



AVR\_UART

임베디드스쿨2기

Lv1과정

2021. 06. 11

박태인

# UART 통신(정의, 직렬방식)

이번 시간에는 UART 통신에 대해서 알아 볼 것이다.

우선 UART 통신이란?

## - UART(범용 비동기화 송수신기: Universal asynchronous receiver/transmitter)

- ㄴ 병렬 데이터의 형태를 직렬 방식으로 전환하여 전송한다.
- ㄴ 일반적으로 RS232, RS422, RS485 와 같은 통신 표준과 함께 사용한다.
- ㄴ UART의 U는 범용을 가리키는데 이는 자료형태나 전송속도를 직접 구성 할 수 있다.
- ㄴ 통신데이터는 메모리 또는 레지스터에 들어 있어 이것을 차례대로 읽어 직렬화 하여 통신한다.(최대 8비트가 기본)
- ㄴ 비동기 방식이라도 시작과 끝은 알아내야 하기에 수신 쪽에서 동기신호를 찾아내어 데이터의 시작과 끝을 시간적으로 알아 처리할 수 있도록 약속되어 있다.

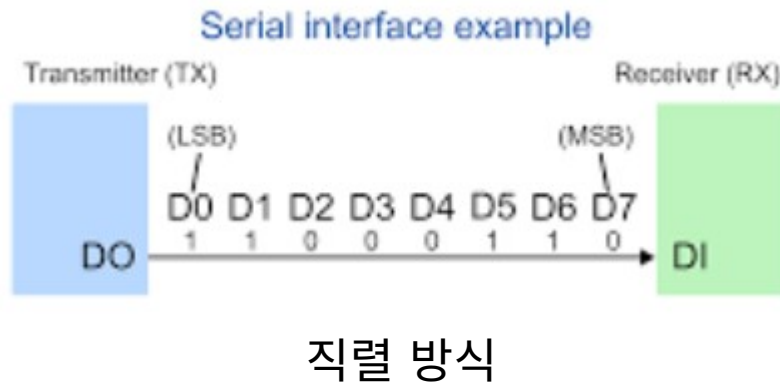
## - UART는 직렬방식으로 데이터를 전송한다고 하는데 그렇다면 직렬(Serial)방식으로 전환 한다는게 무슨 말 일까?

예를 들어 A를 아스키 코드로 변환한다고 생각해 보자.

A는 아스키 코드로 65 이므로 2진수 변환하면 0100 0001 이다.

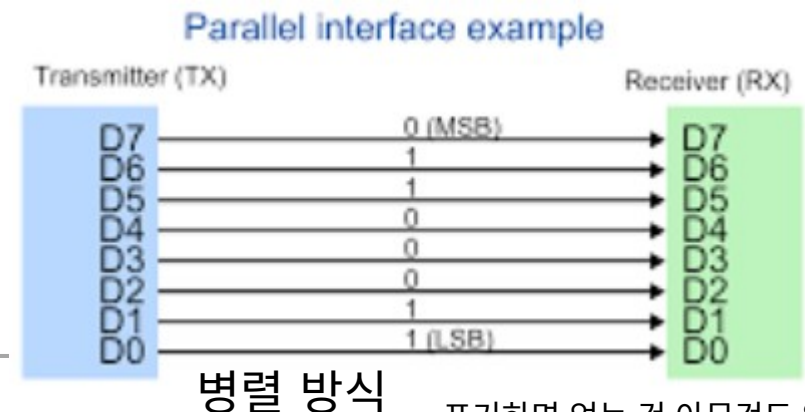
이것을 1을 보내고 0을 보내고 이런식 으로 데이터를 일렬로 바꿔서(직렬화) 한번에 한 비트 씩 전송하는 것이 직렬 통신이다.

아래 그림을 참고하면 좀 더 이해가 쉽다.



송 할 수 있지만  
한다는 단점이 있다.

반대로 병렬 방식은 아래 그림과 같은방식인데,

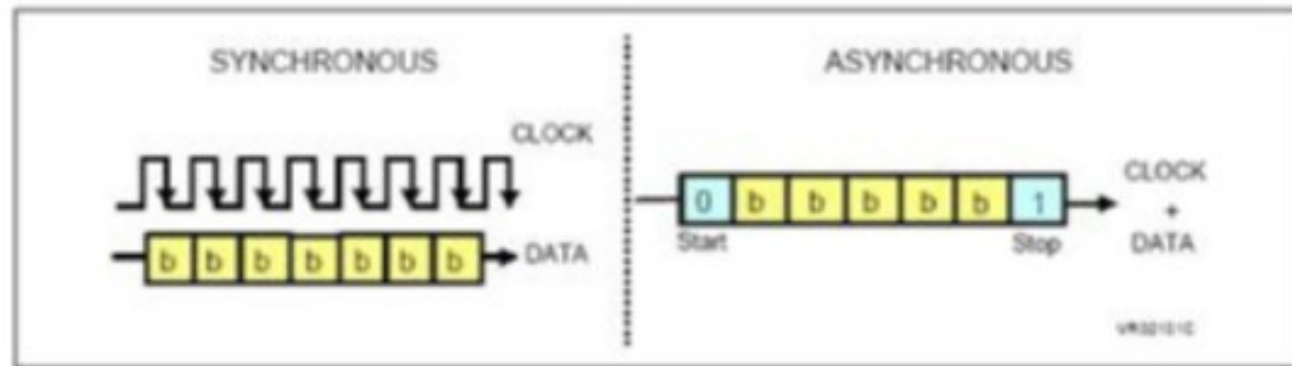


포기하면 얻는 건 아무것도 없다.

# UART 통신(비동기 통신 방식)

UART통신의 정의 중 **비동기 통신 방식**이라고 하는데, 이것은 또 어떤 말인지 알아 보자.

- 먼저 비동기란 무엇일까?
  - ↳ 비동기화란 **별도의 클럭을 사용하지 않고 데이터를 사용한다는 뜻이다.**  
따라서 동일한 **데이터 속도(Baud rate)**로 맞추어야 통신이 가능하다.  
**뿐만 아니라 Start bit와 Stop bit를 통해 시작과 끝을 알 수 있게 하여야 한다.**  
오류를 검사하기 위해 추가로 패리티 비트를 설정 할 수도 있다.
- 그럼 동기는 뭘까?
  - ↳ 동기는 반대로 **데이터를 동기화 하기 위해 별도의 클럭을 전송한다.**  
비동기 모드와 달리 **Start bit와 Stop bit를 사용하지 않기 때문에 전송 효율을 높여 빠르게 전송 가능**하며, **클럭을 사용하므로 오류가 적지만, 클럭 핀을 추가로 사용해야 한다는 단점이 있다.**  
클럭을 요구하는 동기모드의 시리얼 통신을 UART(Universal Synchronous Receiver/Transmit)라고 한다.



동기 통신과 비동기 통신

# UART 통신(UART/USART)

UART와 함께 나오는 말이 USART가 항상 따라 온다. 그것의 차이점을 간략하게 알아보고 넘어가자.

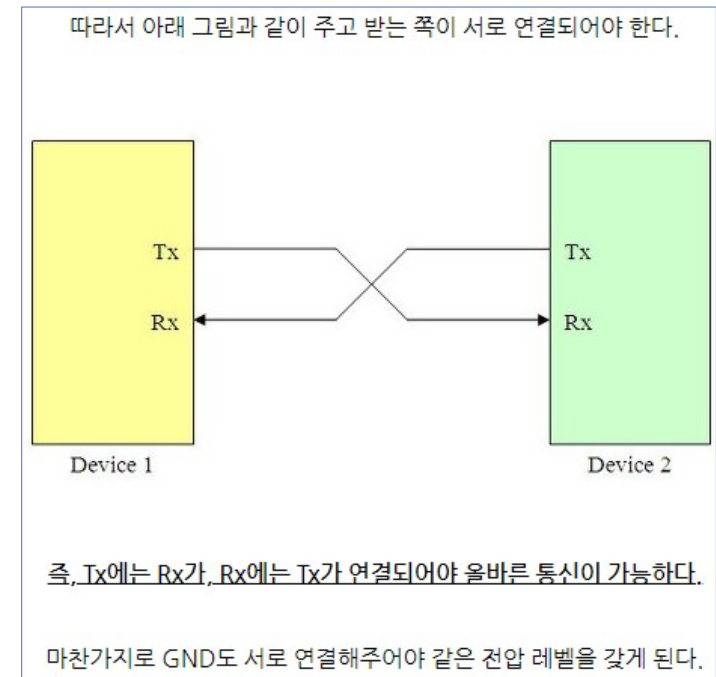
- 수많은 현대의 집적회로(IC)는 동기화 통신인 USART도 함께 지원한다. 이러한 장치들은 'USARTs' 또는 'USART/UART'로 부른다.
- 동기식 통신에는 여러가지 스타일이 있으며 'USART'라는 용어는 종종 '적어도 일부 동기식 통신 스타일도 지원하는 UART'를 의미하는데 사용된다.

**즉, 일반적으로 UART 시리얼 통신은 USART의 비동기 모드에 해당한다.**

그러나 절대적인 것은 아니기에 동기식 통신을 처리 할 수 있는지 여부를 결정 하려면 해당 USART 통신의 데이터 시트를 읽어야 한다.

USART/UART 통신은 기본적으로 Tx,Rx,GND의 3가지 라인을 가지고 있다.

Tx는 Transmit(송신)  
Rx는 Receive(수신)  
GND는 Ground(접지)



# UART 통신(데이터 송/수신형태)

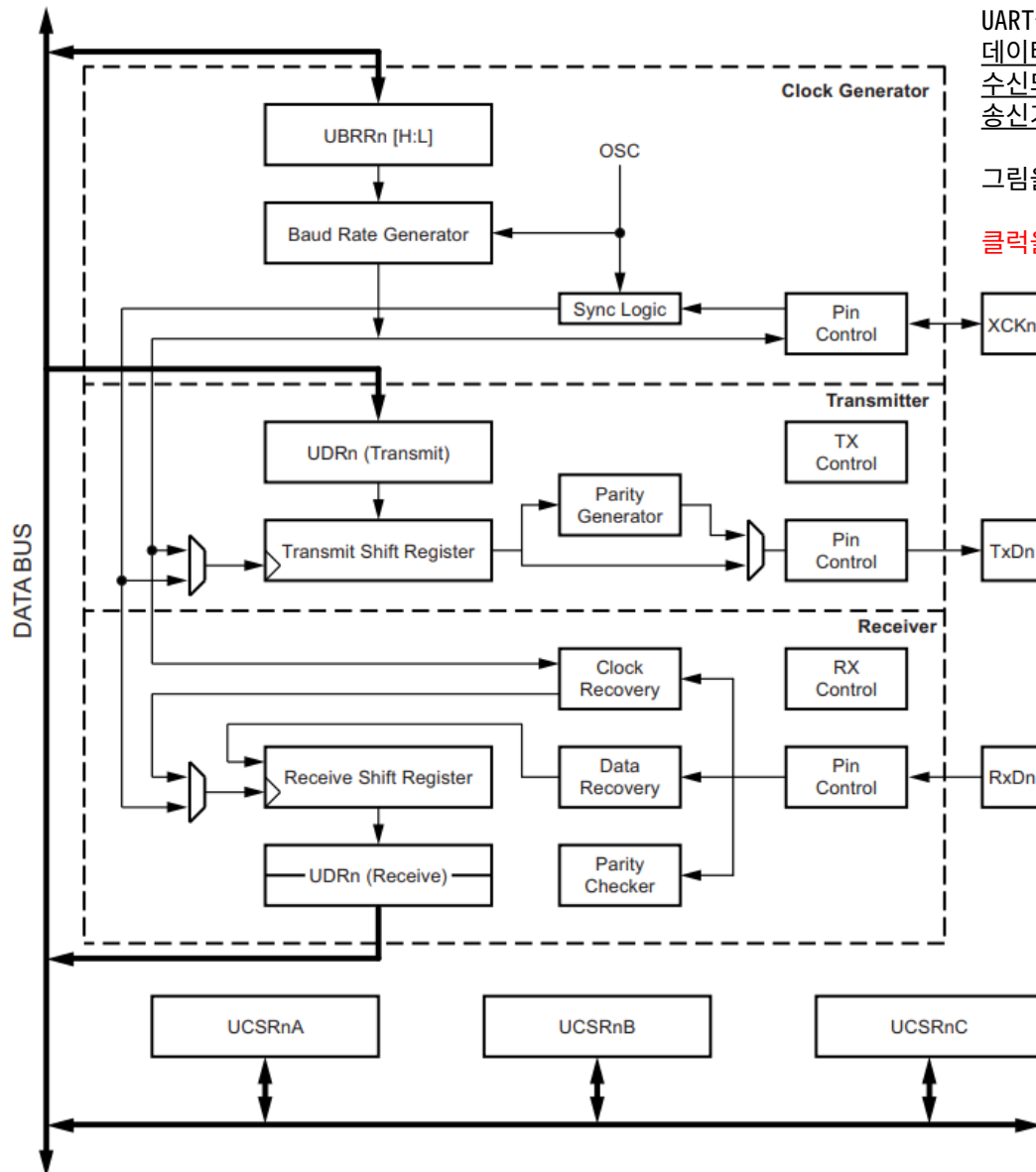
## 데이터 송 수신 형태 [편집]

비 트 수	1	2	3	4	5	6	7	8	9	10	11
	시작 비트 (Start bit)	5-8 데이터 비트								패리티 비트 (parity bit)	종료 비트 (Stop bit(s))
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Parity	Stop

- 가장 일반적으로 각 데이터 비트의 시간에 대해 16/64 배 빠른 클럭 신호를 이용하여 시작 비트로 부터 세어 각 비트의 경계를 찾아낸다.
- 이 클럭 신호는 자체적인 내부 클럭 디지털 회로에 의해 발생한다.
- 프로그래밍에 의한 레지스터 설정에 따라 클럭 신호의 주파수가 바뀐다. 통신 양쪽에서 설정을 미리 약속하고 클럭 신호 발생부의 레지스터를 같은 속도로 설정해야 통신이 원활하게 이루어 진다.
- 시작 비트 : 통신의 시작을 의미하며 한 비트 시간 길이 만큼 유지한다. 지금 부터 정해진 약속에 따라 통신을 시작한다.
- 데이터 비트 : 5 ~ 8 비트의 데이터 전송을 한다. 몇 비트를 사용할 것인지는 해당 레지스터 설정에 따라 결정된다.
- 패리티 비트 : 오류 검증을 하기 위한 패리티 값을 생성하여 송신하고 수신 쪽에 오류 판단한다.  
사용안함, 짝수, 홀수 패리티 등의 세가지 옵션으로 해당 레지스터 설정에 따라 선택 할 수 있다.  
↳ 사용안함을 선택하면 이 비트가 제거 된다.
- 끝 비트 : 통신 종료를 알린다. 세가지의 정해진 비트 만큼 유지해야 한다. 1, 1.5, 2비트로 해당 레지스터 설정에 따라 결정된다.

# UART 통신(블록도)

USART Block Diagram<sup>(1)</sup>



- 좌측 그림은 AVR에 있는 UART 블록의 블록도 이다.

UART블럭은 크게 세부분으로 나누어 볼 수 있다.

데이터를 정확한 시간에 맞춰 내보내거나, 수신되는 데이터를 정확한 위치에서 샘플링 할 수 있게 하기 위한 Clock generator 송신기능을 처리해 주는 transmitter가 있다. 그리고 수신의 receiver가 있다.

그림을 자세히 보면 UART 블록에는 Data bus에 연결된 다섯개의 레지스터를 찾을 수 있다.

클럭을 설정하기 위한 **UBRR** 레지스터.

↳ UART에서 통신속도를 baud rate 라고 하는데 이 값은 아무 값이나 설정해서 사용하는 것이 아니고 정해진 몇 개의 값들이 있다.(예전에는 9600bps를 많이 사용)

그러나 최근 임베디드 시스템에서는 115200bps를 주로 쓴다고 한다.

이 외에도 GPS 모듈과 같이 UART로 통신하는 부품들은 57600이나 38400 등등 정해진 다른 값을 사용하기도 한다.

Baud rate 속도가 미리 정해져 있는 경우는 거기에 맞춰서 동작되도록 할 수 밖에 없겠지만, 그렇지 않고 디버깅을 위하여 컴퓨터와 통신할 때에는 가급적 빠를수록 좋다.

UART 블록의 제어와 인터럽트 설정 등 기타 기능을 제어하기 위한 **UCSRA, UCSRB, UCSRC** 레지스터가 있다.

송신 데이터를 내보내거나 수신데이터가 저장되는 **UDR** 레지스터가 있다.

Transmitter블럭 에도 UDR 레지스터가 있고, receiver블럭에도 UDR 레지스터가 있다. 이름과 주소가 같지만 하드웨어 적으로 서로 따로 존재한다.

CPU에서 UDR 레지스터에 쓰기 동작을 하면 Transmitter에 있는 UDR 값을 쓰게 되고, 읽기 동작을 수행하면 Receiver에 있는 UDR 레지스터에서 데이터를 읽어 오게 된다.

# UART 통신(코드)

```

1  /*
2  */
3  #define F_CPU 16000000UL // 재정의 오류가 날수 있으므로 avr/io 헤더 보다 위에 놓여야 한다. UL을 붙여줘야 일정한 숫자 타입이 될 수 있다.
4  // 16000000 CPU_Clock(fosc)
5  #include <avr/io.h>
6  #include <util/delay.h> // 헤더 선언
7
8  // UART 통신 속도
9  #define USART_BAUDRATE 9600 // Baud rate 9600
10
11 // 테스트 후에 설명..
12 // UART의 최대 단점은 ?? 오차율이 있다. 근래는 반응 속도가 빨라서 꼭 그렇진 않다.
13 // 16000000 / (16 * 9600) - 1 = 103
14 // 11 10 9 8 7 6 5 4 3 2 1 0
15 // 11~8번 비트까지 얻고자 한다면 하위 8 비트를 밀어야함
16 // 남은 8 비트는 그대로 사용하면 됨.
17 #define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
18
19 // USART 모드 선택
20 // 동기 방식이란 ? 클럭의 업다운에 의해 시작과 끝을 결정하게 됨
21 // 또한 데이터의 처리 역시 동기 방식을 취할 수 있음
22 // 그러나 최근 트랜드에서는 속도 측면 때문에 시작과 끝을 동기 처리 하고
23 // 중간 데이터는 비동기로 처리하는 편이다. (하이브리드 방식)
24 // 비동기 방식이란 ? 클럭의 업다운에 영향을 받지 않고 처리가 가능한 선에서 최대한 빨리 여러 데이터를 처리하는 방식이다.
25
26 #define ASYNCHRONOUS (0 << UMSEL00) // 데이터에 한에서만 비동기, 사실 시작과 끝은 알아야 하므로
27
28 // USART PARITY 비트를 선택
29 #define DISABLED (0 << UPM00)
30 #define EVEN_PARITY (2 << UPM00)
31 #define ODD_PARITY (3 << UPM00)
32 #define PARITY_MODE DISABLED // 데이터를 다시 보관하기 위한게 패리티 모드인데 일단은 disable
33

```

Table 19-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate <sup>(1)</sup>	Equation for Calculating UBRRn Value
Asynchronous normal mode (U2Xn = 0)	$BAUD = \frac{f_{osc}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{16BAUD} - 1$

이부분은  
뒷 부분에서  
레지스터 분석.



# UART 통신(코드)

```
26 #define ASYNCHRONOUS (0 << UMSEL00) // 데이터에 한에서만 비동기, 사실 시작과 끝은 알아야 하므로
27
28 // USART PARITY 비트를 선택
29 #define DISABLED (0 << UPM00)
30 #define EVEN_PARITY (2 << UPM00)
31 #define ODD_PARITY (3 << UPM00)
32 #define PARITY_MODE DISABLED // 데이터를 다시 보관하기 위해 패리티 모드인데 일단은 disable
```

// USART STOP 비트 선택

```
#define ONE_BIT (0 << USBS0)
#define TWO_BIT (1 << USBS0)
#define STOP_BIT ONE_BIT
```

// USART DATA 비트 선택, 실제 데이터 넘어 가는 부분

```
#define FIVE_BIT (0 << UCSZ00)
#define SIX_BIT (1 << UCSZ00)
#define SEVEN_BIT (2 << UCSZ00)
#define EIGHT_BIT (3 << UCSZ00)
#define DATA_BIT EIGHT_BIT
```

본격적으로 데이터 형태를 설정 하는 부분이다.

) << UMSEL00 : UART0은 비동기 방식, 1은 동기 방식

- UPM은 패리티 비트 설정(아래의 테이블 참고.)

Table 19-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, even parity
1	1	Enabled, odd parity

- USBS 비트는 STOP 비트의 개수를 선택.(아래 테이블 참조)

Table 19-6. USBS Bit Settings

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



# UART 통신(코드)

```
// USART DATA 비트 선택, 실제 데이터 넘어 가는 부분
#define FIVE_BIT      (0 << UCSZ00)
#define SIX_BIT       (1 << UCSZ00)
#define SEVEN_BIT     (2 << UCSZ00)
#define EIGHT_BIT     (3 << UCSZ00)
#define DATA_BIT     EIGHT_BIT
```

UCSZ는 데이터 비트의 길이를 설정하는 곳이다. (아래 표 참조)

Table 19-7. UCSZn Bits Settings

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

```
47 void USART_Init()
48 {
49     // 통신 속도 설정
50     UBRROH = BAUD_PRESCALE >> 8;
51     UBRROL = BAUD_PRESCALE;
52
53     // 프레임 포맷 설정
54     // 비동기 방식으로 패리티는 사용하지 않고 정지를 알리는 비트 1개와 8개의 데이터비트로 구성된다.
55     UCSROC = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;
56
57     // 인터럽트 빈도수가 높은 경우 Bottom Half 정책을 사용
58     // 송수신 허용
59     UCSROB = (1 << RXEN0) | (1 << TXEN0);
60 }
```

자, 이제 본격적으로 define한 값들을 활용하여 UART를 초기화 한다.

# UART 통신(코드)

```
47 void USART_Init()  
48 {  
49     // 통신 속도 설정  
50     UBRR0H = BAUD_PRESCALE >> 8;  
51     UBRR0L = BAUD_PRESCALE;  
52 }
```

BAUD\_PRESCALE은 앞서 defie에서 계산한 Baud rate 값이다.

이것을 Baud Rate 레지스터인 **UBRRn** 에 입력 한다.

12bit의 값을 H에 넣고 L 에 넣는데, H의 경우 8 비트 우측 쉬프트 하여 값을 추출해 낸다.

## 19.10.5 UBRRnL and UBRRnH – USART Baud Rate Registers

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	

- **Bit 15:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRnH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRnH contains the four most significant bits, and the UBRRnL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing UBRRnL will trigger an immediate update of the baud rate prescaler.

# UART 통신(코드)

```

52
53 // 프레임 포맷 설정
54 // 비동기 방식으로 패리티는 사용하지 않고 정지를 알리는 비트 1개와 8개의 데이터비트로 구성된다.
55 UCSROC = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;
56

```

## 4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	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	

### • Bits 7:6 – UMSELn1:0 USART Mode Select

These bits select the mode of operation of the USARTn as shown in Table 19-4.

Table 19-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM) <sup>(1)</sup>

Note: 1. See Section 20. "USART in SPI Mode" on page 166 for full description of the master SPI mode (MSPIM) operation

### • Bits 5:4 – UPMn1:0 Parity Mode

These bits enable and set type of parity generation and check. If enabled, the transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn flag in UCSRnA will be set.

Table 19-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, even parity
1	1	Enabled, odd parity

### • Bit 3 – USBSn: Stop Bit Select

This bit selects the number of stop bits to be inserted by the transmitter. The receiver ignores this setting.

Table 19-6. USBS Bit Settings

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

UCSRnC 레지스터에서 앞서 define 했던 레지스터 정의들을 활용하여 아래의 정보를 입력 하여 줍니다.

- 동기/비동기
- 패리티 모드 설정(disable)
- 스탑비트 개수 설정
- 데이터 비트 개수 설정 들의 설정을

설정 하여 줍니다.

### • Bit 2:1 – UCSZn1:0: Character Size

The UCSZn1:0 bits combined with the UCSZn2 bit in UCSRnB sets the number of data bits (character size) in a frame the receiver and transmitter use.

Table 19-7. UCSZn Bits Settings

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

### • Bit 0 – UCPOLn: Clock Polarity

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOLn bit sets the relationship between data output change and data input sample, and the synchronous clock (XCKn).

# UART 통신(코드)

```
57 // 인터럽트 빈도수가 높은 경우 Bottom Half 정책을 사용
58 // 송수신 허용
59 UCSROB = (1 << RXEN0) | (1 << TXEN0);
60 }
```

## UCSRnB – USART MSPIM Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIE <sub>n</sub>	TXCIE <sub>n</sub>	UDRIE	RXEN <sub>n</sub>	TXEN <sub>n</sub>	-	-	-	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	1	1	0	

### • Bit 7 - RXCIE<sub>n</sub>: RX Complete Interrupt Enable

Writing this bit to one enables interrupt on the RXC<sub>n</sub> flag. A USART receive complete interrupt will be generated only if the RXCIE<sub>n</sub> bit is written to one, the global interrupt flag in SREG is written to one and the RXC<sub>n</sub> bit in UCSRnA is set.

### • Bit 6 - TXCIE<sub>n</sub>: TX Complete Interrupt Enable

Writing this bit to one enables interrupt on the TxC<sub>n</sub> flag. A USART transmit complete interrupt will be generated only if the TXCIE<sub>n</sub> bit is written to one, the global interrupt flag in SREG is written to one and the TxC<sub>n</sub> bit in UCSRnA is set.

### • Bit 5 - UDRIE: USART Data Register Empty Interrupt Enable

Writing this bit to one enables interrupt on the UDRE<sub>n</sub> flag. A data register empty interrupt will be generated only if the UDRIE bit is written to one, the global interrupt flag in SREG is written to one and the UDRE<sub>n</sub> bit in UCSRnA is set.

### • Bit 4 - RXEN<sub>n</sub>: Receiver Enable

Writing this bit to one enables the USART receiver in MSPIM mode. The receiver will override normal port operation for the RxD<sub>n</sub> pin when enabled. Disabling the receiver will flush the receive buffer. Only enabling the receiver in MSPIM mode (i.e. setting RXEN<sub>n</sub>=1 and TXEN<sub>n</sub>=0) has no meaning since it is the transmitter that controls the transfer clock and since only master mode is supported.

### • Bit 3 - TXEN<sub>n</sub>: Transmitter Enable

Writing this bit to one enables the USART transmitter. The transmitter will override normal port operation for the TxD<sub>n</sub> pin when enabled. The disabling of the transmitter (writing TXEN<sub>n</sub> to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit shift register and transmit buffer register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD<sub>n</sub> port.

### • Bit 2:0 - Reserved Bits in MSPIM mode

When in MSPIM mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnB is written.

UCSRnB 레지스터에서 UART 통신의 송/수신을 허용합니다.

RXEN : Receiver Enable [ 수신 ON ]

TXEN : Transmitter Enable [ 송신 ON ]

# UART 통신(코드)

```
void USART_TransmitPollig(uint8_t DataByte)
{
    //UDR 데이터 준비 이전까지 구동하지 않음
    while ((UCSR0A & (1<< UDRE0)) == 0) {}

    UDR0 = DataByte;
}
```

좌측 함수는 송신시에 데이터 처리 방법을 나타내는 함수이다.

방식에는 크게 polling(폴링) 방식과 interrupt(인터럽트)방식이 있다.

좌측은 Polling 방식을 나타내며 Polling 방식은

리소스 사용량이 많다는 단점이 존재하지만

그와 동시에 응답성이 빠르고, 데이터 처리량을 많다는 장점이 존재 한다.

UCSRnA – USART MSPIM Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	-	-	-	-	-	UCSRnA
Read/Write	R	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	1	1	0	

## • Bit 7 - RXCn: USART Receive Complete

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn flag can be used to generate a receive complete interrupt (see description of the RXCIEn bit).

## • Bit 6 - TXCn: USART Transmit Complete

This flag bit is set when the entire frame in the transmit shift register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn flag can generate a transmit complete interrupt (see description of the TXCIEn bit).

## • Bit 5 - UDREn: USART Data Register Empty

The UDREn flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn flag can generate a data register empty interrupt (see description of the UDRIE bit). UDREn is set after a reset to indicate that the transmitter is ready.

## • Bit 4:0 - Reserved Bits in MSPI mode

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnA is written.

코드는 while 문을 활용하여

송신시 DataByte의 데이터를 송신할 준비가

되지 않았을 때 (UDRE0 이 0 일때)

는 while 문을 계속해서 돌게 되어 데이터를 전송하지 않게 되고,

송신할 준비가 완료 되었을 때 (UDRE0 이 1 일 때)

While 문을 빠져 나가게 되고

함수의 인자로 받게 된 DataByte가 UDR에 입력되고

데이터가 전송 된다.



# UART 통신(코드)

```
int main(void)
```

```
{
```

```
    USART_Init();  
    // Insert code
```

```
    while(1)
```

```
    {
```

```
        USART_TransmitPollig('E');  
        USART_TransmitPollig('M');  
        USART_TransmitPollig('B');  
        USART_TransmitPollig('E');  
        USART_TransmitPollig('D');  
        USART_TransmitPollig('D');  
        USART_TransmitPollig('E');  
        USART_TransmitPollig('D');  
        USART_TransmitPollig(' ');  
        USART_TransmitPollig('M');  
        USART_TransmitPollig('A');  
        USART_TransmitPollig('S');  
        USART_TransmitPollig('T');  
        USART_TransmitPollig('E');  
        USART_TransmitPollig('R');  
        USART_TransmitPollig(' ');  
        USART_TransmitPollig('C');
```

```
81     USART_TransmitPollig('D');  
82     USART_TransmitPollig('D');  
83     USART_TransmitPollig('E');  
84     USART_TransmitPollig('D');  
85     USART_TransmitPollig(' ');  
86     USART_TransmitPollig('M');  
87     USART_TransmitPollig('A');  
88     USART_TransmitPollig('S');  
89     USART_TransmitPollig('T');  
90     USART_TransmitPollig('E');  
91     USART_TransmitPollig('R');  
92     USART_TransmitPollig(' ');  
93     USART_TransmitPollig('C');  
94     USART_TransmitPollig('O');  
95     USART_TransmitPollig('U');  
96     USART_TransmitPollig('R');  
97     USART_TransmitPollig('S');  
98     USART_TransmitPollig('E');  
99     USART_TransmitPollig('!');  
100    USART_TransmitPollig('\r');  
101    USART_TransmitPollig('\n');  
102    USART_TransmitPollig('\0');  
103    _delay_ms(1000);  
104    /* delay.h 가 문제가 될 경우  
105    int i = 0;  
106    for(; i < 999999999; i++);  
107    */  
108    };  
109  
110    return 0;  
111 }  
112
```

이제 본격적으로

Main 문에서 UART 통신을 초기화 한 후

전송 함수를 통해 데이터를 전송하는 코드 이다.

모든 데이터 전송 후 delay 함수를

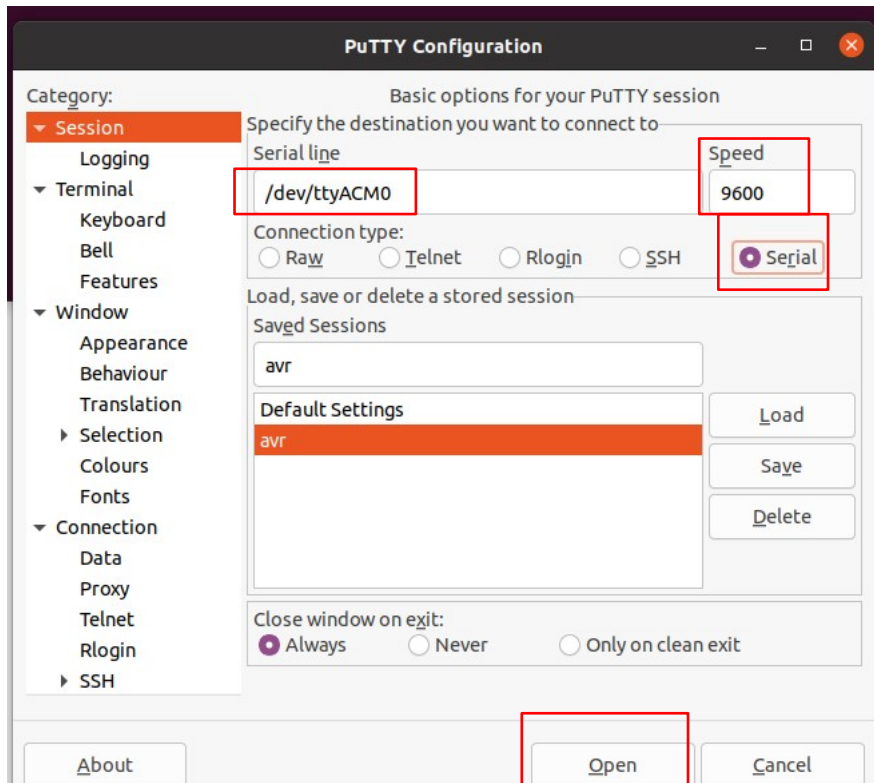
사용하여 1 초 단위로 데이터가 전송이 된다.

# UART 통신(작동확인)

```
taein@taein-Lenovo-ideapad-700-15ISK:~$ putty &
[1] 2519
taein@taein-Lenovo-ideapad-700-15ISK:~$
(putty:2519): Gtk-CRITICAL **: 07:15:28.575: gtk_box_gadget_distribute: assertion 'size >= 0' failed in GtkScrollbar
(putty:2519): Gtk-CRITICAL **: 07:15:28.578: gtk_box_gadget_distribute: assertion 'size >= 0' failed in GtkScrollbar
(putty:2519): Gtk-CRITICAL **: 07:15:28.581: gtk_box_gadget_distribute: assertion 'size >= 0' failed in GtkScrollbar
```

통신확인 프로그램을 **putty &** 명령어로 실행 시킨다.

↳ 여기서 **&**를 넣어 주면 백그라운드 실행 이어서 **putty** 실행 중에도 터미널 조작이 가능해 진다.



putty를 실행 하면 오른쪽과 같은 창을 확인 할 수 있고

통신 설정을 **시리얼**, **baud rate**, **포트 명** 등을 확인하고 **open**을 통하여 작동 확인을 하게 된다.

좌측의 **avr**이라는 명칭은 설정을 **save** 한 것이고 **Save** 한 정보가 있다면 **Load**를 통하여 **putty**를 재실행 시 재빠르게 설정을 완료 할 수 있다.

Tip) 여기서 **PORT명**이 **ttyACM0** 이라는 것을 확인 하는 방법은

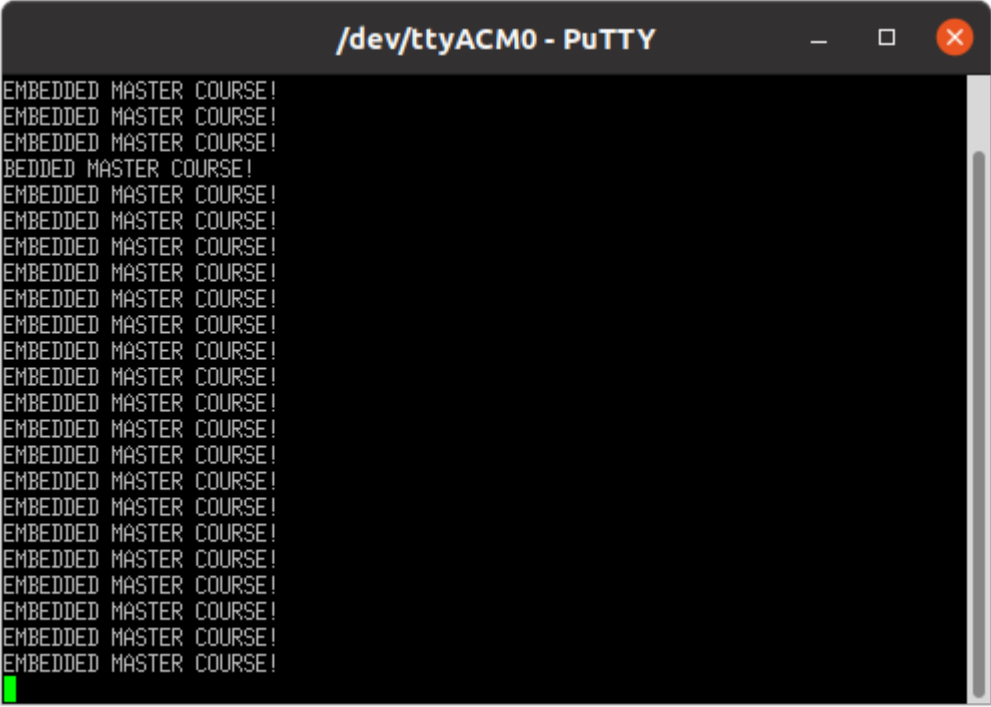
**Dmesg | grep tty**

명령어를 활용하여 찾으면 된다.

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/avr/uart_test/bin/Release$ dmesg | grep tty
[ 0.068722] printk: console [tty0] enabled
[ 6.083763] dw-apb-uart.0: ttyS4 at MMIO 0xd242d000 (irq = 20, base_baud = 115200) is a 16550A
[ 639.617661] cdc_acm 1-2.3:1.0: ttyACM0: USB ACM device
```



## UART 통신(작동확인)

A screenshot of a PuTTY terminal window titled "/dev/ttyACM0 - PuTTY". The window has a black background with white text. It displays 20 lines of the message "EMBEDDED MASTER COURSE!". A green cursor is visible at the end of the 20th line. The window has standard OS controls (minimize, maximize, close) in the top right corner.

```
/dev/ttyACM0 - PuTTY  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!  
EMBEDDED MASTER COURSE!
```

putty를 open 했을 때 최종적으로 열리는 창이며  
코딩에서 적용한 대로 문자열이 1초 단위로 찍히는  
모습을 볼 수 있다.