통신

UART, BLE, SPI, I2C

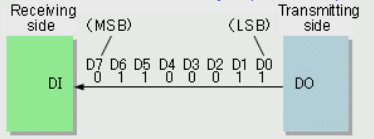
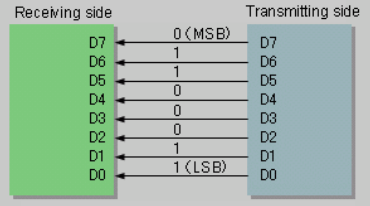
3개의 개념을 이해하기 위해서는 통신에 대해서 간략히 짚고 넘어가 보자.

통신을 나누는 기준은 정말 다양하다. 직렬/병렬, 단방향/양방향, 유선/무선, 동기식/비동기식, 베이스밴드/브로드밴드 등...

이렇게 다양한 기준중에서 이번에는 직렬/병렬의 기준으로 통신을 나누어 생각해보자.

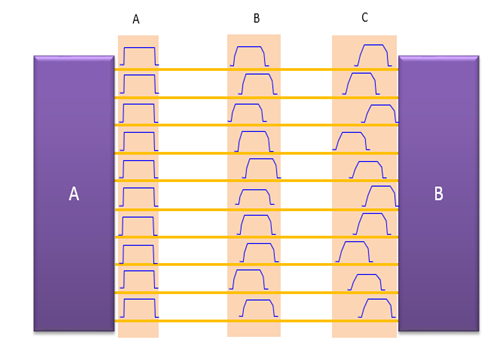
직렬 통신은 통신의 데이터를 직렬의 데이터로 보내는 형태를 의미하게 된다.

반대로 병렬 통신은 데이터를 병렬의 데이터로 보내는 형태를 의미하게 된다.



그렇다면 이러한 통신들은 왜 나누어졌을까? 가장 간단한 이유는 속도와 비용의 문제 때문이다. 통신이 좋다라고 하는 것은 빠르고 정확하게 데이터를 보내면 된다. 직관적으로 두 가지의 통신을 비교해보면 당연히 빠르게 여러데이터를 보내는 것이 당연히 최고일 것이다. 병렬의 방식처럼 말이다. 그러나 왜 사람들은 직렬 방식의 통신을 사용하는 것인가?

이러한 이유는 앞서말한 비용과 신뢰성의 문제 때문이다. 선이 많아지고 포트가 많아질수록 당연히 비용은 비싸진다. 또한 여러 개의 선들을 활용해 통신을 하기 때문에 타이밍의 문제가 생길 수 있다. => 해당 문제로 신뢰성이 떨어진다.



따라서 이러한 이유 때문에 직렬 통신을 주로 많이 사용하고 있는 추세이다. 물론 컴퓨터내의 장치와 데이터를 주고받을 때는 짧은 거리이면서 고속의 속도를 요함으로 이러한 경우 병렬 통신을 많이 활용하기도 한다. 결론적으로는 무조건적으로 좋은 통신이란 없고 경우에 따라서 적절한 통신방식을 선정하는 것이 좋다.

이때 직렬 통신의 방식에도 정말 다양한 방법이 존재한다. 그 중에서 우리는 UART, SPI, I2C 3가지의 통신 방법에 대해서 이해해볼 것이다.

해당 내용을 본격적으로 들어가기전에 통신에 대한 간략한 지식을 짚고 넘어갈 필요가 있다.

1. 직렬 통신의 데이터 전송

직렬 통신은 기본적으로 1바이트 데이터를 8개의 BIT로 쪼개 한번에 1Bit씩 통신선로를 통해 전송을 하는 방식이다. 이때 8개의 BIT의 범위를 식별하기 위해서 Start, End Bit를 추가해 총 10Bit의 데이터를 송수신하게 된다. 그러나 이러한 Start, End Bit를 활용하지 않고 패리티 비트를 활용해 데이터의 범위를 식별하기도 한다.

1. Baud Rate – BPS

Baud Rate는 사실상 보내는 데이터의 속도를 의미한다고 생각하면 된다. 이때, 수신 측과 송신 측의 Baud Rate를 일치시켜야 데이터의 혼동이 없어진다.

1. 동기 통신 vs 비동기 통신

통신에서의 동기/비동기는 CLK의 유무로 판단하다. 동기의 경우 CLK신호를 활용하고 비동기의 경우 CLK신호를 활용하지 않는다는 점에서 있다. 직렬 통신에서는 CLK신호를 통해서 데이터의 범위를 식별 할 수도 있다.

1. RS-232 / RS-422 / RS-485

이러한 용어를 들어본 적이 있을 것이다. 기본적으로 직렬 비동기식 통신을 일반적으로 UART라고 부른다. 이때, UART에서 나오는 신호는 보통 TTL 신호 레벨을 갖기 때문에 노이즈에 약하고, 통신거리에 제약이 있다. 따라서 이러한 TTL 신호를 입력받아 노이즈에 강하고 통신거리를 길게 해주기 위해 인터페이스 IC를 활용하는데 이런 대표적인 예가 RS-232, RS-422, RS485라고 한다.

사전지식을 활용하면 이제 3가지 통신은 쉽게 이해해볼 수 있다.

1. UART

UART는 병렬 데이터를 직렬 데이터로 바꾸어 전송하는 하드웨어이다. 이때 TX(송신), RX(수신)이 따로 있기 때문에 양방향으로 데이터를 전송할 수 있다. 일반적으로 통신에서의 비동기는 전송데이터를 CLK 신호에 맞춰서 전송하지 않는다는 의미이다. 그렇게 되면 수신자의 경우 송신자가 보내는 데이터의 범위를 식별하기 위해 여러 방법을 활용하기도 한다. 앞서 설명한 Start, End Bit를 활용하는 방식, Baud Rate를 일치시켜 보내는 방식을 활용하여 데이터를 보내는 통신 방법이다.

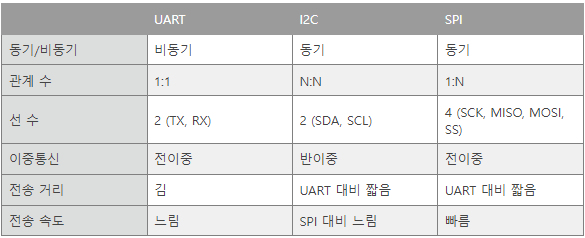
1. I2C

I2C의 경우 UART와는 다르게 동기식 직렬 통신 방법이다. 이때 데이터를 전송하는 선이외에 CLK신호를 주는 선이 따로 존재하며 데이터를 전송하는 선이 1개이기에 한번에 송수신을 동시에 진행하지 못한다. UART와는 다르게 하나의 하드웨어가(Master) 여러 하드웨어(Slave)에 연결할 수 있으며, 이때도 데이터를 전달하는 선은 1개뿐이라 여러 개와 연결되어있어도 통신은 하나씩 해결할 수 있다.

1. SPI

SPI의 경우 I2C의 업그레이드 버전이라고 생각하면 편하다. I2C의 경우 데이터 선이 하나뿐이였다면, SPI는 송신(MOSI), 수신(MISO) 2가지의 데이터 통신 선을 가지고 있다. 따라서 송신을 하면서 받을 수 있다. 그러나 I2C와의 차이점이라면 여러 개의 Master를 가질 수 없다는 점이다.

간단히 3가지 통신에 대해서 이해해보았다. 해당 표를 보고 내용을 정리하면 아래와 같다.



이러한 정보를 활용해 각 통신이 어디에 쓰이면 좋을지 생각해볼 수 있다.

참고: https://hydroponicglass.tistory.com/entry/%EC%9E%84%EB%B2%A0%EB%94%94%EB%93%9C%EC%97%90%EC%84%9C%EC%9D%98-UART-I2C-SPI-%EC%82%AC%EC%9A%A9-%EB%B9%84%EA%B5%90

--------------------------------------------------------------------------------------------------------------

CheckSum

CheckSum을 알기전에 데이터 전송에 대해서 이해해야한다.

일반적으로 우리가 데이터를 전송하기 위해서 어떻게 해야할까?

웹으로 생각해본다면, OSI 계층의 Data Link Layer를 생각해볼 수 있다. 또한, 다양한 직렬/병렬 통신을 통해서도 데이터를 전송할 수 있다. 다양하게 데이터를 전송하는데 이러한 과정중에 데이터의 손실이나 오류가 발생하게 된다면 어떻게 조치를 해야할까?

이럴 때 다양한 데이터 오류 검출 방법을 활용하고 있다.

대표적으로 패리티 검사, CRC, 체크썸, 해밍코드들이 있다.

그중에 우리는 CheckSum 방식의 오류 검사 방식에 대해서 간략히 이해해 보자

* CheckSum

체크썸의 기본 정의는 “중복 검사의 한 형태로 오류 정정을 통해 공간이나 시간 속에서 송신된 자료의 무결성을 보호하는 단순한 방법이다” 라고 정의합니다. 그러나… 이렇게 이해하면 쉽게 알수 없습니다.

단계를 밟아가며 이해해봅시다.

앞서 CheckSum이 데이터를 전송할 때 오류를 검출하기 위해 사용되는 하나의 방법이라고 이해했습니다.

그러면 도대체 어떻게 오류를 검사할까? 라고 생각해봅시다. 데이터를 보낼 때 결론적으로 우리는 컴퓨터를 이용해 보내게 됩니다. 그러면 자연스래 Bit단위의 데이터들이 01010100010101010101 이렇게 값들이 들어오게 됩니다. (물론 다른 곳에서는 **0x84, 0xF2, 0x10, 0x55 이런 데이터가 들어오기도하고 10 45 366, Hello 등 정말 다양한 데이터들이 들어올 수 있는데 우리는 이러한 데이터들을 컴퓨터가 이해하는 언어로 변형해 데이터를 송수신하게 됩니다.)**

**그러면 순서를 따라서 어떻게 오류를 검출하는지 확인해 봅시다.**

1. **들어온 데이터들을 모두 합한다.**
2. **합한 데이터의 캐리 니블을 버린다. (최상위 니블, 가장 앞에 위치한 4bit)**
3. **2의 보수를 취해 CheckSum을 만든다**
4. **수신부에서 모든 데이터들을 합한다.**
5. **수신부에서 모두 합한 데이터와 CheckSum을 더한다.**
6. **수신부에서 2데이터를 더한 값에서 캐리니블을 버린다.**
7. **0x00이 나오지 않으면 오류로 판단한다.**

**이렇게 말로 설명하면 어려우는 당연히 예시를 들어 한번더 이해해봅시다.**

ex)

**0x84, 0xF2, 0x10, 0x55 데이터가 들어왔다고 가정합니다.**

송신부

1) 모든 byte 데이터를 더합니다.

0x84 = 132,

0xF2 = 242,

0x10 = 16,

0x55 = 85

132 + 242 + 16 + 85 = 475

475 = 0x1DB

2) 캐리니블을 버립니다.

0x1DB = 0001 1101 1011 (2) -> 1101 1011

3) 2의 보수를 취해 체크섬을 만듭니다.

기존 :        1101 1011

1의 보수 :  0010 0100

2의 보수 :  0010 0101

체크섬 :     0010 0101 = 0x25

수신부

1) 모든 byte 데이터와 체크섬을 더합니다.

0x1DB + 0x25 = 0x200

2) 캐리니블을 버립니다.

0x200 = 0010 0000 0000 (2) -> 0000 0000

3) 0x00이 나왔기 때문에 오류가 발생하지 않은 것으로 판단합니다.

이런식으로 검출을 하는 방식이 바로 Checksum 방식입니다.

결론적으로 정리해 본다면, 데이터의 송수신 과정에서 외부의 간섭이나 다양한 이유로 인해 데이터가 변형되거나 손실될 수 있다. 이러한 오류를 잡아내기 위해서 데이터 송수신과정에서 오류를 검출하기 위한 여러방식을 활용하는데 그 중 하나의 방식이 CheckSum이다. Checksum은 데이터들의 합/캐리니블 버림/보수를 통해서 오류를 검출할 수 있다.

이렇게 Checksum을 이해하시면 되겠습니다.

참고

https://junboom.tistory.com/32

--------------------------------------------------------------------------------------------------------------

SVN

SVN은 Suvbersion의 약자로 형상관리 툴입니다. 형상관리는 간략히 이해해보면 SW 적으로 코드들의 변경사항을 체계적으로 관리, 추적, 통제하는 것을 의미합니다.

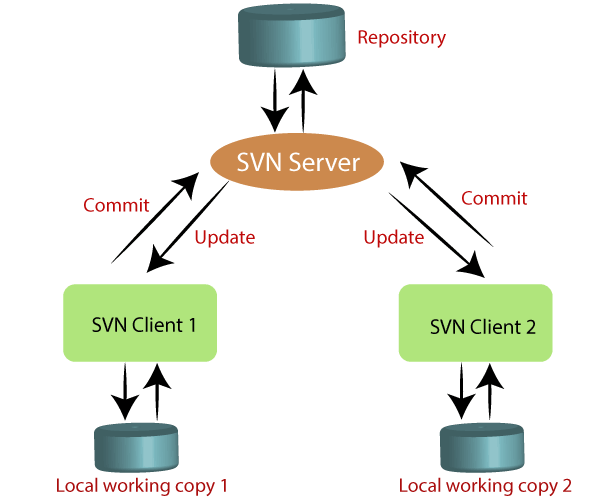
즉, 쉽게 말하면 우리가 작성한 다양한 코드들을 저장하고, 분리하고, 추적하고, 통제하는 것을 도와주는 도구라고 생각하면 됩니다.

형상관리 툴에는 CVS, GIT도 하나의 형상관리 툴로 활용되곤 합니다. 그러나 현 트렌드에 가장 많이 사용하는 형상관리 툴은 GIT이 활용되고 있습니다.

그럼에도 아직까지 많은 회사에서 SVN을 활용하기도 하는데 GIT과 다른 장점이 있기 때문에 SVN을 활용하고 있습니다.

그러면 본격적으로 SVN에 대해서 이해해 봅시다.

SVN의 경우 서버/클라이언트로 나누어서 사용할 수 있습니다. 기본적으로 클리아인트에서 코드를 작성하고 서버의 환경에 코드를 저장하여 활용합니다.



해당 그림처럼 각 클라이언트들은 자신의 Local 환경에서 작업한 코드를 SVN Server에 보내게 됩니다. 그러면 Server에서는 해당 코드들을 Repository를 통해 관리하고 있습니다.

그렇다면, 우리가 svn을 활용하기 위해서는 Server를 만들고 Client로 작업을 하게된다는 의미입니다. (물론 협업이 필요없다면 단순 Local 작업으로 해도 충분합니다.)

SVN의 경우 활용법이 굉장히 단순합니다. 서버와 클라이언트를 생성하였다는 가정하에

1. 코드를 add
2. 코드를 Commit
3. 타 컴퓨터로 코드를 Checkout
4. 타 컴퓨터로 코드를 Update

1, 2번 과정을 통해서 Server에 코드를 넣어두고 3, 4번 과정을 통해서 서버의 코드를 가져와 활용할 수 있습니다.

SVN을 잘 활용하기 위해서는 간단히 용어를 숙지해둘 필요가 있다.

* Repository: 프로젝트 파일 및 변경 정보가 저장되는 장소
* Import: 빈 Repository에 맨 처음 파일들을 채우는 것
* Export: 버전 관리 파일들을 뺀 순수 파일만 빼내는 것
* Checkout: 저장소에서 최신 버전의 소스코드를 최초로 받아오는 것 / Repository에서 프로젝트 관련 파일들을 받아온다
* Update: 로컬 저장소에 있는 파일들을 저장소의 최신 버전으로 받아 오기
* Commit: 로컬 저장소의 변경된 내용을 서버로 전송 / Checkout한 파일의 수정사항을 갱신
* Revert: 로컬 저장소의 내용을 이전 상태로 돌림
* Add: 버전관리 대상으로 파일 등록
* Trunk: 개발 소스를 commit 했을 때 개발 소스가 모이는 곳 / 프로젝트에서 가장 중심이 되는 디렉토리, 소스와 파일 포함
* Branch: trunk에서 분리/복사한 소스로 버전별 배포판을 만들거나 trunk와 별도로 운영환경을 위한 안정화된 소스 관리 목적으로 사용
* Tag: 특정 시점의 상태 보존 목적으로 사용 장기적으로 1.0, 1.1 등 버전 별로 소스 코드를 따로 저장

참고로 클라이언트의 경우 Window 환경에서는 TortoiseSVN을 활용하고 서버의 경우 VisualSVN Server를 활용한다.

참고

https://na27.tistory.com/211

--------------------------------------------------------------------------------------------------------------