 세트되고, 수신 버퍼에 수신된 문자 없으면 0으로 클리어된다.
  - 또한 RXCIEn 비트가 세트되어 있으면 수신완료 인터럽트(Receive Complete Interrupt)를 발생
- 비트 6 : TXCn (USART 송신완료, USART Transmit Complete)
  - 송신 시프트 레지스터에 있는 문자가 모두 송신되어 UDR 레지스터의 송신 버퍼가 비어 있는 상태가 되면 1로 세트된다.
  - TXC 플래그는 송신완료 인터럽트(Transmit Complete Interrupt)가 실행되거나 이 비트에 1을 쓰게 되면 0으로 클리어된다.
  - TXCIE 비트가 세트되어 있으면 송신완료 인터럽트를 발생를 발생
- 비트 5 : UDREn (USART 데이터 레지스터 준비완료, USART Data Register Empty)
  - 송신 버퍼(UDR)에 새로운 데이터를 쓸 준비가 되어있음을 알리는 상태 플래그로서 1로 세트되면 송신 버퍼가 비어 있어서 송신 데이터를 송신 버퍼에 쓸 준비가 되어 있음을 나타낸다.
  - UDRIEn 비트가 세트되어 있으면 데이터 레지스터 준비 완료 인터럽트를 발생
  - UDREn 비트는 MCU가 리셋된 후에는 송신기의 준비 상태를 나타내기 위해 1로 세트된다.

직렬 포트 제어용 레지스터

- **비트 4 : FEn (프레임 오류, Frame Error)**
  - UDR의 수신 버퍼에 현재 수신된 데이터를 수신하는 동안에 프레임 오류가 발생하였음을 나타내는 상태 플래그이다.
  - 프레임 오류는 수신 문자의 첫 번째 정지 비트가 0으로 검출되면 발생한다.
  - 이 플래그는 수신 버퍼 UDR을 읽을 때까지 유효하며, USCRnA 레지스터에 쓰기를 실행하면 항상 0으로 클리어된다.
- **비트 3 : DORn (데이터 오버런 오류, Data OverRun Error)**
  - 데이터를 수신할 경우에 오버런 오류가 발생하였음을 나타내는 상태 플래그이다.
  - 오버런 오류는 수신 버퍼에 현재 읽지 않은 수신 문자가 2개 들어와 있는 상태에서 수신 시프트 레지스터에 새로운 문자가 들어와 3번째 문자의 시작 비트가 검출되면 발생한다.
  - 이 플래그는 수신 버퍼 UDRn을 읽을 때까지 유효하며, USCRnA 레지스터에 쓰기를 실행하면 항상 0으로 클리어 된다.
- **비트 2 : UPEn (USART 패리티 오류, USART Parity Error)**
  - 수신 버퍼에 현재 수신된 데이터를 수신하는 동안에 패리티 오류가 발생하였음을 나타내는 상태 플래그이다.
  - 패리티 오류는 USCSRnC 레지스터의 UPM1 비트를 1로 설정하여 패리티를 검사하도록 설정한 경우에만 발생한다.
  - 이 플래그는 수신 버퍼 UDR을 읽을 때까지 유효하며, USCRnA 레지스터에 쓰기를 실행하면 항상 0으로 클리어된다.

직렬 포트 제어용 레지스터

- **비트 1 : U2Xn (2배속 전송 속도 설정, Double the USART Transmission Speed)**
  - U2X 비트는 비동기 모드에서만 유효하며, USART의 클럭 분주비를 16에서 8로 낮추어 전송 속도를 2배로 높이는 기능을 담당한다.
- **비트 0 : MPCMn (다중 프로세서 통신 모드, Multi-processor Communication Mode)**
  - MPCM 비트는 USART의 동작 모드를 다중 프로세서 모드로 설정하는 기능을 담당한다. MPCM 비트가 1로 설정되어 있으면, USART 수신부를 통해 들어온 수신 데이터가 주소 정보를 포함하지 않은 경우에는 모두 무시되고, 송신부는 이 비트의 설정값에 따라 영향을 받지 않는다. (다중 프로세서 통신 모드에 대한 자세한 설명은 교재 10.5절을 참조.)

☞ FEn, DORn, UPEn 상태 플래그는 수신 버퍼 UDRn을 읽을 때까지 유효하며, UCSRnA 레지스터에 쓰기를 실행하면 항상 0으로 클리어됨.

USART 제어 상태 레지스터 B

- **USART의 송수신 동작을 제어하거나, 전송 데이터를 9비트로 설정한 경우에 전송 데이터의 9번째 비트를 저장하는 기능을 수행하는 레지스터**

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

송수신 제어

9비트 전송용 비트

직렬 포트 제어용 레지스터

- 비트 7 : RXCIEn (USART 수신완료 인터럽트 허가, USART RX Complete Interrupt Enable)
  - 수신완료 인터럽트를 개별적으로 허가하는 비트이다.
  - RXCIE=SREG 레지스터의 I 비트=1; 하나의 문자가 수신되어 UCSRnA 레지스터의 RXC 비트가 1로 설정되면 수신완료 인터럽트가 발생한다.
- 비트 6 : TXCIEn (USART 송신완료 인터럽트 허가, USART TX Complete Interrupt Enable)
  - 송신완료 인터럽트를 개별적으로 허가하는 비트이다.
  - TXCIE=SREG 레지스터의 I 비트=1; 송신 시프트 레지스터가 비어 UCSRnA 레지스터의 TXC 비트가 1로 설정되면 송신완료 인터럽트가 발생한다.
- 비트 5 : UDRIEn (USART 데이터 레지스터 준비완료 인터럽트 허가, USART TX Complete Interrupt Enable)
  - 송신 데이터 레지스터 준비완료 인터럽트를 개별적으로 허가하는 비트이다.
  - UDRIE=SREG 레지스터의 I 비트=1; 송신 버퍼가 비어 UCSRnA 레지스터의 UDRE 비트가 1로 설정되면 데이터 레지스터 준비완료 인터럽트가 발생한다.

☞ 송신 완료와 송신 버퍼 empty 인터럽트의 차이점 : UDRIEn 인터럽트는 송신 버퍼의 데이터가 시프트 레지스터로 이동된 상태를 나타내므로, 이 상태가 송신완료보다 전송 시간 측면에서 빨리 대응이 가능함.

- 비트 4 : RXENn (수신 허가, Receiver Enable)
  - USART의 수신부의 사용을 허가하도록 설정하는 기능을 한다.
  - RXENn=1; RxD 핀이 병렬 I/O 포트가 아니라 직렬 데이터 수신 단자로서 동작하게 되며, 오류 플래그 FE, DOR, UPE의 동작이 유효하게 된다.

직렬 포트 제어용 레지스터

- **비트 3 : TXENn (송신 허가, Transmitter Enable)**
  - USART의 송신부의 사용을 허가하도록 설정하는 기능을 한다.
  - TXEN=1; TxD 핀이 병렬 I/O 포트가 아니라 직렬 데이터 송신 단자로서 동작하게 되며, 이 비트를 0으로 설정하더라도 송신 시프트 레지스터에 전송중인 데이터의 전송이 완료될 때까지는 유효하지 않게 된다.
- **비트 2 : UCSZn2 (문자 길이, Character Size)**
  - UCSZn2 비트는 UCSR 레지스터의 UCSZn1-0 비트와 함께 전송 문자의 데이터 비트수를 설정하는데 사용된다.
- **비트 1 : RXB8n (수신 데이터 비트 8, Receive Data Bit 8)**
  - RXB8 비트는 전송 문자가 9비트로 설정된 경우에 수신된 문자의 9번째 비트를 저장하는데 사용되며, 반드시 UDR 레지스터보다 먼저 읽혀져야 한다.
- **비트 0 : TXB8n (수신 데이터 비트 8, Transmit Data Bit 8)**
  - TXB8 비트는 전송 문자가 9비트로 설정된 경우에 송신할 문자의 9번째 비트를 저장하는데 사용되며, 반드시 UDR 레지스터보다 먼저 씌여져야 한다.

☞ RXB8n과 TXB8n 비트는 미리 액세스한 상태에서 UDRn레지스터를 액세스하여야 함.

직렬 포트 제어용 레지스터

USART 제어 상태 레지스터 C

- USART의 동작 모드 및 전송 제어 기능을 설정하는 레지스터

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	

패리티 **비트** Stop 비트 data length

- 비트 7 : 사용하지 않음
- 비트 6 : UMSELn (모드 선택, Mode Select)
  - 이 비트는 USART의 동작 모드를 비동기 모드와 동기 모드 중의 하나로 설정하는 기능을 담당한다.

UMSEL	모드
0	비동기 모드
1	동기 모드

직렬 포트 제어용 레지스터

- **비트 5-4 : UPMn1-0 (패리티 모드, Parity Mode)**
  - 이 비트들은 패리티 발생과 검사 형태를 설정하는 기능을 담당
  - 패리티 모드의 설정 방법은 표 11.7과 같다.
  - 패리티 사용이 설정되면, UCSRnA 레지스터의 UPEn 비트에 패리티 오류를 알려줌.

UPM1	UPM0	모드
0	0	패리티 기능을 사용하지 않음
0	1	사용하지 않음(reserved)
1	0	짝수 패리티 모드
1	1	홀수 패리티 모드

- **비트 3 : USBSn (정지 비트 선택, Stop Bit Select)**
  - 이 비트는 데이터 프레임에서 정지 비트의 수를 결정하는 기능을 담당
  - 비트의 설정값에 따른 정지 비트의 수

USBS	정지 비트 수
0	1-비트
1	2-비트

직렬 포트 제어용 레지스터

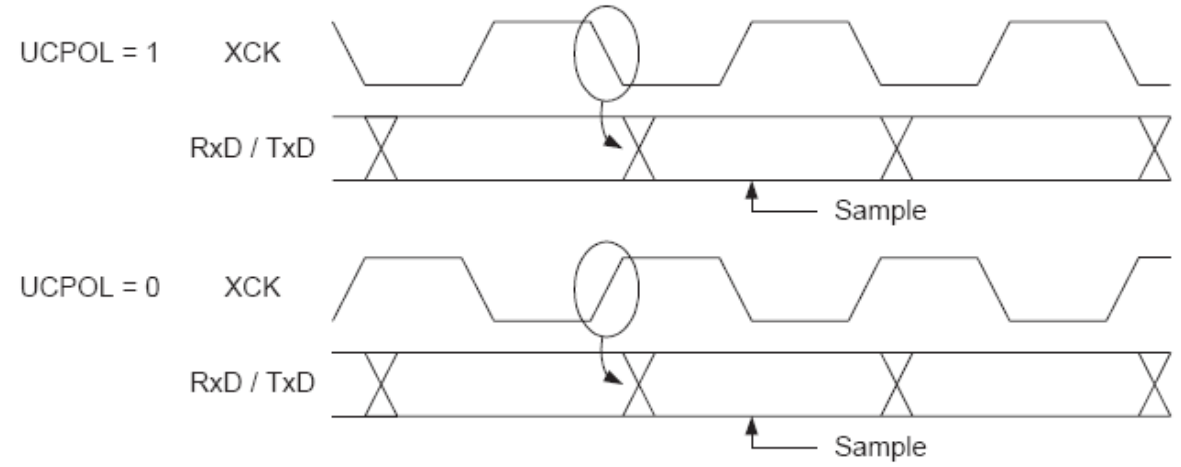
- **비트 2-1 : UCSZn1-0 (문자 길이, Character Size)**
  - 이 비트들은 UCSZ2 비트와 함께 전송 문자의 데이터 비트수를 설정하는데 사용
  - 이 비트들의 설정값에 따른 문자의 비트수

레지스터	UCSRnB	UCSRnC		
비트 이름	UCSZn2	UCSZn1	UCSZn0	전송 문자의 길이
설정 값	0	0	0	5 비트
	0	0	1	6 비트
	0	1	0	7 비트
	0	1	1	8 비트
	1	0	0	사용하지 않음(reserved)
	1	0	1	사용하지 않음(reserved)
	1	1	0	사용하지 않음(reserved)
	1	1	1	9 비트

직렬 포트 제어용 레지스터

- **비트 0 : UCPOLn (클럭 극성, Clock Polarity)**
  - **이 비트는 동기 전송 모드에서만 유효한 것으로 비동기 모드로 사용할 경우에는 0으로 설정한다. 동기 모드에서 이 비트는 동기 클럭에 대해 데이터 출력의 변화 시점과 데이터 입력의 샘플링 시점을 결정하는 기능을 담당**

UCPOL	전송 데이터의 출력 변화 시점 (TxD 핀의 출력)	수신 데이터의 샘플링 시점 (RxD 핀의 출력)
0	XCK의 상승 에지	XCK의 하강 에지
1	XCK의 하강 에지	XCK의 상승 에지



<동기 모드에서 XCK 클럭의 동작 타이밍도>

직렬 포트 제어용 레지스터

USART 보오 레이트 레지스터

- USART의 동작 모드 및 전송 제어 기능을 설정하는 레지스터

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	

- 비트 15-12 : 사용하지 않음
- 비트 11~0 : UBRRn11~0 (보오 레이트 선택 비트)
  - 클럭의 분주비로 작용하여 USART의 보오 레이트를 결정
  - 이 레지스터는 16비트이기 때문에 값을 쓸 경우에는 상위 바이트인 UBRRH를 먼저 쓰고, 하위 바이트인 UBRRL을 나중에 써야 함.

직렬 포트 제어용 레지스터

- USART의 보오 레이트를 설정하는 공식

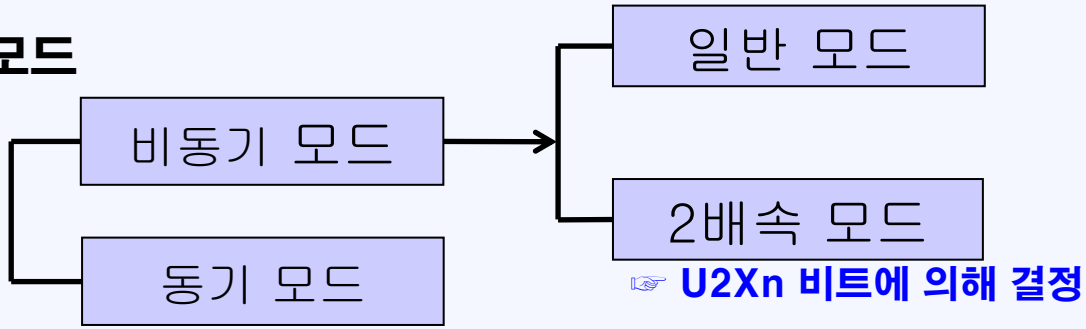
동작 모드	보오 레이트를 결정하는 공식	UBBR 레지스터의 값을 결정하는 공식
비동기 일반 모드 (U2X=0)	$BAUD = \frac{f_{osc}}{16(UBRR+1)}$	$UBRR = \frac{f_{osc}}{16 \times BAUD} - 1$
비동기 2배속 모드 (U2X=1)	$BAUD = \frac{f_{osc}}{8(UBRR+1)}$	$UBRR = \frac{f_{osc}}{8 \times BAUD} - 1$
동기 마스터 모드	$BAUD = \frac{f_{osc}}{2(UBRR+1)}$	$UBRR = \frac{f_{osc}}{2 \times BAUD} - 1$

- 보오 레이트에 따른 UBBRn 레지스터의 설정 예

전송 속도 (bps)	f_{osc} =16MHz				f_{osc} =8MHz				f_{osc}=7.3728MHz			
	비동기 일반 모드 (U2X=0)		비동기 2배속 모드 (U2X=1)		비동기 일반 모드 (U2X=0)		비동기 2배속 모드 (U2X=1)		비동기 일반 모드 (U2X=0)		비동기 2배속 모드 (U2X=1)	
	UBBR	오차(%)	UBBR	오차(%)	UBBR	오차(%)	UBBR	오차(%)	UBBR	오차(%)	UBBR	오차(%)
2400	416	-0.1	832	0.4	207	0.2	416	-0.1	191	0.0	383	0.0
4800	207	0.2	416	-0.1	102	0.2	207	0.2	95	0.0	191	0.0
9600	103	0.2	207	0.2	51	0.2	103	0.2	47	0.0	95	0.0
14,400	68	0.2	138	-0.1	34	-0.8	68	0.6	31	0.0	63	0.0
19,200	51	0.2	103	0.2	25	0.2	51	0.2	23	0.0	47	0.0
28,800	34	-0.8	68	0.6	16	2.1	34	-0.8	15	0.0	31	0.0
38,400	25	0.2	51	0.2	12	0.2	25	0.2	11	0.0	23	0.0
57,600	16	2.1	34	-0.8	8	-3.5	16	2.1	7	0.0	15	0.0
76,800	12	0.2	25	0.2	6	-7.0	12	0.2	5	0.0	11	0.0
115,200	8	-3.5	16	2.1	3	8.5	8	-3.5	3	0.0	7	0.0
230,400	3	8.5	8	-3.5	1	8.5	3	8.5	1	0.0	3	0.0
250,000	3	0.0	7	0.0	1	0.0	3	0.0	1	-7.8	3	-7.8
500,000	1	0.0	3	0.0	0	0.0	1	0.0	0	-7.8	1	-7.8
1M	0	0.0	1	0.0	-	-	0	0.0	-	-	0	-7.8
최대값	1Mbps		2Mbps		0.5Mbps		1Mbps		460.8kbps		921.6kbps	

USART의 동작

- USART의 동작 모드



데이터 송신부

- UCSRnB 레지스터의 TXENn=1인 경우에 데이터 전송 가능
  - ① 8비트 데이터의 전송 : UDRn 레지스터에 기록
  - ② 송신 플래그와 인터럽트(UDREn와 TXCn flag)
  - ➢ 데이터 레지스터 준비완료(UDREn) 플래그 사용
    - 상태 : UDREn=1(송신 버퍼 empty), UDREn=0(송신 버퍼 full)
    - 전송 데이터 쓰기는 UDREn 비트가 1(empty)인 상태에서 이루어져야 하고, UDRn 레지스터에 송신 데이터를 쓰면 클리어됨.
  - ➢ 송신완료(TXCn) 플래그 사용(송신 시프트 레지스터의 데이터 전송 완료 상태)
    - 상태 : TXCn=1(송신 버퍼 empty), TXCn=0(송신 버퍼 full)
    - TXCn 플래그는 송신완료 인터럽트가 실행되면 자동으로 0으로 클리어되거나 또는 이 플래그 비트에 1을 쓰면 클리어됨.

USART의 동작

③ 인터럽트의 발생에 의한 ISR(USARTn_DRE와 USARTn_TXC) 수행

- USARTn_DRE 인터럽트
  - SREG의 I =1이고, UDRIEn =1로 설정 : UDREn 비트가 1로 설정되어 있는 동안 데이터 레지스터 준비완료 인터럽트가 발생하여 USARTn_DRE 인터럽트 함수 수행
- USARTn_TXC 인터럽트
  - SREG의 I =1이고, TXCn =1로 설정 : UDREn 비트가 1로 설정되어 있는 동안 송신 완료 인터텁트가 발생하여 USARTn_TXC 인터럽트 함수 수행
  - 인터럽트 루틴이 수행되면 TXCn 플래그는 자동으로 클리어됨.

④ 패리티 발생기 : UPMn1 비트 =1인 상태에서는 USART 송신부에서는 문자를 송신할 때 패리티 비트를 자동으로 계산하여 패리티 부가함.

⑤ 송신부의 금지(TXENn 비트 = 0) 상태에서는 현재 전송중인 데이터를 제외하고 더 이상 송신부를 통해 데이터를 TxD 핀을 통해 송신할 수 없음.

USART의 동작

데이터 수신부

- UCSRnB 레지스터의 RXENn=1인 경우에 데이터 수신 가능

① 8비트 데이터의 수신 : UDRn 레지스터를 읽음

② 수신완료 플래그와 인터럽트(RXCn flag)

- 데이터의 수신 상태 : RXCn=1(수신 버퍼 full), RXCn=0(수신 버퍼 empty)

③ 인터럽트의 발생에 의한 ISR(USARTn_RXC) 수행

- SREG의 I =1이고, RXCIEn=1로 설정 : UDREn 비트가 1로 설정되어 있는 동안 데이터 수신 완료 인터럽트가 발생하여 USARTn_RXC 인터럽트 함수 수행
- UDRn 레지스터로부터 데이터를 읽으면 RXCn 플래그는 자동으로 클리어됨. 따라서, 데이터를 읽지 않으면 더 이상의 인터럽트는 발생하지 않게 되니 유의하여야 함.

④ 수신 오류 플래그 :

- 프레임 오류(FE), 데이터 오버런(DOR)과 패리티 오류(PE)의 3가지 오류를 검출
- 오류의 상태는 USCRnA 레지스터에 저장됨.

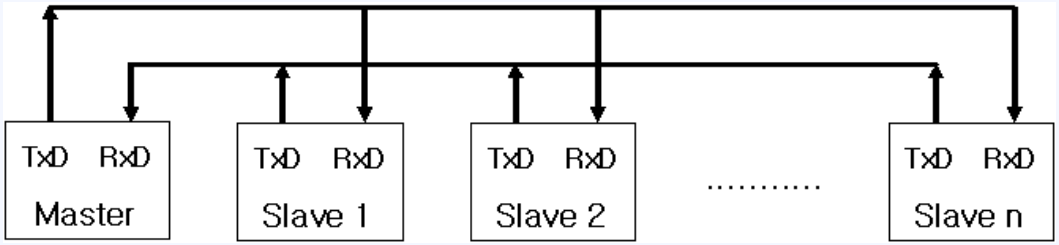
⑤ 패리티 감사기 : UPMn1 비트 =1인 상태에서 검출

- UPMn0=0(짝수 패리티), UPMn0=1(홀수 패리티)를 각각 검사하고, 수신 오류 비트에 상태 저장

⑤ 수신부의 금지(RXENn 비트 = 0) 상태에서는 송신부와는 달리 현재 수신중인 데이터를 포함하여 RxD 핀을 통해 들어오는 모든 수신 데이터는 손실됨.

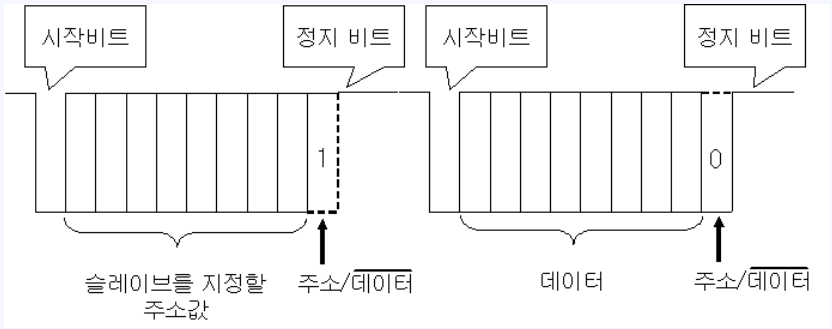
다중 프로세서 통신

다중 프로세서 통신 제어 시스템 구성



- 다중 프로세서 통신 모드는 1개의 마스터 프로세서가 여러 개의 슬레이브 프로세서에게 특정한 주소를 전송함으로써 1개의 슬레이브만을 지정하여 데이터를 전송하는 동작모드
- 이를 위해 송신측에서는 하나의 문자를 9비트로 구성하는 프레임으로 설정하여야 하며, 슬레이브로 동작하는 수신측 또한 9비트의 프레임을 사용하도록 설정하여함

다중 프로세서 통신 제어 프로토콜



다중 프로세서 통신

다중 프로세서 통신 제어 시스템 구성에서의 통신 절차의 예

	Master	Slave 1	Slave 2	Slave n
1	TXB8=1로 세트한다.	MPCM=1로 세트한다	MPCM=1로 세트한다	MPCM=1로 세트한다
2	주소 발생(TXB8=1)	인터럽트 발생	인터럽트 발생	인터럽트 발생
3		주소프레임 판정 = "NO"	주소프레임 판정 = "YES"	주소프레임 판정 = "NO"
4	TXB8=0으로 리셋한다.		MPCM=0으로 세트한다	
5	데이터 송신(TXB8=0)	무시	데이터 수신	무시
6	데이터 통신 완료		MPCM=1로 세트한다	
7		주소 발생에 따른 인터럽트 대기	주소 발생에 따른 인터럽트 대기	주소 발생에 따른 인터럽트 대기
8	필요에 따라 주소 발생 (TXB8=1)	인터럽트 발생	인터럽트 발생	인터럽트 발생

- 모든 슬레이브 프로세서는 MPCM=1로 하고, 마스터 프로세서가 주소 프레임을 송신하는 것을 기다림
- 마스터 프로세서는 TXB8=1로 해서 9번째 비트가 1인 주소 프레임을 송신
- 모든 슬레이브 프로세서는 주소 프레임을 수신하면 인터럽트를 발생시켜 수신된 데이터가 자신을 지정한 주소 프레임인지를 판별함
- 지정된 슬레이브 프로세서는 MPCM=0으로 하고, 마스터 프로세서에서 송신되는 데이터를 기다린다. 지정되지 않은 슬레이브 프로세서는 MPCM=1로 그대로 유지함
- 마스터 프로세서는 TXB8=0으로 해서 9번째 비트가 0인 데이터 프레임을 송신
- 지정된 슬레이브 프로세서는 MPCM=0으로 되어 있기 때문에 마스터 프로세서에서 송신된 데이터로 전부 수신할 수 있다. 그러나 지정되지 않은 슬레이브 프로세서는 MPCM=1로 되어 있기 때문에 주소 프레임 정보의 이하 데이터는 모두 무시하고, 다음의 주소 프레임을 기다림
- 지정된 슬레이브 프로세서에서는 통신이 전부 완료되는 시점에서 MPCM=1로 해서 다음의 주소 프레임을 기다림

USART의 초기화 및 액세스

- RS232C 통신 규격에서 정의하고 있는 통신 프로토콜을 설정(통신 속도, 패리티 사용여부, 통신 비트 수, 정지 비트 등)
- 송수신 가능하도록 초기화
- 송수신 플래그나 인터럽트 플래그 등을 사용하여 문자를 하나씩 송수신하는 방법을 구현

보오 레이트를 비롯한 통신 모드의 설정

- 비동기 모드에서의 통신 속도 : UCSR1A 레지스터의 U2X1 비트와 보오 레이트 레지스터 UBRR1H/L를 설정함으로써 결정
- 먼저 비동기 모드에서 115,200bps의 통신 속도를 갖도록 하기 위해서는 다음과 같은 과정을 수행

```
UCSR1A &= ~(1 << U2X1);                  // U2X 비트를 0으로 설정
UCSR1B = (1<<RXEN1) | (1<<TXEN1);        // RXEN0=1(수신 허가), TXEN0=1(송신 허가)
UCSR1C &= ~(1 << UMSEL1);                // 비동기 모드의 선택
UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);      // 8비트 문자열 설정

UBBR1H = 0;          // 115,400 보오 레이트 설정 (X-tal : 14.7456 Mhz)
UBBR1L = 7;          // UBBR0H/L 레지스터에 7을 기록
```

USART의 초기화 및 액세스

U2X 설정에 따른 UBRR값 설정 방법

비동기 2배속(U2X1=1)	비동기 1배속(U2X1=0)	
UCSR1C &= ~(1 << URSEL1);	UCSR1C &= ~(1 << URSEL1);	// 비동기 통신 모드 선택(공통)
UCSR1A |= (1<<U2X1);	UCSR1A &= ~(1<<U2X1);	// 비동기 2배속/일반 선택
UBRR1H = 0x00;	UBRR1H = 0x00;	// UBRR1H값 설정
UBRR1L = 0x0f;	UBRR1L = 0x07;	// UBRR1L값 설정

문자의 전송 형식의 설정 (USART1 기준)

- 문자의 길이, 패리티 사용 여부, 오류의 검출 여부
- 이는 UCSR1C 레지스터와 USCR1B 레지스터의 UCSZ12~UCSZ10 비트를
- 예) 문자의 전송 형식은 8비트의 문자, NO-패리티, 1비트의 정지 비트를 사용

```
UCSR1B &= ~(1<< UCSZ12);                    // UCSZ12=0 (8 비트 문자열 선택)
UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);
// UPM11~UPM10=00, USBS1=0, UCSZ11 ~ UCSZ10=11,
// UCSR1C = 0x0c;
```

비트	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
초깃값	0	0	0	0	0	0	0	0	

	0	0	0	0	1	1	0	0	
비트	7	6	5	4	3	2	1	0	
		UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초깃값	0	0	0	0	0	0	0	0	

USART의 초기화 및 액세스

송수신 허가

- **USART1을 이용하여 송수신을 하기 위해서는 UCSR1B 레지스터의 TXEN1 비트와 RXEN1 비트가 1로 설정되어 있어야 함.**

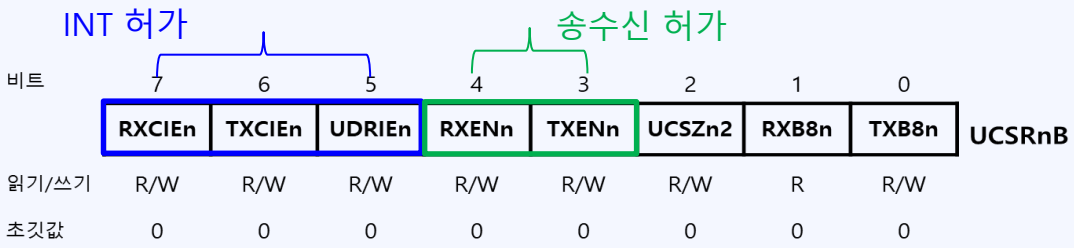
```
// 송수신 허가하여 송수신 준비 완료
UCSR1B = (1<<RXEN1) | (1<<TXEN1);  // RXEN1=1(수신 허가), TXEN1=1(송신 허가)
```

- **인터럽트를 사용하기 위한 준비**
  - **수신 인터럽트(RXCIE1) 비트를 1, 데이터 레지스터 준비 인터럽트(UDRIE1) 비트를 1로 설정하여 수신과 송신 준비를 함.**

```
UCSR1B | = (1<<RXCIE1) | (1<<UDRIE1);
```

  - **이상의 동작을 하나의 과정으로 수행**

```
UCSR1B |= 0xb8;                                  // 10111000로 설정
또는  UCSR1B |= (1<<RXCIE1) | (1<<UDRIE1) | (1<<RXEN1) | (1<<TXEN1);
```



USART의 초기화 및 액세스

데이터의 송수신

- 폴링 방법 : USART1의 경우, USCR1A 레지스터의 데이터 레지스터 준비 완료(UDRE1) 플래그와 수신 완료(RXC1) 플래그를 이용
- 인터럽트 방법 : USCR1B 레지스터의 송수신 인터럽트(RXCIE1, UDRIE1)를 허가하여 인터럽트가 발생하면 인터럽트 서비스 루틴에서 데이터를 송수신 하는 방법

① 폴링 방법으로 데이터를 송수신하는 방법

```
// 데이터 송신 방법
 unsigned char ch;
while( !(UCSR1A & (1<<UDRIE1)));
// USART1의 RDRE1 비트가 1이 될 때까지 기다림
UDR1 = ch;
// 문자를 전송, 문자를 전송한 후에는
// 자동으로 UDRE1 비트는 클리어 됨.

// 데이터 수신 방법
while(! (UCSR1A & (1 << RXC1)));
// USART1의 RXC1 비트가 1이 될 때까지 기다림
ch = UDR1;
// 문자를 수신, 문자를 수신한 후에는
// 자동으로 RXC1 비트는 클리어 됨.
```

② 인터럽트를 이용하여 데이터를 송수신하는 방법

```
UCSR1B |= (1<<RXCIE1) | (1<<UDRIE1) | (1<<RXEN1) | (1 <<TXEN1);
// RXCIE1=1(수신 인터럽트 설정), UDRIE1=1(송신 인터럽트 설정),
// RXEN1=1(수신 허가), // TXEN1 = 1(송신 허가)
sei();                    // 전체 인터럽트 허가

// 인터럽트 루틴에서의 데이터 수신 방법
ISR(USART1_RX_vect){
        ch = UDR1;
}

// 인터럽트 루틴에서의 데이터 송신 방법
ISR(USART1_UDRE_vect){
        UDR1 = ch;
}
```

USART의 초기화 및 액세스

9번째 데이터 비트의 처리

- 송신 : 9번째 데이터 비트를 사용하여 문자를 전송할 때에는 데이터를 UDR1로 쓰기 전에 USCR1B 레지스터의 TXB81 비트를 먼저 설정
- 수신 : 9번째 데이터 비트를 사용하여 문자를 수신할 때에는 데이터를 UDR1을 읽기 전에 USCR1B 레지스터의 RXB81 비트를 먼저 읽어야 함

송신

```
// USART1의 RDRE1 비트가 1이 될 때까지 기다림
while(!(UCSR1A & (1<<UDRE1)));

// copy 9th bit to TXB81
UCSR1B &= ~(1<<TXB81);
if (ch & 0x0100) UCSR1B |= (1<<TXB81);

// put data into buffer, send the data
UDR0 = ch;
```

수신

```
// USART1의 RXC 비트가 1이 될 때까지 기다림
while(!(UCSR1A & (1<<RXC1)));

// get status and 9th bit then read data
        status = UCSR1A;
        resh = ( UCSR1B & (1<<RXB81) ) >> 1;
        // 9번째 비트 읽음.USCR1B;
        resl = UDR1;

// if error, error process
if (status & (1<<FE1) | (1<<DOR1) | (1<<UPE1))
{
...
}

// Filter the 9th bit then write data(9 bit)
ch_rec9 = (resh<<8) | resh;
// ch_rec9 변수는 unsigned int로 미리 선언되어 있어야 함.
```

USART 활용 실험

예제 1: 직렬 포트의 초기화

- **115200 보오 레이트의 동작 속도를 갖고, 1-비트 짝수 패리티, 8-비트 문자, 1-정지 비트의 문자 전송 형식을 갖고, 비동기식으로 동작하도록 USART1을 초기화하는 프로그램을 작성**

UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1
설정값	0	0	0	0	0	0	0	0
UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ21	RXB81	TXB81
설정값	1	0	1	1	1	0	0	0
UCSR1C	-	UMSEL1	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1
설정값	0	0	1	0	0	1	1	0
UBRR1H	-	-	-	-	UBRR1[11:8]			
설정값	0	0	0	0	0	0	0	0
UBRR1L	UBRR1[7:0]							
설정값	0	0	0	0	0	0	1	1

```
void Init_USART1(void)
{

   UCSR1A = 0x00;                  // 1배속 전송모드
   UCSR1B =(1<<RXEN1)|(1<<TXEN1);  // 송수신 허가
   UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);
   // 비동기통신, 짝수 Parity, 1 Stop Bit, 8 데이터 비트 설정
   // UCSR1C &= ~(1<<UMSEL1); // 비동기 모드의 선택
   UBRR1H = 0x00;
   UBRR1L = 0x07;                  // 115200bps
}

void Init_USART1_IntCon(void)
{
   UCSR1A = 0x00;                   // 1배속 전송모드
   UCSR1B = (1<<RXCIE1)|(1<<UDRIE1)|(1<<RXEN1)|(1<<TXEN1);
   // 송수신 및 송수신 인터럽트 허가
   UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);
   // 비동기통신, 짝수 Parity, 1 Stop Bit, 8 데이터 비트 설정
   // UCSR1C &= ~(1<<UMSEL1); // 비동기 모드의 선택
   UBRR1H = 0x00;
   UBRR1L = 0x07; // 115200bps
   sei();
}
```

USART 활용 실험

예제 2: 출력 문자 서브루틴

- **USART1을 포트를 통해 8-비트의 ASCII 문자를 송수신하기 위한 putch_USART1()와 getch_USART1() 함수를 작성**

```
void putch_USART1(char data)
{
    while( !(UCSR1A & (1<<UDRE1)));        // 전송 인터럽트가 걸릴 때까지 while문 반복
    UDR1 = data;                           // 데이터 값을 UDR1에 넣는다. = 전송
}
```

- **한 문자의 전송 과정**

  putch_USART1('a'); // 문자 'a'를 출력하는 방법

- **문자열의 전송 : 송신하고자 하는 문자열이 메모리 내에 미리 지정되어 있어야 함.**

```
char array[] = "Serial Comm Test₩r₩n";
char i;
for(i = 0; array[i] != 0x00; i++)
        putch_USART1(array[i]);     // 문자열의 마지막은 "NULL"이므로 이를 검사
```

```
Byte getch_USART1(void)
{
    while(! (UCSR1A & (1 << RXC1)));
    return UDR1;
}
```

※Byte -> unsigned char

USART 활용 실험

예제 3: 문자열을 출력하는 프로그램

- **putch_USART1() 함수를 이용하여 USART1 통신 포트로 정해진 문자열을 송신하는 프로그램을 작성**

```
void puts_USART1(char *str)                 // 문자열 출력 루틴
{
      while( *str != 0)
      {                                     // 문자의 마지막에는 '₩0'이 들어가 있으므로
            putch_USART1(*str);             // '₩0'이 나올 때까지 출력한다.
            str++;
      }
}
```

➢ **"Serial Comm Test" 라는 문자열을 전송하는 프로그램**

```
int main(void)
{
     char array[] = "Serial Comm Test₩r₩n";  // ₩r₩n-> /r/n
     Init_Serial();
     while(1)
     {
            puts_USART1(array);
     }
}
```

USART 활용 실험

예제 4: 입력 문자 에코 백 시험

- **교육용 보드와 PC를 USART1 통신 포트를 사용하여 연결한 후에 다음과 같은 과정을 수행하는 프로그램을 작성.**
  - **PC에서 키보드로 입력된 영문자를 교육용 보드로 전송**
  - **교육용 보드에서는 수신된 문자를 다시 PC로 재전송**
  - **PC에서 수신된 문자는 CRT 화면에 표시**
  - **단, 전송 모드는 8-비트 문자, No-패리티, 1-정지 비트를 사용하고, 눌려진 문자와 에코 백 되는 문자를 처리하는 과정은 폴링 방식을 사용하여 작성**

```
#include <avr/io.h>
#define F_CPU 14.7456E6

void putch_USART1(char data)
{
        while( !(UCSR1A & (1<<UDRE1)));
        UDR1 = data;
}
Byte getch1(void)
{
         while(! (UCSR1A & (1<<RXC1)));
         return UDR1;
}
void Init_USART1(void)
{

        UCSR1A = 0x00; // 1배속 전송모드
        UCSR1B = (1 <<RXCIE1) | (1<<UDRIE1) | (1<<RXEN1) | (1 <<TXEN1);
        UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);
        UCSR1C &= ~(1<<UMSEL1); // 비동기 모드의 선택


        UBRR1H = 0;
        UBRR1L = 7; // 115200bps
}

int main(void)
{
      Init_USART1();
      while(1)
      {
            putch_USART1(getch_USART1());
            // 입력된 문자를 그대로 출력함.
      }
}
```

※Byte -> unsigned char

USART 활용 실험

예제 5: 문자열 전송 프로그램

- “Welcome to AVR Worlds” 라는 문자열과 “AVR ATmega128 USART Test.” 라는 문자열을 PC로 전송하여 PC의 화면에 표시하는 프로그램을 작성. 단, 문자열의 송신 방법은 USART1 송신 인터럽트(송신 버퍼 대기상태)를 사용

```
#include <avr/io.h>
#define F_CPU 14.7456E6
#include <avr/interrupt.h>

unsigned char count;
unsigned char *ptr;

char E_Welcom[] = "Welcome to AVR Worlds\r\n";
char H_Welcom[] = "AVR ATmega128 USART Test.\r\n";

// 인터럽트 루틴에서의 데이터 송신 방법
ISR(USART1_UDRE_vect)
{

        UDR1 = *ptr;
        ptr+;
                if(*ptr== 0) {
                    if(count == 0)
                            ptr= E_Welcom;
                    else
                            ptr= H_Welcom;
                    count = ~count;
                  }
}
```

USART 활용 실험

```
void Init_USART1(void)
{

    UCSR1A = 0x00; // 1배속 전송모드
    UCSR1B = (1<<RXCIE1) | (1<<UDRIE1) | (1<<RXEN1) | (1<<TXEN1);

    UCSR1C = (1<< UCSZ11) | (1<< UCSZ10);
    UCSR1C &= ~(1 << UMSEL0); // 비동기 모드의 선택

    UBRR1H = 0x00;
    UBRR1L = 0x07; //115200 bps

    sei(); //SREG |= 0x80;
}

int main(void)
{
     Init_USART1();
     ptr = E_Welcom;
     count = 0;

      while(1);
}
```

USART 활용 실험

예제 6:PC 키보드의 입력을 LCD에 표시

- **PC에서 키보드로 입력되는 영문자를 USART1 수신 인터럽트로 받아 교육용 보드의 LCD에 표시하는 프로그램을 작성. 키보드 입력은 선택 라인에서 링 버퍼 형태로 출력하며 “Enter"키 입력이 발생 될때마다 라인 변경을 수행함. 통신속도는 115200bps 로 함.**

```
#include <avr/io.h>
#define F_CPU 14.7456E6
#include <avr/interrupt.h>

#include "lcd.h"                              // LCD 사용을 위해 본인이 작성한 헤더 파일 참조
                                              // 또는 CLCD 드바이스 드라이버 코드 기재
#define MAXLEN  16        // LCD 16 문자
#define ENTER   0x0A      // ASCII LF, CR(0x0D)+LF(0x0A), '/r' + '/n'
#define enable  1
#define disable  0

unsigned char buffer_count=0, enter_flag=0;
unsigned char str[MAXLEN]={0,};          // 임시로 수신된 문자를 저장하기 위한 공간

// 인터럽트 루틴에서의 데이터 수신
ISR(USART1_RX_vect){

    str[buffer_count] = UDR1;

    if(str[buffer_count] == 0x0d)
        str[buffer_count]=0;
    else if(str[buffer_count] == ENTER)
        enter_flag = enable;
    else{
        if(buffer_count >= MAXLEN-1)
            buffer_count = 0;
        else
            buffer_count++;
    }
}
```

USART 활용 실험

```
void Init_USART1_IntCon(void)
{

    // RXCIE1=1(수신 인터럽트 허가), RXEN0=1(수신 허가), TXEN0 = 1(송신 허가)
    UCSR1B = (1<<RXCIE1)| (1<<RXEN1) | (1 <<TXEN1);
    UBRR1L = 0x07;              // 115200bps 보오 레이트 설정
    sei();                      // 전체 인터럽트 허가
}

int main(void)
{
    unsigned char USART_test[]="Press Enter...PC";
    unsigned char LCD_line_eraser[]= "                ";
    unsigned char loop_count=0, display_line=0;

    LCD_Init();                 // LCD 초기화
    Init_USART1_IntCon();

    LCD_Pos(display_line,0);
    LCD_Str(USART_test);
    display_line = (~display_line) & 0x01;

    buffer_count=0;
```

```
while(1){
        if(enter_flag)
        {
            display_line = (~display_line) & 0x01;
            LCD_Pos(display_line, 0);
            LCD_Str(LCD_line_eraser);

            buffer_count = enter_flag = 0;

            for(loop_count=0; loop_count < MAXLEN; loop_count++)
              str[loop_count] = 0;          // 문자열 초기화
        }
        else if(str[buffer_count-1] !=0)
        {
            LCD_Pos(display_line, 0);       // LCD에 위치 지정
            LCD_Str(str);                   // 입력된 문자열 출력
        }
        else;
    }
}
```

OS에서 Enter 키의 처리
Windows 계열... CR+LF 두 글자로 개행 함.
Linux, Mac은 LF로 개행을 대신 함.

따라서 보통적으로 텍스트 모드에서는 LF만 검사함.

Q & A