직렬(Serial) 통신 = 1개의 입출력 핀을 통해 8개 비트를 한 번에 전송하는 방법

병렬(Parallel) 통신 = n비트의 데이터를 전송하기 위해 n개의 입출력 핀을 사용하는 방법

대부분의 MCU에서는 Serial 통신을 사용한다.

시리얼 통신을 사용하기 위해서는 보내는 쪽(TX)와 받는 쪽(RX)에서 약속을 정해야 하는데, 이를 프로토콜(Protocol)이라고 한다.

MCU에서는 0과 1의 값만을 처리할 수 있으므로 0은 GND, 1은 VCC로 데이터를 전송하고, 받는 쪽에서는 GND와 VCC를 다시 0과 1의 이진값으로 변환하여 사용한다.

보내는 쪽(TX)와 받는 쪽(RX)이 원할하게 데이터를 교류하려면, 데이터를 보내는 속도에 대하여 프로토콜이 정해져 있어야 한다.

1) UART (범용 비동기화 송수신기)

병렬 데이터의 형태를 직렬 방식으로 전환하여 데이터를 전송하는 컴퓨터 하드웨어의 일종이다. UART는 범용이기 때문에 자료 형태나 전송 속도를 직접 구성할 수 있고 실제 전기 신호 수준과 방식이 일반적으로 UART 바깥의 특정한 드라이버 회로를 통해 관리를 받는다는 뜻이다.

즉, UART는 병렬 신호를 직렬 신호로 바꾸는 직렬 통신이며, TXD와 RXD회선을 사용해, 한 번에 하나의 비트를 송수신하는 통신방식이다. 8bit가 기본 단위다.

TXD(transmitted Data = 송신 데이터), RXD(Recieved Data = 수신 데이터)

UART에서는 보내는 쪽(TX)와 받는 쪽(RX)에서 데이터를 보내는 속도를 보율(baud rate)로 정하고 있다. 보내는 속도와 받는 속도만 같다고 해서 통신이 이루어지는 것은 아니다. TX는 필요한 경우에만 데이터를 보내고, RX도 언제부터 TX가 데이터를 보내는지, 언제부터 시작인지(idle때 HIGH의 신호를 보내고 있기 때문에) 알 수 있는 방법이 필요하다.

따라서 UART에서는 '0'의 시작 비트(start bit)와 '1'의 정지 비트(stop bit)를 사용한다.

UART는 바이트 단위 통신을 주로 사용하며, 시작 비트(start bit)와 정지 비트(stop bit)가 추가되어,

10비트 데이터를 전송하는것이 일반적이다. (parity 비트를 사용하지 않을 경우에)

-> parity bit란 오류를 잡기위한 비트다.

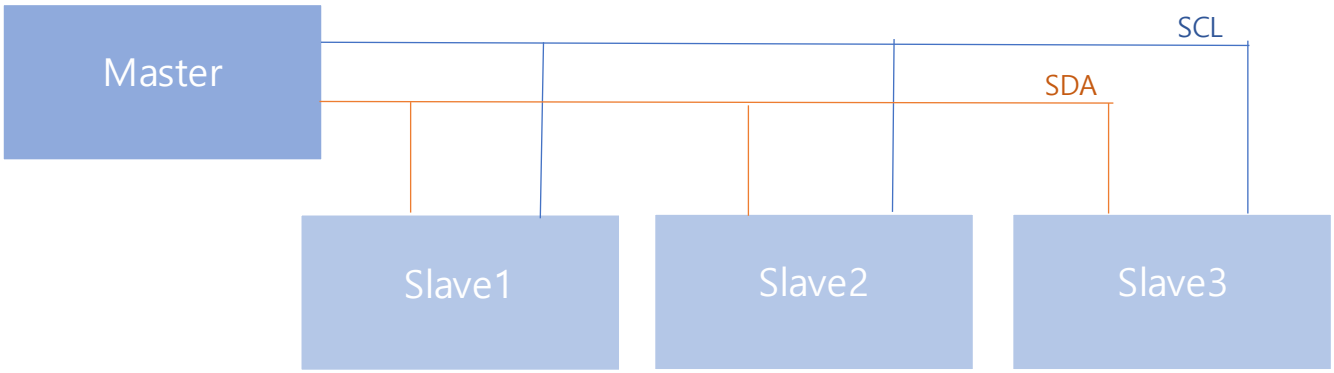
start	0bit	1bit	2bit	3bit	4bit	5bit	6bit	7bit	parity	stop

따라서 idle에서 RX에서는 신호를 계속 모니터 하고 있다가 신호가 HIGH에서 LOW 상태로 변경되면 데이터를 받을 준비를 한다.

AVR에서는 신호의 falling edge를 검출하면 1비트 길이의 16배의 클럭 속도로 신호를 샘플링한다. Falling edge에서부터 8,9,10번째 신호를 검출하여 1로 검출되는 비트가 2개나 3개가 되면 이것은 start bit가 아니고 노이즈에 의한 신호의 출렁임이라고 판단한다. 그러나 0으로 검출되는 신호가 2개나 3개가 되면 start bit으로 판단하여 다음 bit 위치에서 데이터를 읽게 된다. Data bit도 start bit와 마찬가지로 noise를 판단해, HIGH와 LOW를 결정한다.

UART는 전이중 방식(양방향 동시에 통신 가능)이기 때문에 2개의 범용 입출력 핀이 필요하다.

2) I2C



I2C는 두 개의 신호선(SCL, SDA)로 다수의 디바이스와 통신할 수 있는 방식이다. SDA는 데이터를 주고받기 위한 데이터 선이고, SCL은 송수신 타이밍 동기화를 위한 클럭선이다. I2C는 송수신을 주도하는 master와 slave로 이루어진다. Master와 Slave는 여러 개일 수 있고, Slave는 최대 127개까지 연결이 가능하다. 마스터에서 기준 clock을 생성하고, 이 clock에 맞춰 데이터를 송수신한다. 전이중 방식이었던 UART와 달리, I2C는 반이중 방식이기 때문에 한 번에 송신 혹은 수신 한가지 동작만 할 수 있다.

각 Slave는 개별 주소를 가지고 있어서, 이를 기준으로 데이터를 식별한다. 기준 클럭과 데이터는 모든 slave에 전달이 되지만 해당 주소의 slave에서만 데이터를 받아들인다.

Master에서 Slave에 데이터를 송신할 때는,

Start(1)	Address(7)	Write(1)	ACK	Data (8)	ACK	Stop(1)

Start(falling edge)가 된 후, 데이터를 받을 slave의 주소 값을 보내고, Write(LOW) bit를 보내고 난 후, slave에서 응답 신호(ACK(LOW))를 만들어, 수신을 확인한다. 그 다음 data 8 bit를 송신한 후, 다시 slave에서는 ACK를 보내 수신을 확인한 후 STOP bit(Rising edge)를 보낸다. 송수신이 제대로 되지 않았을 경우에는 NACK(HIGH)를 slave에서 보내게 된다.

Master에서 Slave의 데이터를 수신할 때는,

Start(1)	Address(7)	Read(1)	ACK	Data (8)	ACK	Stop(1)

송신과 마찬가지로, Start(falling edge)가 된 후, 데이터를 보낼 slave의 주소 값을 보내고, Read(HIGH) bit를 보내고 난 후, slave에서 응답 신호(ACK(LOW))를 만들어, 수신을 확인한다. 그 다음 data 8 bit를 slave에서 송신한 후, Master에서는 ACK를 보내 수신을 확인하고 STOP bit(Rising edge)를 보낸다. 송수신이 제대로 되지 않았을 경우에는 역시 NACK(HIGH)를 master, slave에서 보내게 된다.

➔ Master, Slave 모두 데이터를 전송하고, 수신하는 동작을 모두 사용하기 때문에 SDA 선은 INPUT과 OUTPUT을 모두 사용하게 되고, 이로 인해 floating 현상이 발생할 수 있기 때문에 Pull-Up 저항이 필요하다.

3) SPI (Serial Peripheral Interface)

SPI 의 직렬 주변기기 통신으로 1:N 의 통신 방식을 지원하는 동기식 통신 방식이다. 한 개의 Master 와 1 개 이상의 slave 가 필요하다. 통신을 위해서는 MOSI(Master OUT, Slave IN), MISO(Master IN, Slave OUT), SCLK(동기화 clock), SS(slave select) 4 개의 선이 필요하다.
SS 로 slave 를 선택하고, Master 에서 slave 로 clock(edge 방식 선택 가능)을 보내면 Master 와 Slave 사이의 data 전송이 시작된다. 즉, edge 시 slave, master 의 값을 읽는데, idle 을 HIGH 로 할지, LOW 로 할지는 선택할 수 있다. 전이중 통신방식이기 때문에, I2C 보다 속도가 빠른 것이 특징이다. 하지만 Slave 의 수가 많아지면 SS 선의 수도 많아지기 때문에 물리적으로 효율이 좋지는 못하다.