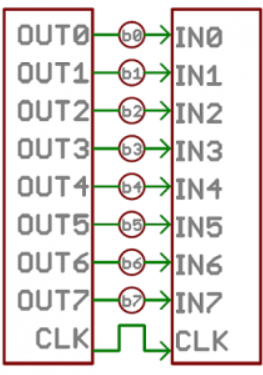
통신 패래럴, 시리얼 통신으로 나뉜다. Parallel 인터페이스는 다수의 비트를 한번에 전송하는 방법이다. 이들은 보통 8 또는 16 또는 그 이상의 라인을 통해 동시에 데이터를 보내준다.



그리고 타이밍(clock)에 맞춰 모든 라인이 같이 동작할 수 있도록 clock(CLK) 라인이 필요하므로 8-bit data bus 의 경우 9라인이 사용됩니다.

반대로 시리얼(Serial) 인터페이스는 데이터를 스트림으로 바꿔서(직렬화, serialization) 한 번에 한 비트씩 전송합니다. 아래와 같이 시리얼 인터페이스는 clock 라인을 포함 2라인으로 데이터를 전송할 수 있습니다.

[50e1ccf1ce395f962b000000](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/50e1ccf1ce395f962b000000.png)

패러럴이 고속도로라면 시리얼은 국도입니다. 대신 고속도로가 국도보다 소모되는 자원이나 비용이 많은 셈입니다.

패러럴은 빠르고 상대적으로 구현이 쉽지만 input/output(I/O) 라인을 많이 소모합니다. 아두이노 UNO 보드에 사용한다고 생각해 보시면 패러럴 방식이 소모하는 I/O 라인이 너무 많다고 느껴지실겁니다. 따라서 속도를 좀 희생하더라도 I/O 라인을 아낄 필요가 있습니다.

**시리얼 동작 방식을 알아보자.**

비동기식 시리얼 프로토콜은 몇 가지 규칙에 기반해서 동작합니다.

* Data bits,
* Synchronization bits,
* Parity bits,
* and Baud rate.

프로토콜의 세부 내용들은 유연하게 변경될 수 있도록 짜여져 있습니다. 따라서 두 기기가 시리얼 통신을 하기 위해서는 반드시 같은 설정을 사용해야 합니다.

**Baud Rate (통신 속도)**

Baud rate 는 시리얼 라인으로 전송되는 데이터 속도를 말합니다. Bits-per-second (bps) 단위로 표시하는데 이 값을 바탕으로 1 bit가 전송되는데 필요한 시간을 알 수 있습니다. 즉 1bit 데이터를 전송할 때 시리얼 라인을 high/low 상태로 유지하는 시간이며, 데이터를 받기 위해 시리얼 라인을 체크하는 시간입니다.

Baud rates 값은 어떻게든 설정할 수 있지만 통신 속도에 크게 영향을 받지 않는 경우 일반적으로 9600 bps를 사용합니다. 다른 표준 baud are 값으로 1200, 2400, 4800, 19200, 38400, 57600, 115200 을 사용할 수도 있습니다.

값이 높을수록 전송/수신 속도가 빠르지만 115200 를 초과할 수는 없습니다. 많은 마이크로 컨트롤러에서 이 값이 상한선으로 사용됩니다. Baud rate 값이 너무 높거나 양쪽의 설정 값이 틀릴 경우 데이터 수신에 문제가 발생합니다.

**Framing the data (데이터 구조)**

전송에 사용되는 데이터 패킷은 아래와 같이 구성됩니다. 각각의 구성요소는 Start bit를 제외하고 가변적인 크기를 가질 수 있습니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/50d2066fce395fc43b000000.png)

**Data chunk (데이터 영역)**

실제 전송할 데이터를 말합니다. 5~9 bit 를 사용할 수 있는데 8 bit가 기본이긴 하지만 다른 사이즈를 가질수도 있습니다. 예로 7 bit 크기인 경우는 7-bit ASCII 문자를 보내는데 적합니다.

데이터 크기가 설정되면 데이터의 엔디안 처리에 대해서도 송신, 수신측에서 합의가 되어야 합니다. After agreeing on a character-length, both serial devices also have to agree on the endianness of their data. Is data sent most-significant bit (msb) to least, or vice-versa? If it’s not otherwise stated, you can usually assume that data is transferred least-significant bit (lsb) first.

**Synchronization bits (동기화 비트)**

Synchronization bits 는 2개 혹은 3개의 특수한 비트로 전송되는 데이터 패킷의 시작과 끝을 알립니다. 위 그림에서 start bit, stop bit가 해당됩니다. Start bit 는 1 bit, stop bit 는 1~2 bit 로 설정할 수 있습니다. (일반적으로 stop bit는 1bit를 사용)

Start bit 는 idle 상태(데이터 전송이 없는 상태)에서 active 상태로의 변화(1-->0)로 표시되며 stop 비트는 반대로 idle 상태로 변화함(1)을 의미합니다.

**Parity bits (패리티 비트)**

Parity(패리티) 비트는 매우 단순한 저레벨 에러 보정 방법으로 홀수 또는 짝수(odd or even)로 체크합니다. 데이터 영역에 해당하는 5-9 bit 를 모두 더해서 홀수, 단수인지를 패리티 비트(0 또는 1)에 기록하고 전송합니다. 수신측에서도 마찬가지의 작업으로 패리티 비트와 비교해서 수신 데이터에 문제가 있는지 체크합니다. 예로 **0b01011101** 데이터를 전송하면 5개의 1이 있으므로 홀수이고 패리티 비트는 1로 설정됩니다.

패리티 비트는 선택사항(optional)이며 잘 사용되지 않습니다. 노이즈에 취약한 환경에서 써볼만 하지만 송신, 수신측에서 추가적으로 연산에 대한 부담을 해야합니다. 그리고 문제 발생시 데이터를 다시 받을 수 있도록 처리해줘야 합니다.

예제 : 9600 8N1

**9600 8N1** - 보통 이런 방식으로 Serial 통신의 설정을 표시합니다.(본 예제가 가장 많이 사용되는 시리얼 설정) 이 설정의 의미는 9600 통신속도(baud rates), 8 data bits, no parity, and 1 stop bit 설정을 사용한다는 의미입니다. 이 설정으로 실제 데이터를 보내면 아래처럼 데이터가 전송됩니다.

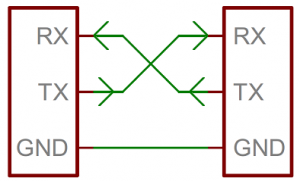
[ASCII](http://www.asciitable.com/) 문자 ‘O’, ‘K’ 를 보낼 때 아래와 같이 2개의 패킷이 만들어집니다. 대문자 O에 해당하는 ASCII 값은 79 이므로 이진수로 01001111 입니다. K 는 01001011 입니다. 그리고 설정에 표시되진 않았지만 기본 least-significant bit (lsb) first 방식으로 보내집니다. 그래서 O와 K를 나타내는 비트가 역전되어(가장 우측의 비트부터 우선 전송) 전송됩니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/50d2201ece395f0a15000001.png)

9600 bps 속도로 보내므로 각 비트는 1/(9600 bps) = 104 µs 시간에 보내집니다. 1byte = 8bit 전송할 때 start bit, stop bit가 더해져서 10bit 패킷을 사용하므로 초당 960 byte 를 전송할 수 있습니다.

## 연결 방법과 하드웨어 (WIRING AND HARDWARE)

시리얼 버스는 단 2개의 선으로만 구성됩니다. 하나는 전송용(TX)이며 다른 하나는 수신용(RX) 입니다. 시리얼 통신을 사용하는 모든 모듈은 이 두 핀을 기본으로 가지고 있습니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/50e1ce8bce395fb62b000000.png)

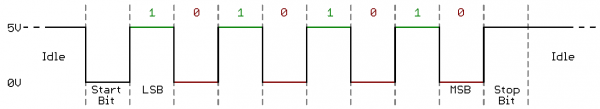
RX, TX 라는 이름은 각 장치 자신의 입장에서 바라봤을 때 라인이 담당하는 역할입니다. 따라서 두 장치를 연결할 때는 TX-RX, RX-TX로 엇갈리게 연결되어야 합니다.

시리얼 인터페이스는 full-duplex 또는 half-duplex로 동작할 수 있습니다. Full-duplex 는 전송과 수신을 동시에 처리함을 의미하고 Half-duplex 는 상황에 따라 하나씩 처리함을 의미합니다.

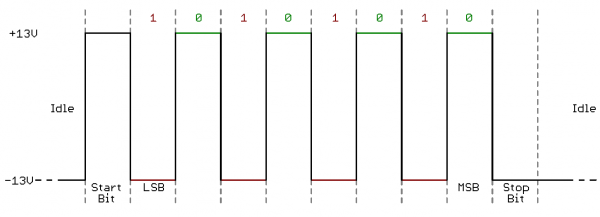
[Serial Enabled LCDs](https://www.sparkfun.com/products/10097) 의 경우는 특이하게 데이터를 받기만하고 보내주진 않습니다. 따라서 이 경우 하나의 시리얼 라인만 사용해서 단방향 통신을 하게됩니다.

**Hardware Implementation**앞선 내용들은 시리얼의 개념에 대한 내용이었습니다. 여기서는 시리얼의 하드웨어 구현, signal level 에서 살펴봅니다. (RS-232, logic-level (TTL))

마이크로 컨트롤러나 low-level IC 의 경우 TTL (transistor-transistor logic) level 에서 시리얼 통신을 구현합니다. TTL serial signal 은 마이크로 컨트롤러의 동작 전압에 의해 변화하는데 보통 0V ~ 3.3V 또는 5V 입니다. VCC level (3.3V, 5V, etc.)의 signal의 경우 idle 상태를 의미하며 bit 값으로는 1로 표현됩니다. 이 값은 stop bit 의 값(1)과 같습니다. 0V (GND) signal 은 start bit 의 값이며 0으로 표현됩니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/51142c09ce395f0e7e000002.png)

오래된 컴퓨터나 주변장치에서 종종 사용되는 RS-232 도 TTL serial 과 유사합니다. RS-232 signal 의 경우 -13V ~ 13V 로 변화하며 +/- 3V to +/- 25V 까지 허용됩니다. 이 때는 low voltage (-5V, -13V, etc.) 가 idle 상태를 나타내고(stop bit, value 1) high RS-232 signal 이 반대상태(start bit, 0 value)를 나타냅니다. 즉 TTL serial과 정반대입니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/51142cacce395f877e000006.png)

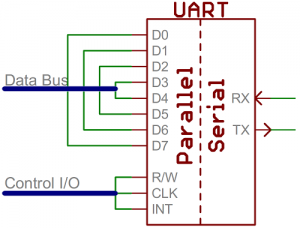
이 두가지 serial signal 표준 중 TTL 이 임베디드 회로에서 구현하기 훨씬 더 간편합니다. 하지만 전압 레벨이 낮은 경우 원거리 전송에 취약하기 때문에 RS-232 또는 더 복잡한 표준인 RS-485 가 장거리 전송용으로 사용됩니다.

두 개의 시리얼 장치를 연결하는 경우 전압 레벨을 맞춰주는 것이 중요합니다. TTL serial 장치와 RS-232 bus를 연결한다면 시그널 변환을 위한 적절한 장치가 필요합니다.

## UARTS

A universal asynchronous receiver/transmitter (이하 UART) 는 시리얼 통신을 담당하는 회로를 말합니다. 필수적으로 UART 는 패러럴과 시리얼 인터페이스를 상호 변환하는 역할을 맡게 됩니다. UART 의 한쪽은 8 혹은 그 이상의 라인 (여기에 컨트롤 라인까지 더해서) 에 물려있고 반대쪽은 시리얼 라인 RX, TX 에 물려있게 됩니다.

아래는 이걸 간단하게 표현한 UART 구조도입니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/50e1cf30ce395fb227000000.png)

UART 는 독립적인 회로로 존재하기도 하지만 일반적으로는 마이크로 컨트롤러에 탑재되어 있습니다. 마이크로 컨트롤러의 데이터시트를 보면 UART를 찾을 수 있습니다. 예로 아두이노 UNO의 경우 ATmega328 칩에 하나의 UART가 탑재되어 있습니다. 그리고 아두이노 Mega 의 경우에는 ATmega2560 칩에 4개의 UART를 탑재하고 있습니다.

UART는 전송을 할 때 동기화 비트와 패리티 비트를 더해 데이터 패킷을 생성하는 작업을 합니다. 그리고 TX 라인을 통해 정확한 타이밍으로 신호를 생성합니다.(baud rate에 따라) 반대로 수신할 때는 RX 라인에서 baud rate에 따라 신호를 읽고 데이터를 분리해 냅니다.

발전된 형태의 UART 에서는 수신된 데이터를 버퍼에 저장해서 마이크로 컨트롤러가 필요할 때 수신한 순서대로 가져다 쓸 수 있도록 해줍니다.

**Software UARTs**

마이크로 컨트롤러가 UART 를 가지고 있지 않거나 이미 사용중인 경우 마이크로 컨트롤러가 직접 제어해서(Bit-banging) 시리얼 통신을 구현할 수 있습니다. 아두이노에서 [SoftwareSerial](http://arduino.cc/en/Reference/SoftwareSerial) 라이브러리가 하는 역할이 바로 이것입니다. 하지만 이 방식은 프로세서의 자원을 많이 소모하고 UART 만큼 정확하지는 않습니다. 비동기식 시리얼 통신은 1:1 통신 방식이기 때문에 시리얼 라인이 이미 사용중일 때 SoftwareSerial 을 사용하게 됩니다.

## 주의사항

**RX-TX, TX-RX**

반드시!! 시리얼 통신 연결은 TX-RX 를 엇갈리게 해주어야 합니다. 다른 모듈의 일반적인 라인 연결 방법과 반대이기 때문에 실수하기 쉽습니다.

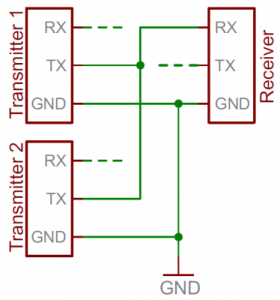
**Baud Rate Mismatch**

가끔 데이터가 들어오긴 하는데 데이터가 깨지거나 수신 타이밍이 들쑥날쑥 한 경우가 있습니다. 많은 경우 이건 시리얼 통신 장치 양쪽의 Baud rate 가 맞지 않아서입니다.

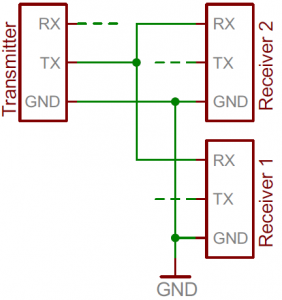
**Bus Contention**

시리얼 통신은 1:1 통신을 위해 디자인 되었습니다. 하지만 임의로 1:N 통신처럼 연결할 수도 있습니다.

예로 GPS 모듈을 아두이노에 아래와 같이 연결할 수도 있습니다. 아래 그림에서 Transmitter 1은 PC가 되고 Transmitter 2는 GPS, Receiver는 아두이노가 됩니다. 이렇게 해도 동작이 되긴 합니다.(적절한 보호 회로가 있는 경우..) 다만 동시에 Transmitter 1, Transmitter 2 에서 데이터를 전송하는 경우 문제가 발생합니다. 두 개의 Transmitter 가 TX 라인을 적절히 조절해서 사용할 수 있다면 한번 해보시길... 일부 아두이노의 경우 PC 에서 업로드 할 때 TX, RX 에 연결된 다른 장치와 충돌하지 않도록 보호 회로가 있다고 합니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/50d249a8ce395faa6f000000.png)

아래 방법은 앞서와는 반대로 Transmitter(아두이노) 하나에 두 개 이상의 Receiver(주변장치)를 연결한 경우입니다. Clock 라인만 추가된다면 I2C의 동작 방법과 유사합니다. 여러개의 장치가 데이터를 수신만 하는 경우라면 이 방법으로 문제없이 연결할 수 있습니다. 예를 들어 Serial LCD 모듈처럼 RX 라인에 아두이노의 TX 라인이 연결되기만 하면 동작하는 경우가 해당합니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2015/09/5114296cce395f8f7d000004.png)

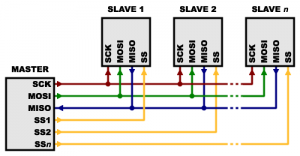
Serial, I2C, SPI 의 상세 원리를 다루는 문서에서는 우리가 흔히 얘기하는 Serial 통신(비동기식 시리얼 통신)을 UART 통신이라 표기합니다. 패러럴/시리얼 통신과 같이 넓은 의미에서의 시리얼 통신과 구분하기 위해서입니다.

## I2C 통신 개요

I2C (Inter-Integrated Circuit, 또는 TWI - Two Wire Interface) 는 복수개의 슬레이브 장치가 복수개의 마스터 장치와 통신하기 위한 프로토콜입니다. 앞서 살펴본 SPI 와 마찬가지로 하나의 완성품을 구성하는 요소들 간의 근거리 통신을 위해 고안되었습니다.

시리얼 통신에서 살펴보았다시피 **비동기식 시리얼 통신(이하 UART)**은 클럭을 맞춰줘야 하고 데이터 라인으로 들어오는 신호를 항상 주시해야 하기 때문에 오버헤드가 있으며 하드웨어가 복잡해지는 단점이 있습니다. 결정적으로 UART 통신은 1:1 통신만 가능합니다.

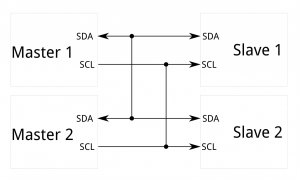
반면 동기식 시리얼 통신인 SPI 는 클럭(CLK) 라인을 이용해 데이터 라인을 동기화 하므로 하드웨어 구조도 간단하고 1:N 통신이 가능합니다. 송신용 핀과 수신용 핀이 분리되어 있고, full-duplex(동시 송수신) 연결을 이용해 10만 비트의 전송 속도를 지원하기도 합니다. 하지만 SPI 통신은 통신에 필요한 핀이 많아지는 단점이 있습니다. 하나의 장치를 연결하는데 4개의 라인 (CLK, MOSI, MISO, SS)이 필요하고, 추가로 장치를 더할 때마다 라인이 하나씩 추가됩니다. 그리고 N:N 통신은 할 수 없습니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2016/04/50e5d529ce395f2f7a000000.png)

I2C 통신은 SPI 통신의 이런 단점들을 보완할 수 있는 동기식(synchronous) 시리얼 통신 방법입니다. UART 통신처럼 단 두 라인만 사용하고 1008 개의 슬레이브 장치를 지원합니다. 또한 N:N 통신도 지원이 가능합니다. (단 마스터 장치끼리 통신은 불가) 하드웨어 요구사항이 SPI 보다 복잡하긴 하지만 UART 통신보다는 간단합니다. 통신 속도면에서도 SPI 와 UART 통신의 중간쯤 됩니다.(10kHz/40kHz)

## I2C 하드웨어

I2C 통신은 두 개의 라인 - SCL, SDA 를 사용합니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2016/04/51adfda8ce395f151b000000.png)

SCL 은 SPI 의 CLK 처럼 클럭 신호를 생성해서 전달하는 역할을 하고 SDA 는 실제 데이터가 전달되는 라인입니다. 클럭 신호는 반드시 마스터 장치가 생성하는데 슬레이브 장치가 데이터 전송을 지연하기 위해 클럭 신호를 만드는 경우도 있습니다. 이것을 clock stretching 이라고 합니다.

I2C 통신은 UART/SPI 통신과는 달리 I2C 하드웨어(bus driver) 장치가 "open drain" 속성을 가진다는 점입니다. 이 말은 I2C 통신을 사용하는 장치 하나가 데이터 전송을 위해 라인에 LOW 신호를 넣고 있으면 다른 장치들은 이걸 강제로 HIGH로 바꿀 수 없다는 의미입니다. I2C 통신의 두 라인 SDA, SCL 은 모두 [풀업 저항(pull-up resistor)](http://www.hardcopyworld.com/gnuboard5/bbs/board.php?bo_table=lecture_pract&wr_id=3&page=2)에 연결되어 있으므로 통신라인이 사용이 끝나면 자동으로 HIGH 상태로 복귀됩니다.

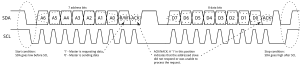
또한 [풀업 저항](http://www.hardcopyworld.com/gnuboard5/bbs/board.php?bo_table=lecture_pract&wr_id=3&page=2)은 동작 전압이 다른 장치들이 연결되어 통신하는데 발생하는 문제점들을 없애줍니다. 풀업 저항이 두 장치가 사용하는 전압보다 낮은 레벨로 유지시켜주기 때문에 별도의 level-shifting 회로가 필요치 않습니다. (두 장치의 전압차가 큰 경우는 I2C level shifter 가 필요함)

## 프로토콜

I2C 통신의 데이터 구조는 UART/SPI 보다 복잡합니다. UART/SPI 에서 문제가 되는 부분들을 소프트웨어적으로 처리하기 때문입니다.

**기본 구조**

I2C가 전송하는 메시지는 2개의 프레임 - 주소(address) 프레임과 데이터(data) 프레임으로 구분할 수 있습니다. 주소 프레임은 마스터가 전송하는 데이터가 어떤 슬레이브로 가는지 알려주기 위한 프레임입니다. 데이터 프레임은 실제 전송하려는 데이터가 담긴 프레임입니다.

[](http://www.hardcopyworld.com/ngine/aduino/wp-content/uploads/sites/3/2016/04/51ae0000ce395f645d000000.png)

**통신 시작 조건**

데이터 통신을 시작하기 위해서는 주소 프레임부터 전송해야 합니다. 이를 위해 마스터 장치는 SCL(클럭) 라인을 HIGH 상태로 두고, SDA(데이터) 라인을 LOW로 변경합니다. 이 신호는 모든 슬레이브 장치들에게 데이터를 전송할 준비가 되었다는 신호로 인식됩니다. 만약 회로에 마스터 장치가 여러개 붙어 있다면 먼저 SDA 라인을 LOW 로 잡는 이가 제어권을 갖습니다.

**주소 프레임(address frame)**

주소 프레임은 항상 모든 통신 데이터의 앞에 위치합니다. 데이터 라인으로 처음 수신되는 1byte 중 앞선 7 bit 데이터(MSB, most significant bit)가 I2C 주소를 나타냅니다. 그리고 마지막 1 bit 는 읽기(read, 1, 데이터 요청) 또는 쓰기(write, 0, 데이터 전송) 동작임을 표시합니다.

프레임의 9번째 bit 값은 NACK/ACK 비트입니다. 이 비트는 마스터가 전송한 주소 정보(8bit)에 해당하는 장치가 만들어줘야 하는 신호입니다.(즉, 해당하는 슬레이브 장치가 잠시 SDA 라인의 컨트롤 권한을 가짐) 마스터 장치는 이 신호를 보고 데이터가 전송이 성공적인지, 실패라면 앞으로 어떤 동작을 할지 판단해야 합니다. NACK/ACK 비트는 주소 프레임, 데이터 프레임의 끝에 항상 붙습니다.

**데이터 프레임(data frame)**

주소 프레임의 전송이 완료되면 이제 데이터 프레임의 전송을 시작합니다. 마스터 장치는 SCL 라인의 클럭 신호를 계속 생성하며 SDA 라인으로는 데이터가 오고갑니다. 이때 SDA 라인의 데이터 프레임은 read, write 상태를 나타내는 bit 값에 따라 마스터 또는 슬레이브가 사용합니다.

**통신 종료 조건**

모든 데이터 프레임이 전송되면 마스터 장치는 종료 신호를 만들어야 합니다. 종료 신호는 클럭 라인인 SCL 을 0->1(LOW to HIGH) 상태로 변경한 뒤, SCL 라인이 HIGH 인 상태에서 SDA 라인을 0->1 (LOW to HIGH)로 변경하면 됩니다. 따라서 반드시 모든 장치들은 반드시 SCL 라인이 HIGH 인 상태에서 SDA 라인의 상태를 변경해서는 안됩니다. 이런 동작은 잘못된 종료신호로 인식될 수 있기 때문입니다.