

# 응답하라 XBee

- XBee를 이용한 무선 통신 가이드 -

메카솔루션

[www.mechasolution.com](http://www.mechasolution.com)

무단 배포 및 공유를 허락합니다.

## 목차

1. 시작하기에 앞서 .....	3
2. XBee 에 대한 소개 및 선택 가이드 .....	3
I. 802.15.4 (XBee Series 1) vs ZigBee Mesh (XBee Series 2) .....	3
II. 안테나에 따른 분류 .....	4
III. XBee 레귤러 모듈 vs XBee 프로 모듈 .....	4
IV. 시중에 판매중인 XBee 모듈 비교 .....	4
V. XBee 모듈 쉴드 및 액세서리 .....	6
VI. XBee 통신 모듈: AT VS API .....	7
3. 가장 쉽고 빠른 XBee 사용 방법 .....	7
I. 원격 센서값을 XBee 를 통해 컴퓨터에 전송하는 예제 .....	7
A. 아두이노와 XBee 의 연결방법 .....	8
II. 아두이노와 아두이노간에 XBee 로 무선통신 예제 .....	14
4. X-CTU 소프트웨어 설치 및 Configuration 변경 .....	18
I. X-CTU 소프트웨어 설치 .....	19
II. X-CTU Modem Configuration (XBee 파라미터 설정하기) .....	23
A. XBee 시리즈 1 파라미터 설정 .....	23
B. XBee 시리즈 2 파라미터 설정 .....	25
5. 프로젝트 1: 키보드 입력으로 RC 탱크 구축하기 .....	29
6. 프로젝트 2: RSSI 를 이용한 XBee 간 거리 센싱하기 (AT-API) .....	34
7. 프로젝트 3: RSSI 를 이용한 XBee 간 거리 센싱하기 (API-API) .....	39

## 1. 시작하기에 앞서

본 강좌 혹은 튜토리얼은 XBee를 이용한 무선통신을 활용하여 수업을 진행할 수 있도록 고안되었습니다. 특히, 제품 개발자, 엔지니어, 연구원 등 XBee를 이용하여 마이크로컨트롤러 및 컴퓨터 간의 통신을 함에 있어, 두꺼운 데이터시트를 일일이 찾아보는 번거러움을 줄여서 바로 사용할 수 있도록 "따라하기"에 초점을 맞추어서 설명하려고 노력하였으며, 최근 들어 많이 사용하는 마이크로컨트롤러인 아두이노 플랫폼을 적극 활용하여 추후 다양한 아두이노 애플리케이션과도 응용할 수 있도록 하였습니다. Digi사의 기술자료 및 매뉴얼을 통해서 공부를 할 수도 있지만, 방대한 양의 명령어 및 동작 조건등으로 인해서 매우 복잡함을 알게 되었고, 어떻게 하면 쉽게 설명할 수 있을까에 대한 고민을 하던 끝에 작업을 시작하였습니다. 기본적으로 아두이노에 대해 기초적인 지식과 경험을 가지고 있다고 가정된 후에 설명을 하였으며, 아두이노를 처음 접하시는 분들께 뻔뻔한 아두이노 매뉴얼 (소프트웨어 설치, 라이브러리 사용방법, 및 각종 센서와 액추에이터에 예제가 들어 있습니다.)을 다운로드 받으셔서 익히신 후에 본 튜토리얼을 보시길 추천드립니다.

관련된 참고문헌은 본 문서의 마지막에 인용하였으며, 이 튜토리얼을 만들 수 있도록 허락해주신 많은 익명의 블로거 및 소스를 오픈해주신 여러 엔지니어분들의 노고에 감사를 드립니다.

## 2. XBee에 대한 소개 및 선택 가이드

### I. 802.15.4 (XBee Series 1) vs ZigBee Mesh (XBee Series 2)

오류 없이 데이터를 목적지까지 무선 전송하기 위해서는 무선통신 모듈은 각각 나름의 규약인 프로토콜을 갖게 됩니다. 예를 들어, IEEE 802.11 혹은 IEEE 802.15.1과 같이 Wifi나 블루투스의 프로토콜을 이야기할 수 있는데, XBee는 장치간의 저가격, 저속도 유비쿼터스를 지향하기에 이에 맞는 802.15.4를 채택하게 됩니다. ZigBee Mesh 프로토콜은 세 개의 노드를 가지고 있으며 802.15.4에 비해서 더 많은 기능을 가지고 있습니다. 802.15.4와 ZigBee Mesh 프로토콜의 비교는 XBee의 시리즈 1과 2의 비교로 이어지는데, 802.15.4 프로토콜을 채택한 시리즈 1의 경우 P2P 네트워크에 효율적이고, 간단한 무선통신을 하기 위해서 별도의 파라미터 구성이 필요 없이 공장 출하 그대로 사용할 수 있는 편리함이 있습니다. ZigBee Mesh 프로토콜을 채택한 시리즈 2의 경우, 시리즈 1에 비해서 추가적인 기능들 (star topology routing, encryption, application services)과 보다 먼거리 통신이 가능하다는 장점이 있지만, 사용하기 위해서는 별도의 파라미터 구성이 필요하다는 번거로움이 있습니다.

✓ 전력 사용에 크게 민감하지 않고(50mA), 별도의 파라미터 변환 없이 바로 사용 > XBee

## 시리즈 1

- ✓ 추가적인 기능들을 사용하고, 40mA의 전류 사용과 파라미터 변환도 괜찮으면 > XBee 시리즈 2

## II. 안테나에 따른 분류

안테나에 따른 분류로는 칩 안테나, 와이어 안테나, u.FL 안테나, 트레이스 안테나 등이 있습니다. 칩 안테나, 와이어 안테나, 그리고 트레이스 안테나는 별도의 안테나를 연결할 필요없이 내장되어 있는 안테나를 사용하면 되고, u.FL 안테나는 안테나를 직접 디자인하거나 u.FL 안테나를 연결하여 사용할 수 있습니다. 칩 안테나와 트레이스 안테나는 공간을 줄일 수 있는 장점이 있고, 와이어 안테나는 신호의 세기가 방사형이 아닌 거의 원형으로 나타나기 때문에 신호의 세기를 이용하여 거리를 측정하는 용도로 사용할 수 있습니다.

- ✓ 안테나를 직접 디자인하여 XBee를 사용 > u.FL 안테나 모듈
- ✓ 신호의 세기를 이용하여 거리를 측정 > 와이어 안테나 모듈
- ✓ 안테나가 차지하는 공간을 줄여서 소형화 > 칩 안테나 혹은 트레이스 안테나 모듈

## III. XBee 레귤러 모듈 vs XBee 프로 모듈

XBee 레귤러 모듈과 XBee 프로 모듈을 비교한다면, 단연코 신호의 세기와 전력, 그리고 이에 따른 통신 거리입니다. XBee 레귤러 모듈은 약 100m 정도의 통신 거리이지만, XBee 프로 모듈은 1km 이상이기 때문에 긴거리 통신을 위해서는 XBee 프로를 사용하는 것이 바람직합니다.

- ✓ 100m 이하의 통신 > XBee 레귤러 모듈
- ✓ 100m 이상의 통신 > XBee 프로 모듈

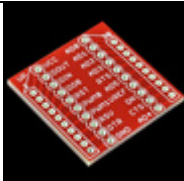

## IV. 시중에 판매중인 XBee 모듈 비교



	이름	거리	전력	주파수	프로토콜	전송파워	속도	안테나
	XBee 1mW Chip Antenna - Series 1	100m	50mA	2.4GHz	802.15.4 시리즈 1	1mW	250kbps	칩

	XBee 1mW U.FL Connection - Series 1	100m	50mA	2.4GHz	802.15.4 시리즈 1	1mW	250kbps	미포 함
	XBee 1mW Wire Antenna - Series 1	100m	50mA	2.4GHz	802.15.4	1mW	250kbps	와이 어
	XBee 1mW Trace Antenna - Series 1	100m	50mA	2.4GHz	802.15.4	1mW	250kbps	PCB
	XBee 2mW PCB Antenna - Series 2	120m	40mA	2.4GHz	ZigBee Mesh	2mW	250kbps	PCB
	XBee 2mW RPSMA - Series 2	120m	40mA	2.4GHz	ZigBee Mesh	2mW	250kbps	미포 함
	XBee 2mW U.FL Connection - Series 2	120m	40mA	2.4GHz	ZigBee Mesh	2mW	250kbps	미포 함
	XBee 2mW Wire Antenna - Series 2	120m	40mA	2.4GHz	ZigBee Mesh	2mW	250kbps	와이 어
	XBee Pro 63mW PCB Antenna - Series 2B	1.6km	295mA	2.4GHz	ZigBee Mesh	63mW	250kbps	PCB
	XBee Pro 63mW RPSMA - Series 2B	1.6km	295mA	2.4GHz	ZigBee Mesh	63mW	250kbps	미포 함

	XBee Pro 50mW U.FL Connection - Series 2	1.6km	295mA	2.4GHz	ZigBee Mesh	50mW	250kbps	미포 함
	XBee Pro 63mW Wire Antenna - Series 2B	1.6km	295mA	2.4GHz	ZigBee Mesh	63mA	250kbps	와이 어
	XBee Pro 60mW PCB Antenna - Series 1	1.6km	215mA	2.4GHz	802.15.4	60mA	250kbps	PCB
	XBee Pro 60mW U.FL Connection - Series 1	1.6km	215mA	2.4GHz	802.15.4	60mA	250kbps	미포 함
	XBee Pro 60mW Wire Antenna - Series 1	1.6km	215mA	2.4GHz	802.15.4	60mW	250kbps	와이 어

#### V. XBee 모듈 쉴드 및 액세서리

	이름	기능
	Breakout Board for XBee Module	XBee의 촘촘한 핀들간의 간격을 브레드보드에 끼울 수 있도록 해 줌
	XBee Shield	아두이노 우노에 적층하여 사용할 수 있어서 편리함

	XBee Explorer Regulated	아두이노 프로 미니와 직접 연결할 수도 있고, 우노와 연결도 용이함 (3.3V 레귤레이터 포함)
	XBee Explorer USB	아두이노와 컴퓨터와 통신시에 컴퓨터쪽의 USB를 통해서 신호를 수신할 수 있도록 해 줌

## VI. XBee 통신 모듈: AT VS API

XBee는 AT와 API 모드를 지원하는데, AT모드는 투명(Transparent)모드라고도 하며, 공장 출하시 초기 세팅으로 제공된다. AT모드는 메시지 데이터 자체를 보내고 받을 수 있으며 단순한 전송 및 수신에 가능하다. API모드에서는 프로그래머가 필요한 정보인 도착 주소, 패킷의 형태, 그리고 체크섬과 데이터를 패키징화 한다. 또한 수신 노드는 소스 주소, 패킷의 종류, 신호 강도, 그리고 체크섬과 같은 정보와 함께 데이터로 접수한다.

✓ AT모드를 사용해야할까, 아니면 API모드를 사용해야할까?

초보자라면 별도의 프로그래밍 및 패키징화가 필요하지 않은 AT모드를 사용하는 것이 편리하다. 다만, Radio Signal Strength Indicator (RSSI) 등의 신호 세기 및 특정 주소를 프로그래밍화하여 노드간에ダイナ믹하게 통신을 하기 위해서는 유연성이 있는 API모드를 사용하는 것이 적절할 것이다.

## 3. 가장 쉽고 빠른 XBee 사용 방법

### I. 원격 센서값을 XBee를 통해 컴퓨터에 전송하는 예제

원격의 센서값을 XBee를 통해 컴퓨터에 전송할 필요가 있을 때, 어떻게 하면 쉽고 빠르게 할 수 있을까요? 구현하기 위한 준비물로는 1) 센서, 2) 아두이노, 3) XBee Explorer Regulated, 4) XBee Explorer USB, 5) USB mini-B 케이블, 6) 컴퓨터, 7) 송신부를 위한 전원이 필요합니다.

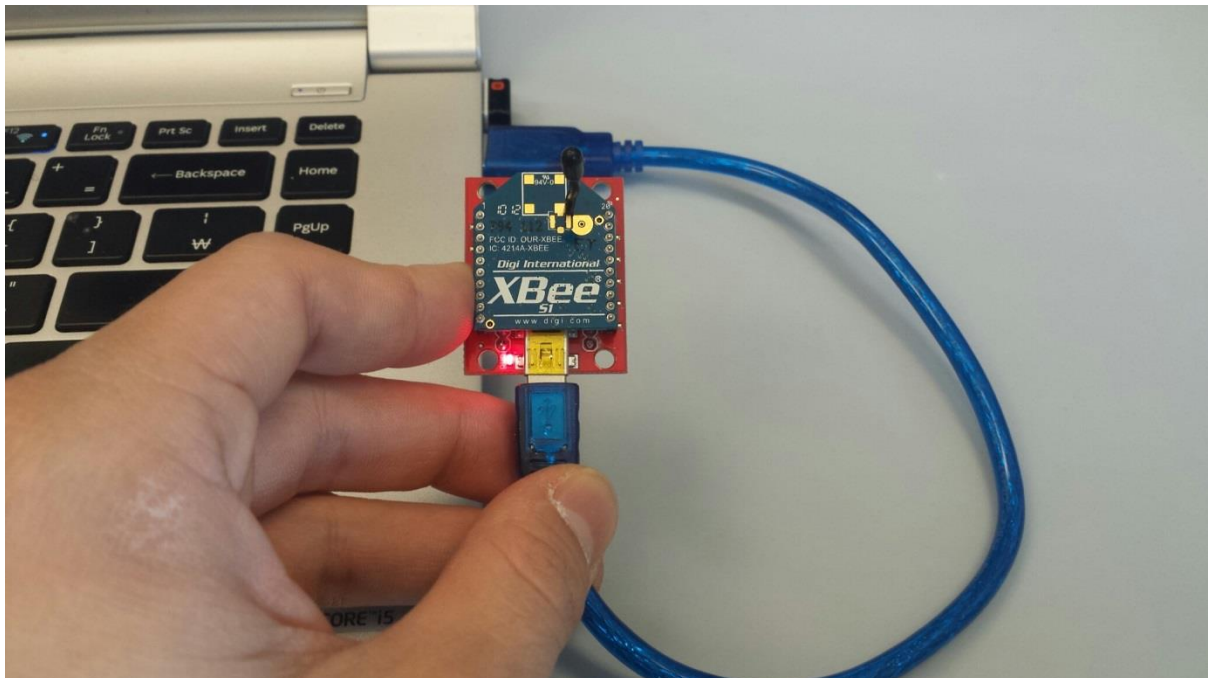
예를 들어서, 30m 밖의 거리에서 온도를 측정한 후, 그 값을 내 컴퓨터에 전송한다고 하면, 1) 아두이노에서 센서값을 읽어서 Serial.print 함수를 통해서 아두이노 시리얼포트를 통해 값을 출력합니다. 2) 이 값은 아두이노 시리얼포트에 연결되어 있는 XBee를 통해 수신부의 XBee까지 무선으로 전송되게 됩니다. 그리고 3) 전송된 값은 XBee Explorer USB를 통해 컴퓨터에 전달됩니다. 4) 컴퓨터의 USB 포트를 통해 전송된 값은 하이퍼터미널이나 시리얼 모니터링 기능을 통해서 값을 확인할 수 있습니다.

이 때, XBee는 시리즈 1을 사용합니다. 30m간의 통신을 위해서 XBee 프로를 사용하는 것은 비용과 효율 측면에서 적합하지 않고, 시리즈 2는 별도의 소프트웨어를 통해서 셋업을 해 주어야 하기 때문에 초보자들이나 쉽고 빨리 구현하기 위해서는 적합하지 않을 수 있습니다. 안테나의 종류는 와이어 안테나/ 칩 안테나/ PCB 안테나 등을 사용할 수 있는데, 어느 것을 사용하여도 무방하기 때문에 사이즈를 고려하여 선택하면 됩니다.

#### A. 아두이노와 XBee의 연결방법

##### 수신부 (받는 쪽)

XBee USB Breakout 보드에 XBee Series 1 안테나를 끼운 후, USB mini-B 케이블을 통해 컴퓨터에 연결합니다.



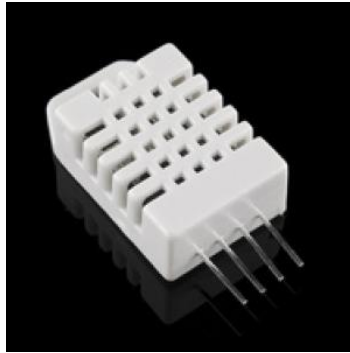
XBee USB Breakout 보드와 USB mini-B 케이블을 이용한 수신부

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=1&category=001001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=1&category=001001)

##### 송신부 (보내는 쪽)

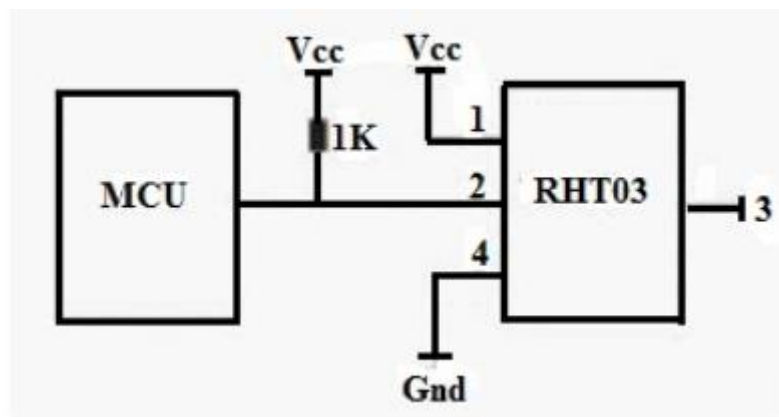
원격 모니터링을 위해서 RHT03이라는 온도습도센서를 사용해보기로 합니다. 송신부를 제작하기에 앞서, 어떤 아두이노를 사용할 것인지와, RHT03과 아두이노의 연결방법에 대한 정보를 알고 있어야 합니다. 구글 검색을 통해서 RHT03과 Arduino로 검색을 하면, 다양한 정보가 나오고, RHT03을 사용하기 위해 라이브러리를 사용하면 편리하다는 것을 알 수 있습니다.





RHT03 온도습도 센서

([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=1085&category=002010](http://mechasolution.com/shop/goods/goods_view.php?goodsno=1085&category=002010))



RHT03 온도습도 센서와 아두이노(MCU)와의 연결

사용하게 될 아두이노는 아두이노 나노입니다. 우노를 사용해도 무방하며 다른 어떤 아두이노 시리즈를 사용해도 좋지만, 송신부를 작게 만들면 좋겠다는 생각에 아두이노 나노를 사용하였습니다. 그리고, 9V 배터리를 사용해서 독립적으로 전원을 공급해주었습니다.

아두이노 나노와 RHT03과의 연결

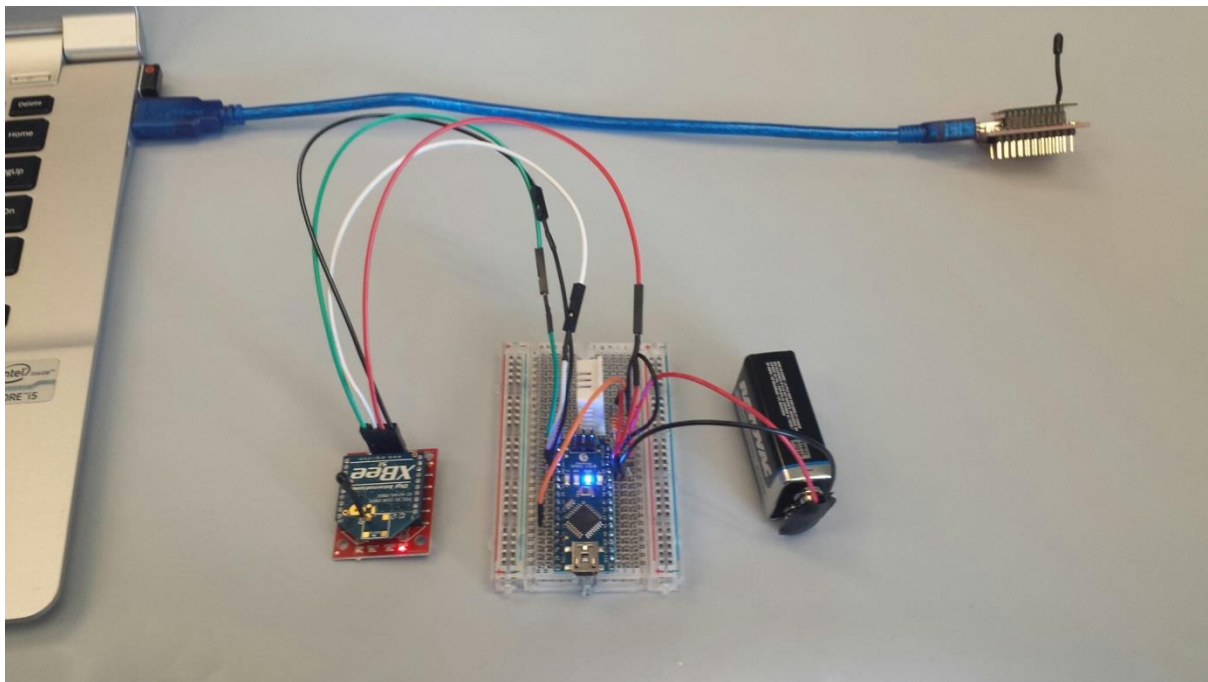
아두이노 나노	RHT03
GND	GND (정면에서 봤을 때 왼쪽에서 4번째)
5V	VCC (왼쪽에서 1번째)
D7	정면에서 봤을 때 2번째
5V + 1K 저항 (풀업저항)	정면에서 봤을 때 2번째

## 아두이노 나노와 XBee와의 연결

아두이노 나노	XBee
GND	GND
5V	5V
TX	DIN
RX	DOUT

## 아두이노 나노와 9V 배터리와의 연결

아두이노 나노	XBee
GND	-
VIN	+



XBee Regulator Breakout과 XBee 시리즈 1 안테나 모듈, RHT03 온도 습도 센서, 아두이노 나노, 그리고 9V 배터리를 사용한 송신부의 모습입니다.

## 프로그래밍

회로가 완성이 되면, 송신부의 아두이노에 프로그램을 업로드하여 온도와 습도값을 시리얼포트를 통해서 출력해주도록 합니다. 이 출력된 값은 XBee를 통해서 수신부에 무선으로 전송되게 됩니다.

먼저, 라이브러리를 다운로드 받고 파일 압축을 푼 후에 폴더명의 하이픈을 빼줍니다. 아두이노 소프트웨어에서 하이픈이 있는 폴더명을 인식하지 못하는 경우가 있습니다.

<https://github.com/nethoncho/Arduino-DHT22>

위의 링크에서 오른쪽 하단의 Download ZIP을 클릭해서 다운로드 받을 수 있습니다.

다운로드 받은 파일을 아두이노 폴더의 라이브러리에 복사한 후에 다시 소프트웨어를 오픈하면

파일-예제-dht22-Serial이라는 파일이 있고, 이를 업로드하면 됩니다. (업로드 할 때, XBee와 아두이노의 연결을 해지한 후에 업로드를 하고 업로드 후에 다시 연결해주세요, 업로드 할 때 사용하는 시리얼포트와 아두이노와 XBee간의 시리얼포트의 충돌로 인해서 업로드가 되지 않습니다.)

### 소스코드

```
#include <DHT22.h>
// Only used for sprintf
#include <stdio.h>

// Data wire is plugged into port 7 on the Arduino
// Connect a 4.7K resistor between VCC and the data pin (strong pullup)
#define DHT22_PIN 7

// Setup a DHT22 instance
DHT22 myDHT22(DHT22_PIN);

void setup(void)
{
    // start serial port
    Serial.begin(9600);
    Serial.println("DHT22 Library Demo");
}

void loop(void)
{
    DHT22_ERROR_t errorCode;

    // The sensor can only be read from every 1-2s, and requires a minimum
```

```

// 2s warm-up after power-on.
delay(2000);

Serial.print("Requesting data...");
errorCode = myDHT22.readData();
switch(errorCode)
{
    case DHT_ERROR_NONE:
        Serial.print("Got Data ");
        Serial.print(myDHT22.getTemperatureC());
        Serial.print("C ");
        Serial.print(myDHT22.getHumidity());
        Serial.println("%");
        // Alternately, with integer formatting which is clumsier but more compact to store and
        // can be compared reliably for equality:
        //
        char buf[128];
        sprintf(buf, "Integer-only reading: Temperature %hi.%01hi C, Humidity %i.%01i %% RH",
                myDHT22.getTemperatureCInt()/10, abs(myDHT22.getTemperatureCInt()%10),
                myDHT22.getHumidityInt()/10, myDHT22.getHumidityInt()%10);
        Serial.println(buf);
        break;
    case DHT_ERROR_CHECKSUM:
        Serial.print("check sum error ");
        Serial.print(myDHT22.getTemperatureC());
        Serial.print("C ");
        Serial.print(myDHT22.getHumidity());
        Serial.println("%");
        break;
    case DHT_BUS_HUNG:
        Serial.println("BUS Hung ");
        break;
    case DHT_ERROR_NOT_PRESENT:
        Serial.println("Not Present ");
        break;
    case DHT_ERROR_ACK_TOO_LONG:
        Serial.println("ACK time out ");
        break;
    case DHT_ERROR_SYNC_TIMEOUT:

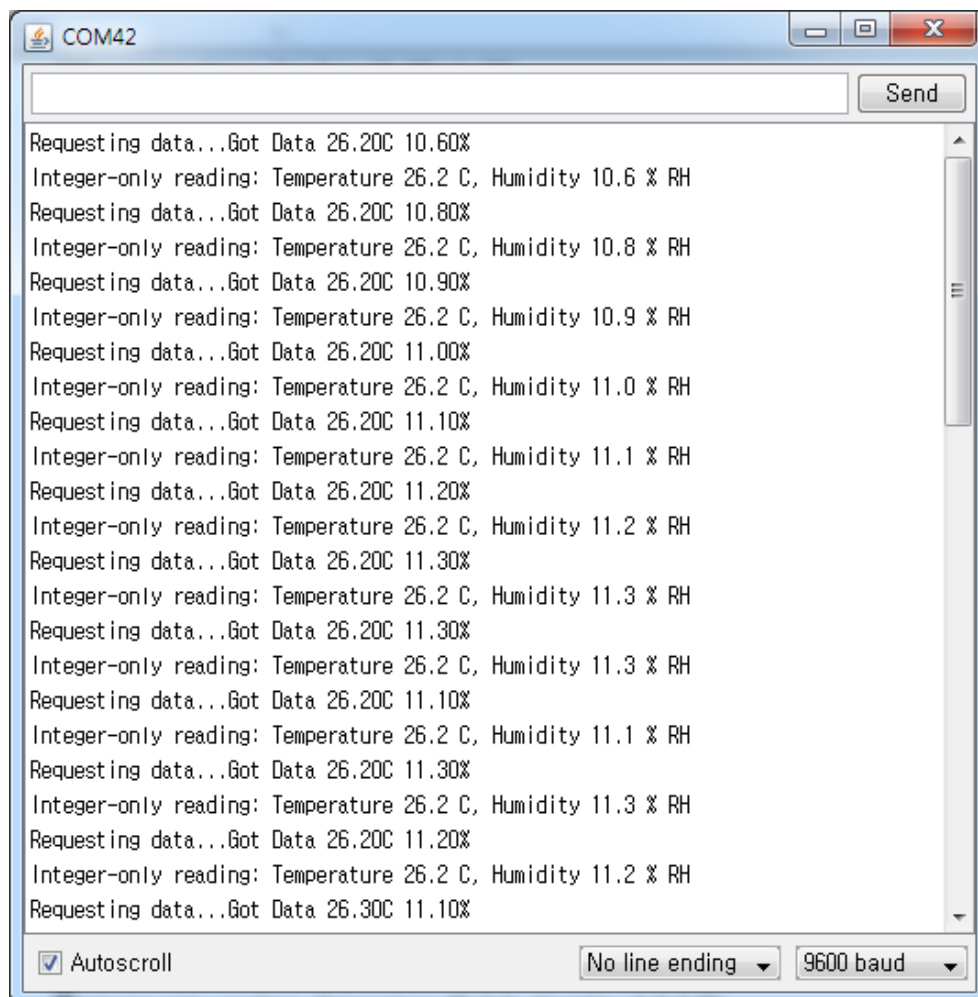
```

```

    Serial.println("Sync Timeout ");
    break;
case DHT_ERROR_DATA_TIMEOUT:
    Serial.println("Data Timeout ");
    break;
case DHT_ERROR_TOOQUICK:
    Serial.println("Polled to quick ");
    break;
}
}

```

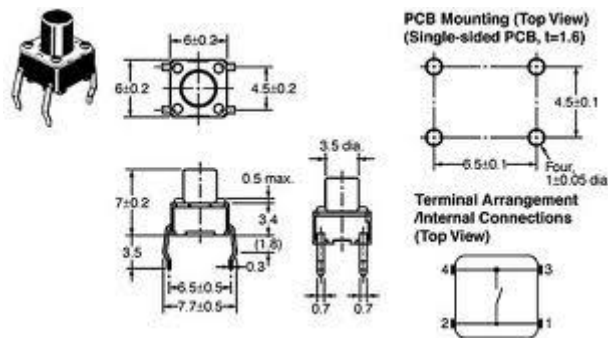
송신부의 프로그램이 업로드되면 연결된 USB 케이블을 뽑고, 9V 배터리로부터 전원을 공급받아서 완전히 독립된 모듈로 만들어줍니다. 그리고 수신부 컴퓨터에 설치되어 있는 하이퍼터미널 혹은 아두이노의 시리얼 모니터링을 통해서 값이 무선으로 전송되는 것을 다음과 같이 확인하실 수 있습니다.



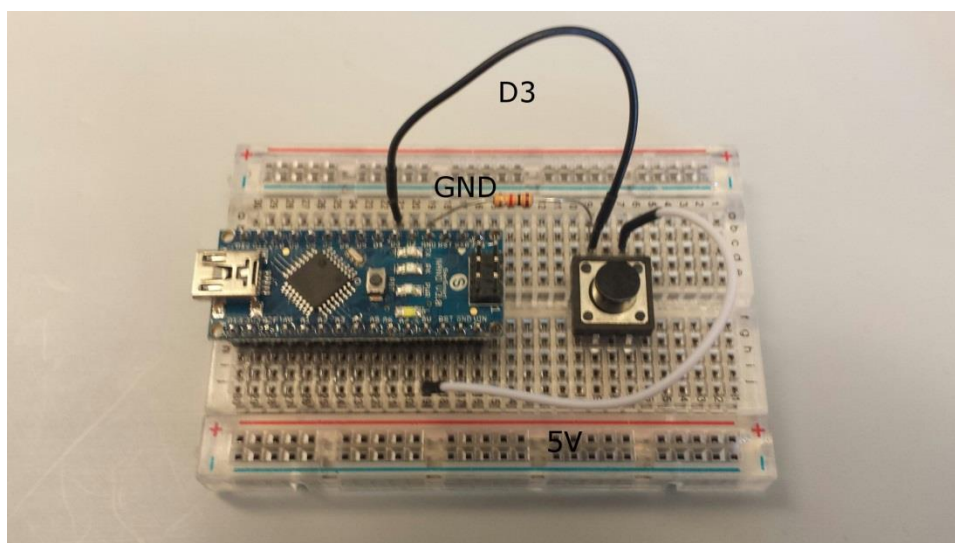
## II. 아두이노와 아두이노간에 XBee로 무선통신 예제

아두이노와 아두이노간에 통신을 해야 할 경우, 예컨대, 다수의 로봇끼리 통신을 통해서 맵핑을 하거나, 주변환경을 모니터링할 때, XBee를 사용하면 편리합니다. 이번 예제는 아두이노 A에서 버튼을 누르면 아두이노 B에서 LED가 켜지는 예제를 통해 아두이노와 아두이노간의 XBee 무선통신에 대해서 살펴봅니다.

먼저, 송신부에서는 푸시버튼을 사용하여 디지털 핀에 HIGH 혹은 LOW 레벨을 인가합니다. 그렇다면 푸시버튼의 구조를 알아야겠죠. 푸시버튼의 구조는 데이터시트를 참고하거나, 구글 검색, 혹은 멀티미터로 어느 핀끼리 연결이 되어 있는지 확인해볼 수 있습니다.



위의 그림에서와 같이 1은 2와, 그리고 3은 4와 연결이 되어 있고, 버튼을 누르게 되면 1,2와 3,4가 모두 연결이 됩니다. 버튼을 누르지 않았을 경우에는 LOW 그리고 버튼을 눌렀을 경우 HIGH를 디지털핀에 인가하기 위해서는 풀다운저항을 통해서 디지털핀으로 들어가는 신호를 강제로 다운시켜줍니다. 그리고 다른 쪽의 핀은 5V에 연결함으로써 버튼을 눌렀을 경우, 디지털핀으로 HIGH가 들어갈 수 있도록 합니다.



XBee를 연결하기 이전의 송신부입니다.

XBee와의 아두이노, 그리고 아두이노와 배터리와의 연결은 다음과 같습니다.

아두이노 나노와 XBee와의 연결

아두이노 나노	XBee
GND	GND
5V	5V
TX	DIN
RX	DOUT

아두이노 나노와 배터리와의 연결

아두이노 나노	XBee
GND	-
VIN	+

송신부의 프로그래밍은 다음과 같습니다.

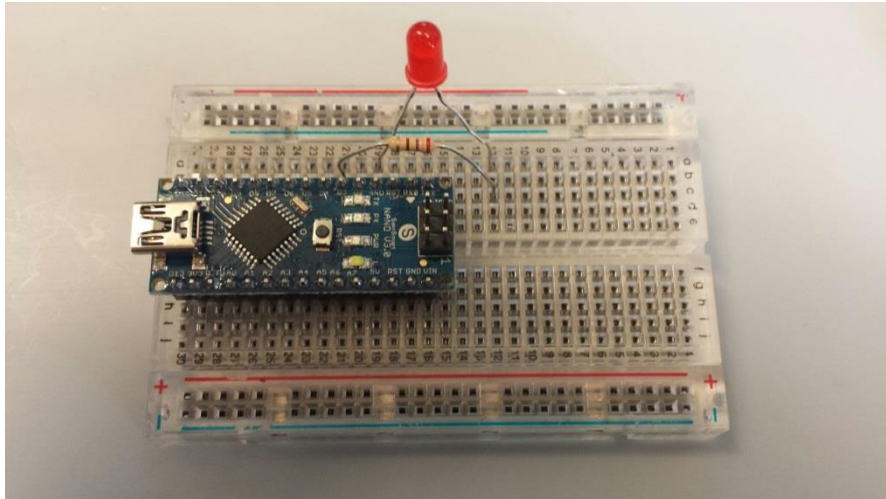
```
int SW = 3;
int SWread = 0;
void setup()
{
  Serial.begin(9600);
  pinMode(SW, INPUT);
}
void loop() {
  SWread = digitalRead(SW);
  if(SWread == 1)
  {
    Serial.println('1');
  }
  else
  {
    Serial.println('0');
  }
}
```

송신부에서는 디지털핀 3번으로 들어오는 신호에 따라서 시리얼포트를 통해 1 혹은 0을 출력하게 됩니다. 그리고 이 값은 XBee를 통해서 수신부로 무선으로 전해지게 됩니다.

수신부 측에는 1 혹은 0에 따라서 LED를 깜빡이는 프로그램을 구현해 보겠습니다.

LED와 220옴의 저항을 디지털 3번에 연결합니다.

XBee 그리고 배터리를 제외한 회로의 구성은 다음과 같습니다.



D3 -- 저항 -- LED의 +극 (긴쪽) - LED의 -극 (짧은 쪽) -- GND

마찬가지로, XBee 그리고 배터리와의 연결은 다음과 같습니다.

XBee와의 아두이노, 그리고 아두이노와 배터리와의 연결은 다음과 같습니다.

아두이노 나노와 XBee와의 연결

아두이노 나노	XBee
GND	GND
5V	5V
TX	DIN
RX	DOUT

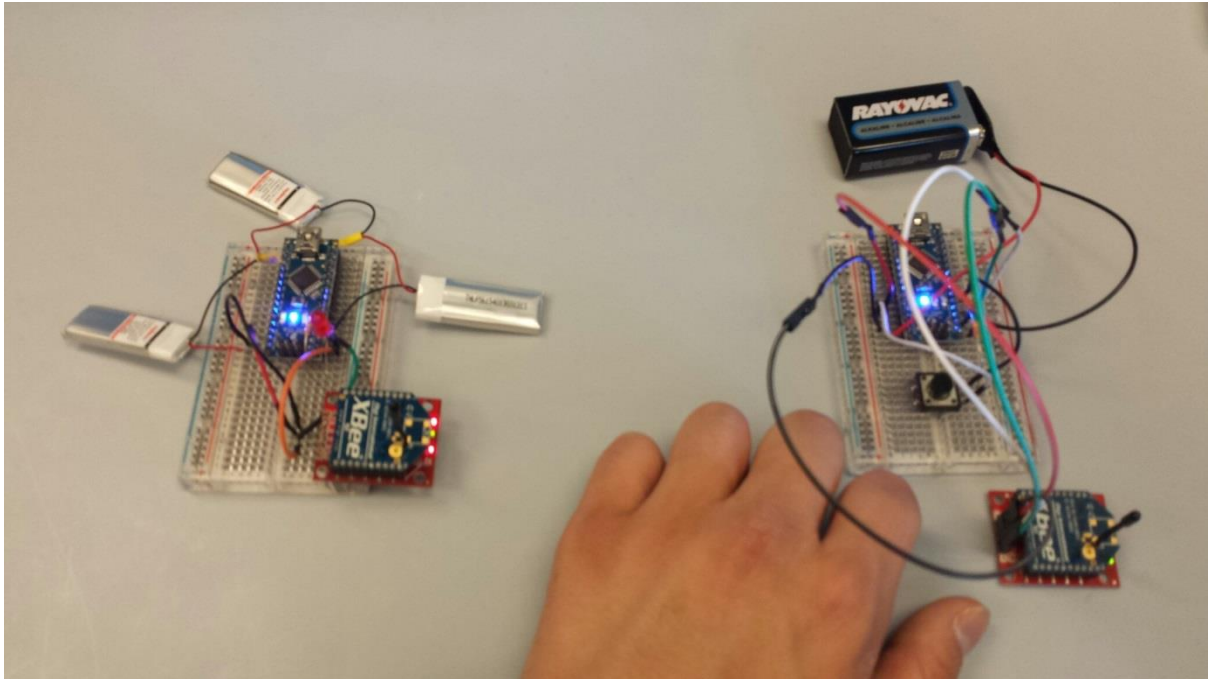
아두이노 나노와 배터리와의 연결

아두이노 나노	XBee
GND	-
VIN	+

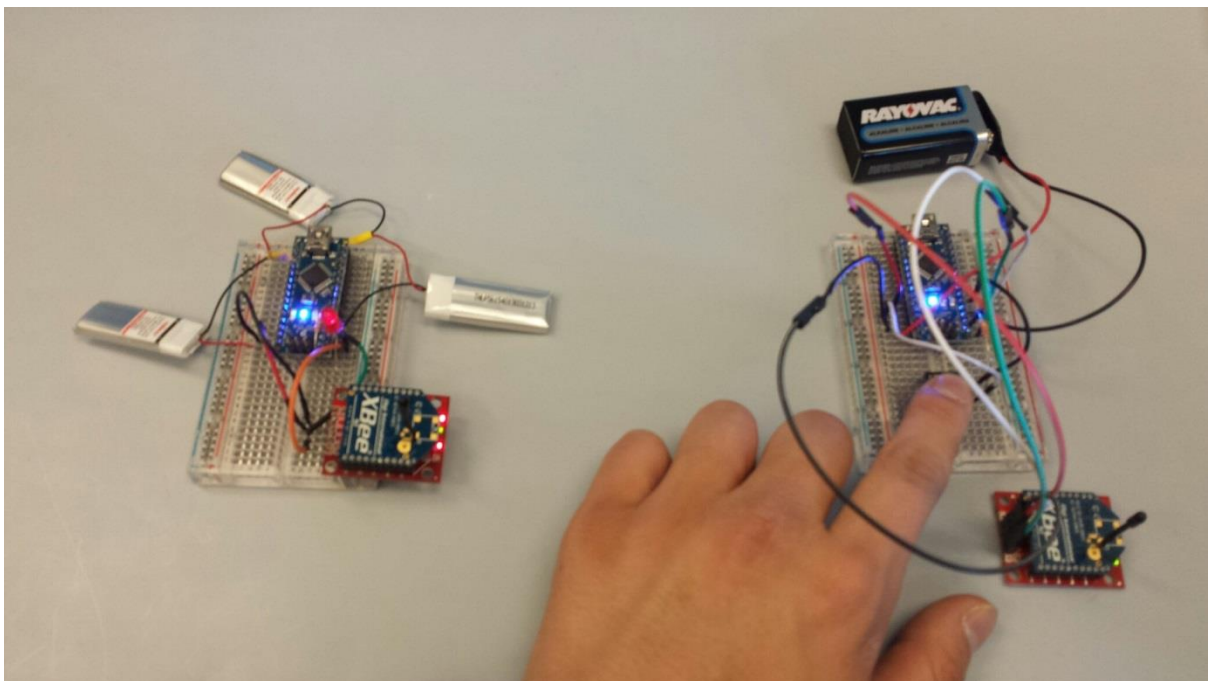


수신부의 프로그래밍은 다음과 같습니다.

```
int LEDpin = 3;
char input;
void setup()
{
    Serial.begin(9600);
    pinMode(LEDpin, OUTPUT);
}
void loop()
{
    if (Serial.available())
    {
        input = Serial.read();
        //Serial.println(input);
        if(input == '1')
        {
            digitalWrite(LEDpin, HIGH);
        }
        else
        {
            digitalWrite(LEDpin, LOW);
        }
    }
}
```



버튼을 누르기 전의 LED는 OFF



버튼을 누른 후의 LED는 ON (희미하게 보이지만 왼쪽의 빨간 LED에는 불이 들어왔습니다.)

#### 4. X-CTU 소프트웨어 설치 및 Configuration 변경

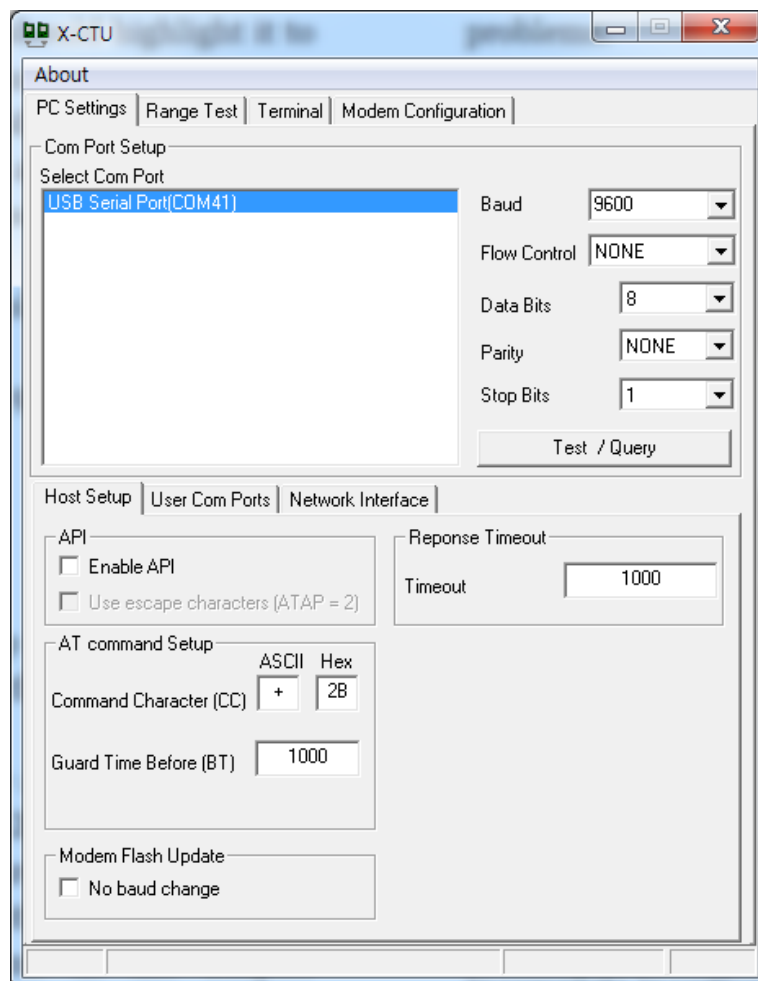
이번 강좌에서는 Digi International X-CTU 소프트웨어를 사용하여 XBee를 설정하고 컨트롤하는

방법에 대해서 소개합니다. 이 XBee 모듈은 간단한 시리얼 통신 프로토콜을 사용하여 컴퓨터, 마이크로컨트롤러 (아두이노 포함) 및 집적회로 등과 통신을 하게 됩니다. 이 때, UART (Universal Asynchronous Receiver Transmitter)를 통해서 통신이 이루어지게 되며, XBee에 기본적으로 설정된 UART속도는 9600bits/sec입니다.

#### I. X-CTU 소프트웨어 설치

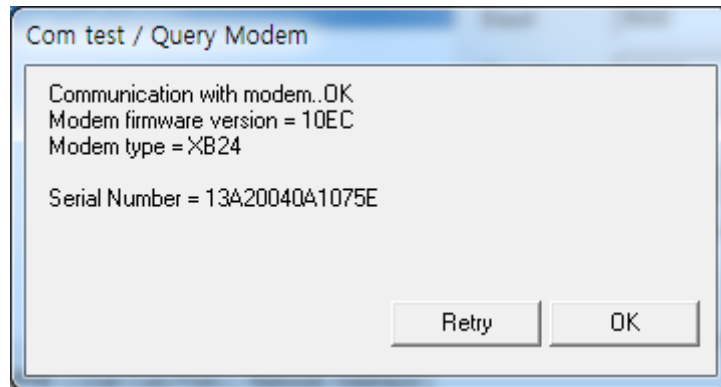
먼저, X-CTU라는 소프트웨어는 digi.com에서 다운로드 가능합니다. Digi.com의 사이트에서 검색란에 XCTU라고 검색하면 찾을 수 있고, 혹은 <http://www.digi.com/support/productdetail?pid=3352>의 링크로 바로 찾아서 OS에 맞는 버전으로 다운로드 하시면 됩니다.

윈도우 7용인 [Drivers Installer for Windows \(XP, Vista, 7 and 8\)](#) 를 다운로드 받아서 XBee를 설정해보겠습니다.

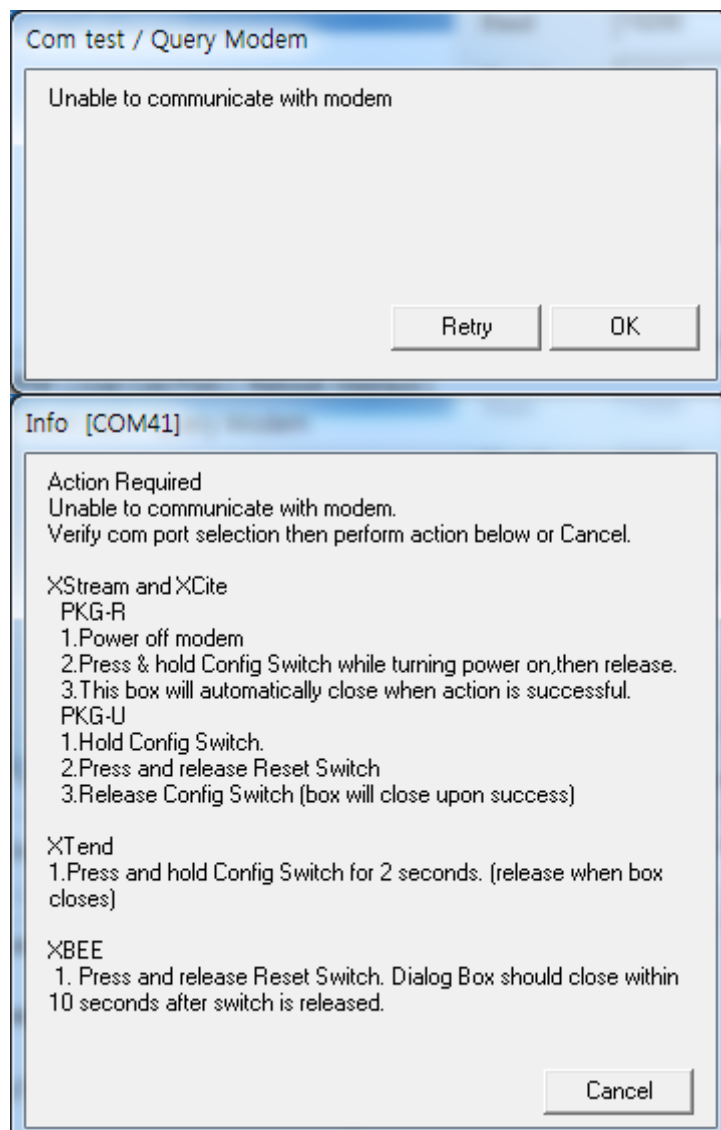


X-CTU를 실행하면 나타나는 초기화면입니다. Com Port Setup에는 현재 연결되어 있는 포트가 나열되어 있습니다.

오른쪽의 Test / Query 버튼을 누르면 선택된 포트에 대한 연결상태를 체크할 수 있습니다. 만약 다음과 같은 메시지가 생성되면 올바르게 연결되어서 사용할 준비가 된 것이고 그렇지 않으면 올바르게 연결되지 않은 것입니다.



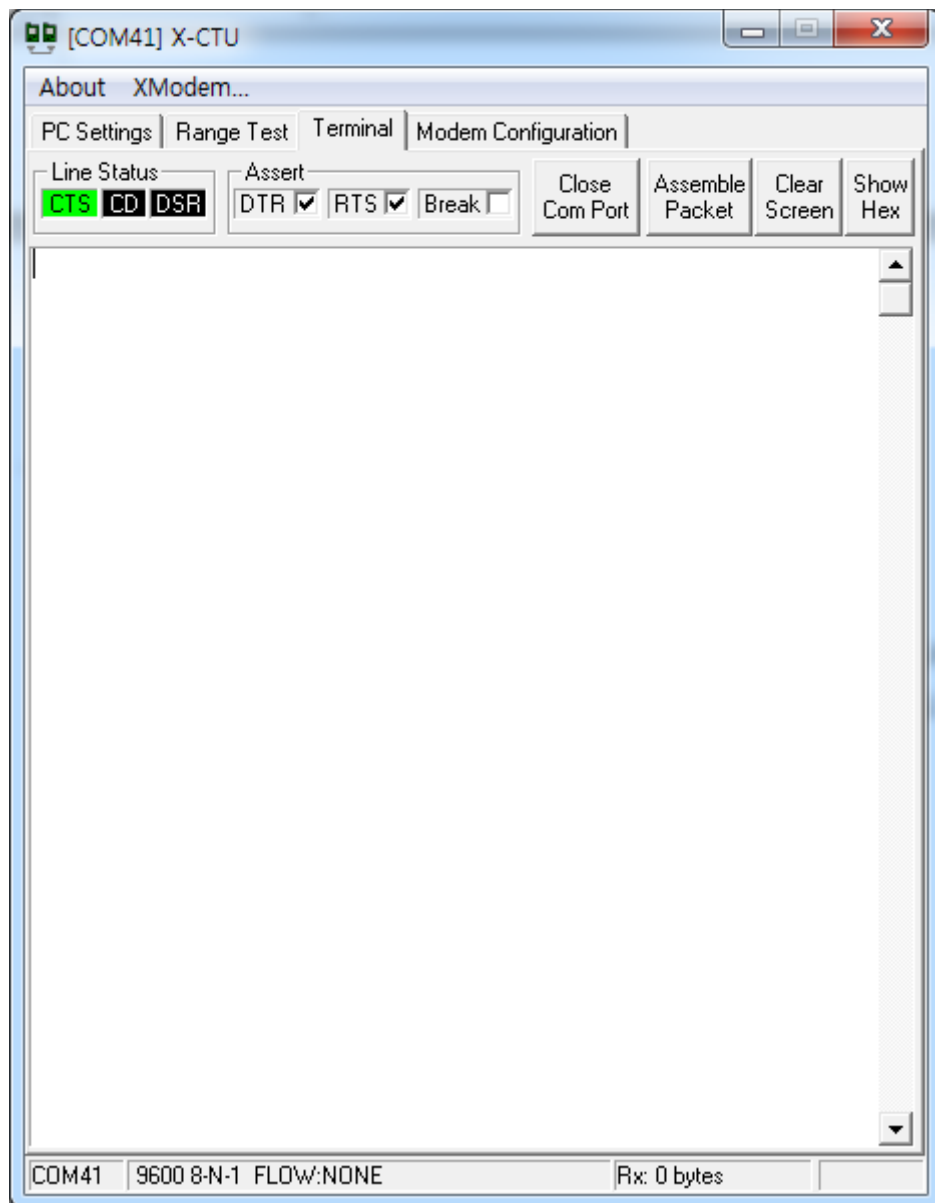
올바른 연결시



## 올바른 연결이 아닐 경우

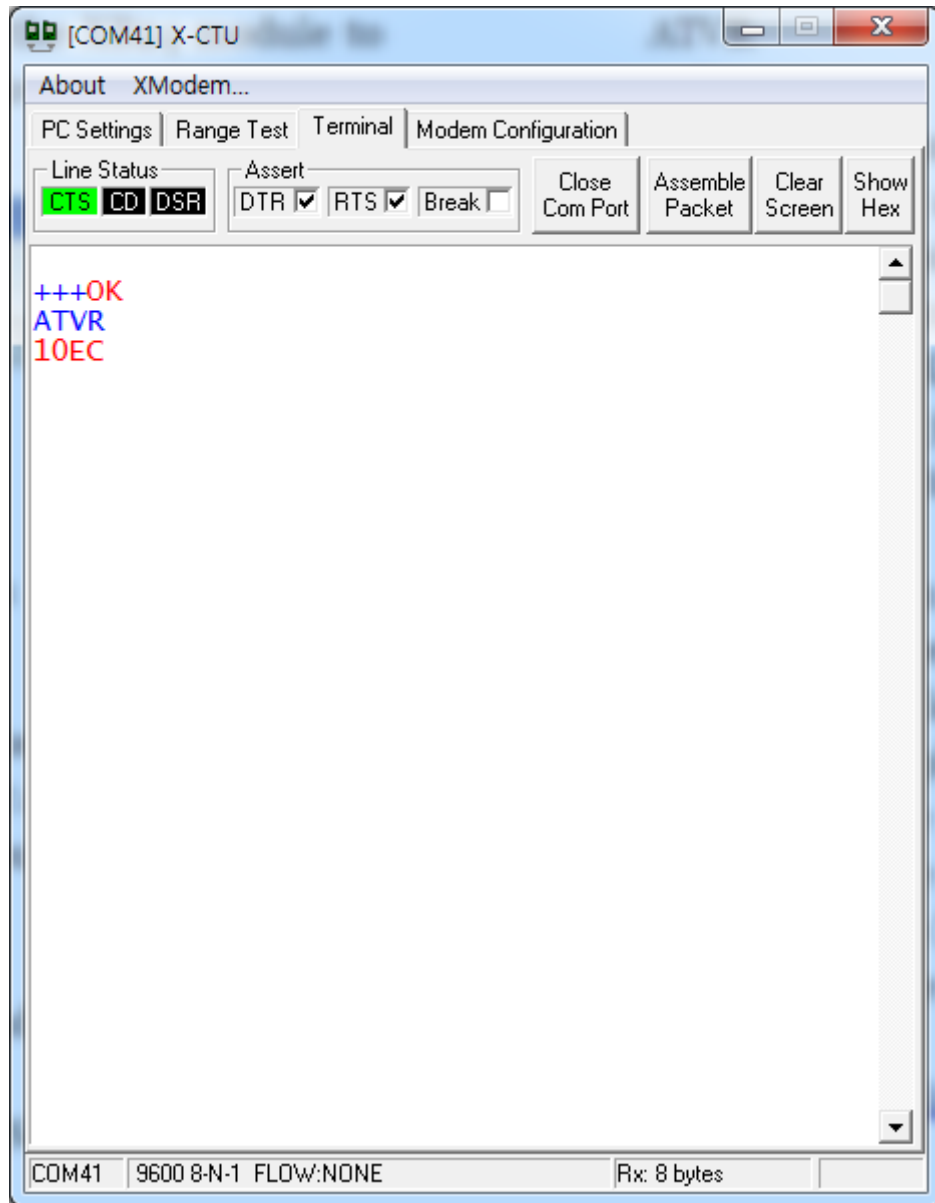
올바른 연결이 아닐 경우는 여러가지 원인이 있을 수 있는데, 하드웨어적으로 문제가 있거나, Baudrate이 9600이 아닌 다른 값으로 설정되어 있을 때, 혹은 AT 모드가 아닌 API 모드로 설정되어 있는 경우가 있습니다.

올바로 연결이 되었으면, X-CTU의 프로그램에서 Terminal 탭을 클릭해봅니다.



터미널에서는 XBee에 AT 커맨트를 이용하여 파라미터를 변경하는 기능을 수행할 수 있습니다. 예를 들어, Baud rate을 9600에서 19200으로 바꾸거나, 어드레스값을 변경하는 일을 할 수 있습니다. AT 커맨드를 사용하기 위해서는 +++를 사용하는데, +++를 입력한 후에는 엔터 혹은 다른 키 입

력을 하지 말아 주세요.



ATVR을 입력한 후에 엔터를 치면, 10EC라는 버전이 나옵니다. 이는 Firmware Version을 출력하는 명령어입니다.

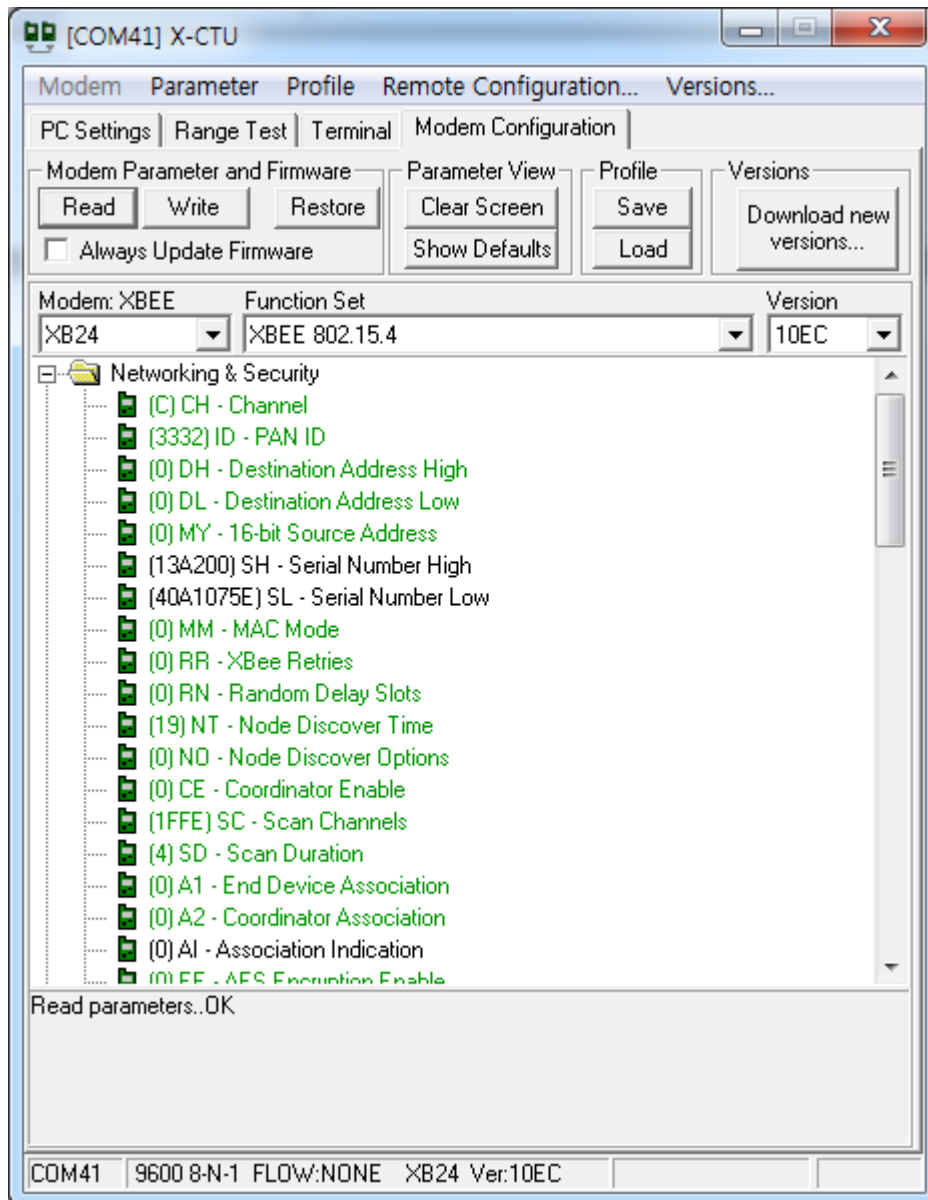
실제로 XBee의 파라미터 변경은 Terminal이 아닌 Modem Configuration에서 이루어지게 되는데,

다음 장에서는 Modem Configuration에 대해 알아보겠습니다.

## II. X-CTU Modem Configuration (XBee 파라미터 설정하기)

### A. XBee 시리즈 1 파라미터 설정

X-CTU의 Modem Configuration 탭을 클릭한 후에 Read 버튼을 누릅니다. 그러면 연결된 XBee의 파라미터들을 읽을 수 있습니다. XBee 시리즈 1은 802.15.4 프로토콜을 사용하고 있으며, 채널은 C, ID는 3332, 그리고 DH, DL, MY는 각각 0으로 설정되어 있습니다. (공장 출하시)

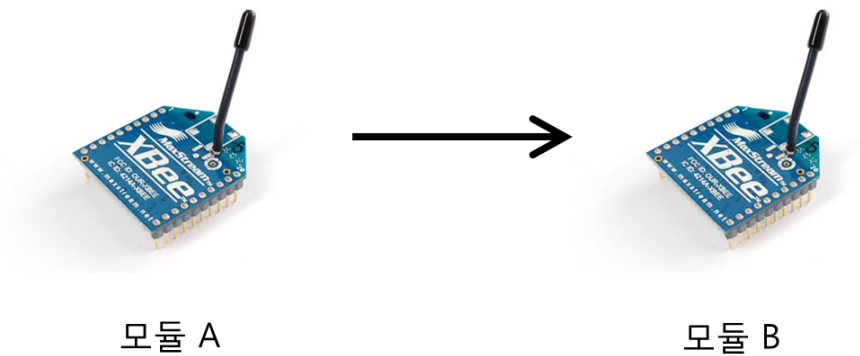


Always Update Firmware는 체크하지 않는 것이 좋습니다. 불필요하게 쓰기(Write)할 때마다 업데이트할 필요 없고, 문제를 야기할 수도 있기 때문입니다.

먼저, 자주 사용하는 파라미터에 대해서 알아보겠습니다.

Command	Description	가능한 값들	초기값
ID	XBee의 네트워크 ID	0~0xFFFF	3332
CH	XBee의 채널	0x0B~0x1A	0x0C
SH, SL	시리얼 넘버 (SH는 high 32비트, SL은 low 32비트)	0~0xFFFFFFFF	각각 다르다
MY	XBee 모듈의 내주소	0~0xFFFF	0
DH, DL	도착 주소 (DH는 high 32비트, DL은 low 32비트)	0~0xFFFFFFFF	DH = 0, DL = 0
BD	Baud rate	1200bps~1152bps	3(9600bps)

### 양방향 통신 주소 설정



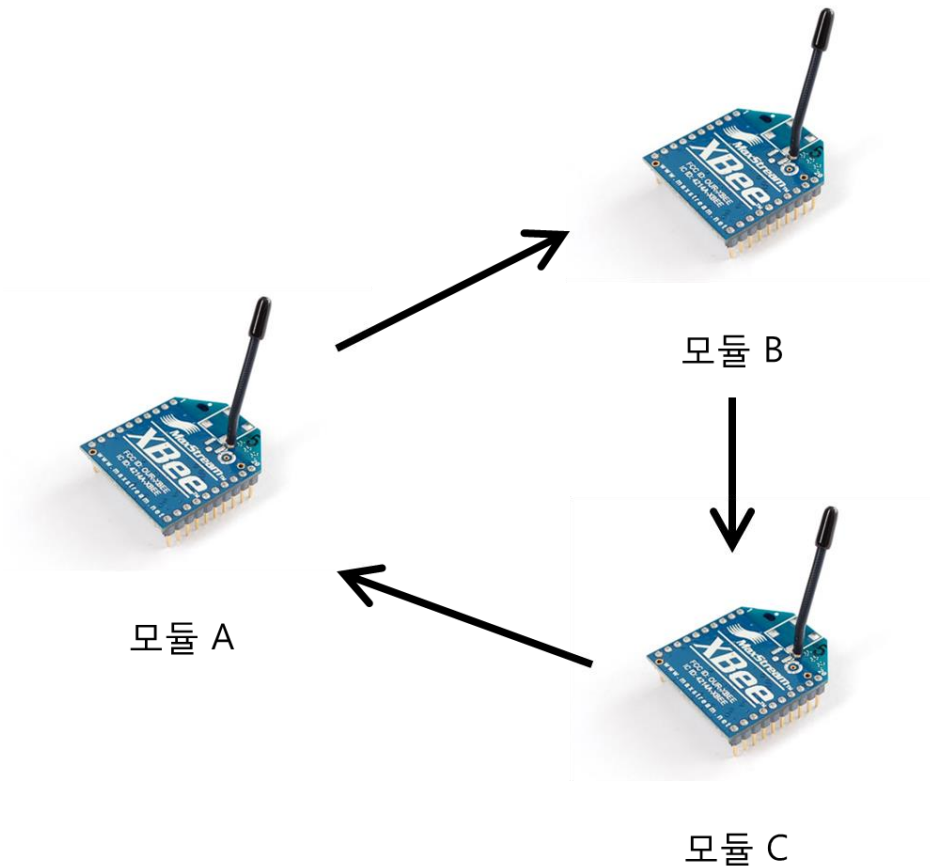
모듈 A와 모듈 B가 통신을 위해서는 다음의 주소를 설정해야 합니다.

예를 들어,

모듈 A	모듈 B
CH: 0x0C	CH는 모듈 A의 CH와 같아야 합니다. (0x0C)
ID: 3332	ID는 모듈 A의 ID와 같아야 합니다. (3332)
DL: 5678	DL은 모듈 A의 MY와 같아야 합니다. (1234)
MY: 1234	MY는 모듈 A의 DL과 같아야 합니다. (5678)

### 원형 통신 주소 설정





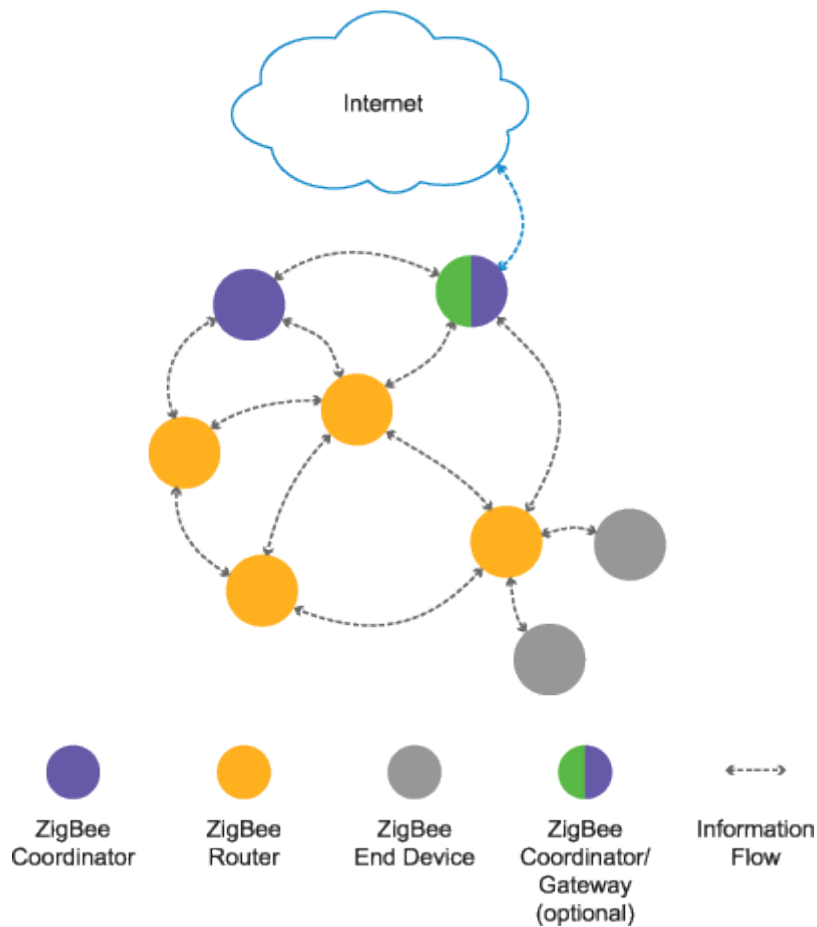
모듈 A에서 모듈 B로, 그리고 모듈 B에서 모듈 C로, 다시 모듈 C에서 모듈 A로 통신하기 위해서는 다음과 같이 주소를 설정할 수 있습니다.

	모듈 A	모듈 B	모듈 C
<b>CH</b>	0x0C	0x0C	0x0C
<b>ID</b>	3332	3332	3332
<b>DL</b>	3	1	2
<b>MY</b>	1	2	3

#### B. XBee 시리즈 2 파라미터 설정

XBee 시리즈 2는 시리즈 1과 달리 ZigBee Mesh 프로토콜을 사용합니다. 이 ZigBee Mesh 프로토콜은 다음의 세가지 디바이스 타입을 사용하는데, 각각 Coordinator, Router, 그리고 End Device입니다. 각각의 하는 역할을 설명하기 위해서 다음의 그림을 참고해주세요.

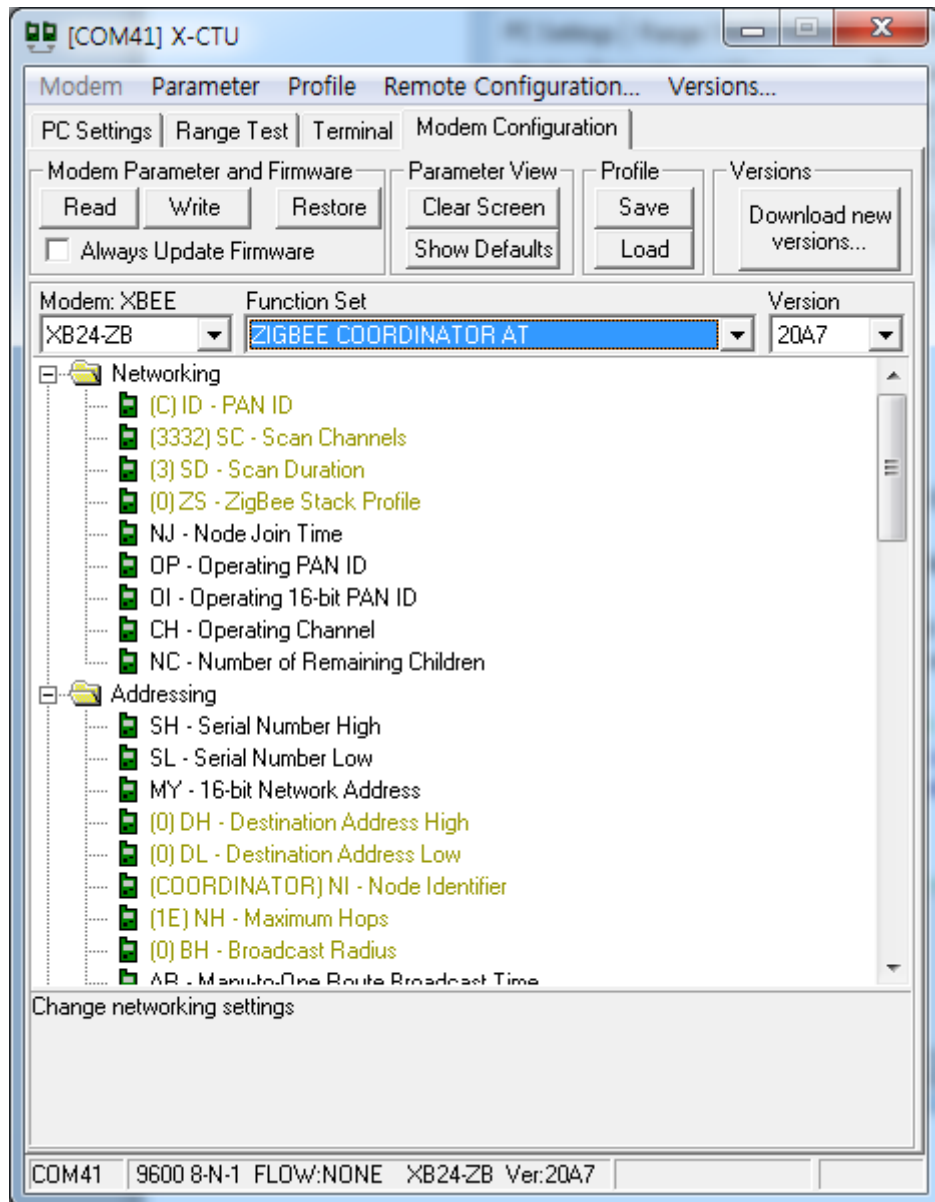
## ZigBee Network Topology



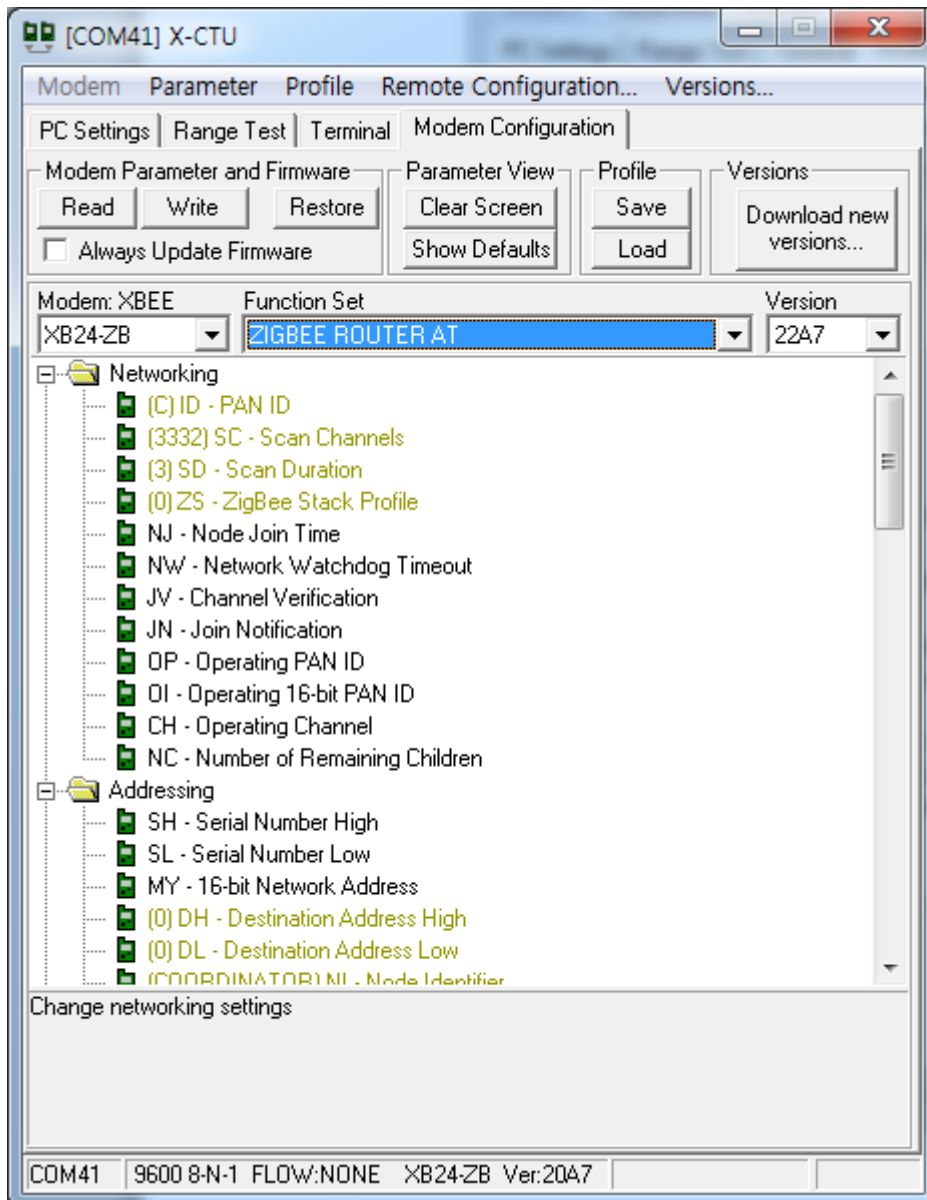
Coordinator는 네트워크의 형태를 결정하거나 보안을 담당합니다. 그리고, Router는 네트워크의 범위를 확장할 수 있고, End Device는 개별적인 센싱 혹은 제어를 담당합니다. 이해를 돕기 위해서 스마트홈을 생각해 보면, Coordinator는 홈씨어터로서 음향 및 영상을 제어하고, Router는 음향 기기 및 조명/영상 기계들입니다. 그리고 End Device는 각각의 센서 및 스위치들이라고 생각할 수 있습니다.

그럼, 본격적으로 XBee Series 2 설정을 통해서 양방향 통신 방법에 대해서 알아보도록 하겠습니다.

먼저, 모듈 A를 Coordinator로 설정하겠습니다.



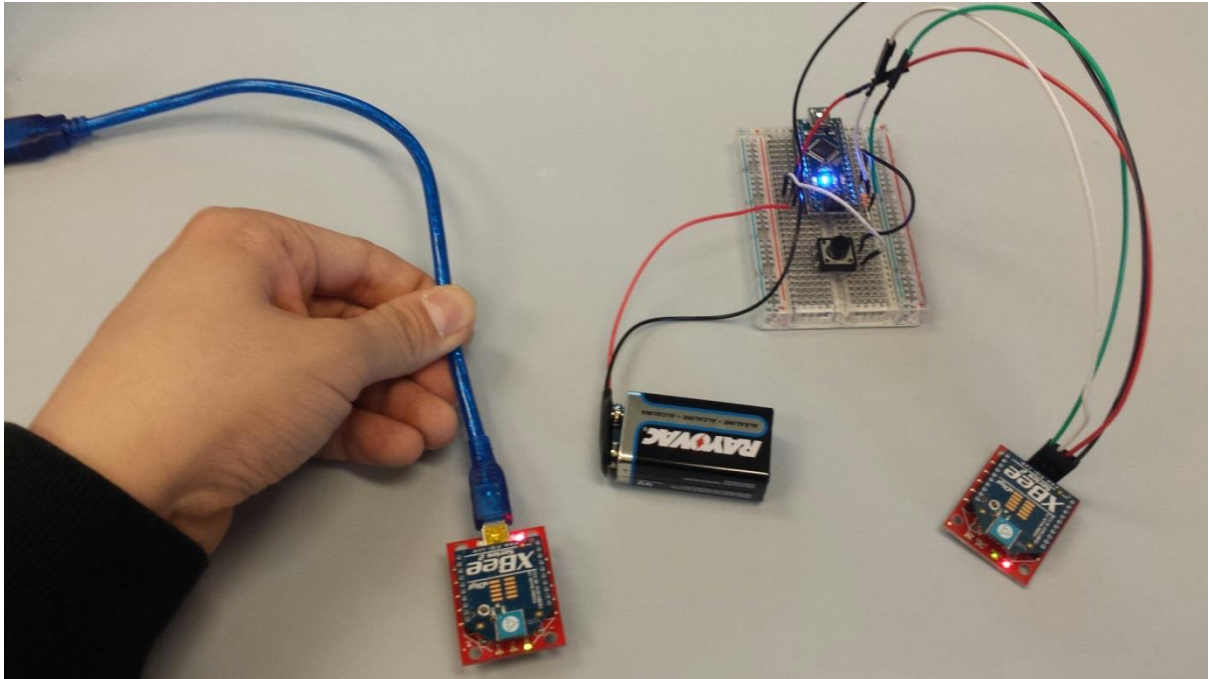
다음, 모듈 B를 Router로 설정하겠습니다.



Router가 있는 쪽에서 Coordinator쪽으로 값을 전송하는 프로그램을 구성해보겠습니다.

Router쪽에 아두이노와 스위치를 연결하고, Coordinator쪽에 컴퓨터를 연결하여 컴퓨터의 하이퍼터미널 혹은 아두이노 소프트웨어의 시리얼 모니터링을 통해 값을 출력합니다.

Router쪽의 회로 및 프로그램은 시리즈 1을 사용하였던 이전 챕터의 3-II의 송신부와 같습니다.

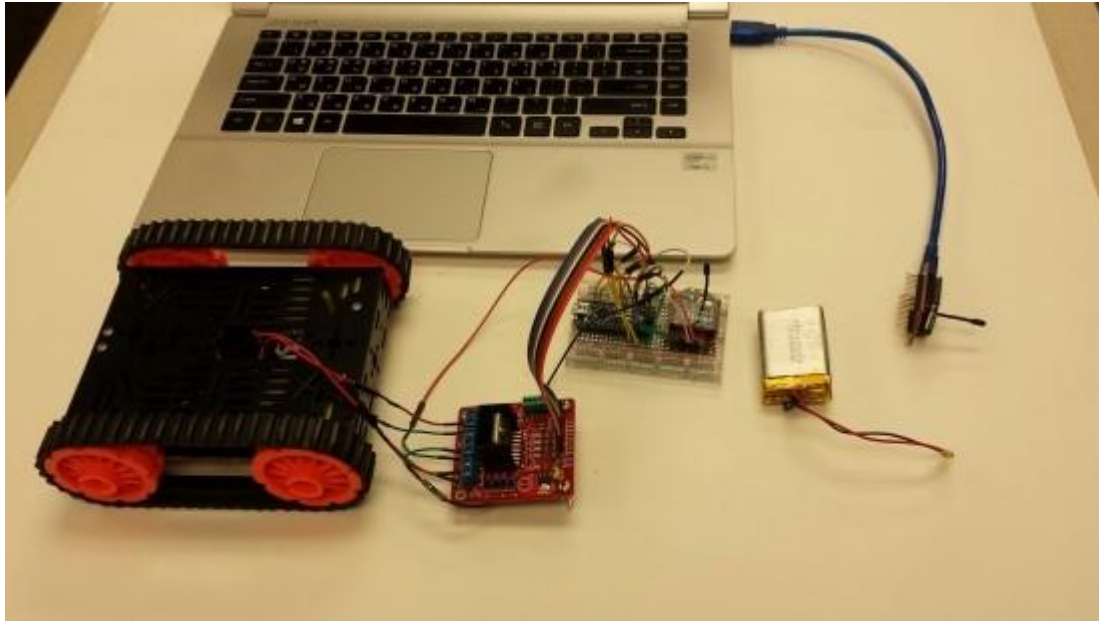


## 5. 프로젝트 1: 키보드 입력으로 RC 탱크 구축하기

아두이노와 XBee 를 통해서 RC 탱크를 제어하는 데모를 구현해봤습니다.

키보드에서 'A' (왼쪽), 'D' (오른쪽), 'W' (전진), 'S' (정지), 'X' (후진)으로 컨트롤을 하며, 필요한 부품으로는

- 1) 아두이노 보드 ([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=1&category=001001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=1&category=001001))
- 2) 모터 드라이버  
([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=206&category=007001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=206&category=007001))
- 3) XBEE 시리즈 1 - 2 개  
([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=377&category=006002](http://mechasolution.com/shop/goods/goods_view.php?goodsno=377&category=006002))
- 4) 배터리 3 개 ([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=893&category=025001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=893&category=025001))
- 5) RC 탱크 프레임  
([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=1205&category=013](http://mechasolution.com/shop/goods/goods_view.php?goodsno=1205&category=013))
- 6) XBee USB 브레이크 아웃보드  
([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=15&category=006002](http://mechasolution.com/shop/goods/goods_view.php?goodsno=15&category=006002))
- 7) XBee 레귤레이터 보드  
([http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=16&category=006002](http://mechasolution.com/shop/goods/goods_view.php?goodsno=16&category=006002))



먼저, Youtube 에 올린 동영상입니다.

<http://youtu.be/LwvzimztsQI>

RC 탱크 프레임의 설명서에 맞게 조립한 후에 내부에 배터리를 넣었습니다. 3.7V 배터리 3 개를 사용하였으며 (직렬), 양면 테이프로 차체에 부착하였습니다.



배터리를 넣은 후에, 그 위에 브레드보드를 이용하여 아두이노 나노와 XBee 를 탑재하였습니다. XBee 는 레귤레이터보드를 사용하여 브레드보드에 끼웠습니다. (XBee 의 핀이 간격이 일반 핀헤더보다 작기 때문에 브레드보드에 바로 삽입이 되지 않습니다.)

\* XBee 와 나노의 연결은 다음과 같고, XBee 와 연결하기 전에 아두이노 코드를 업로드 해줍니다. (XBee 가 연결되어 있는 상태에서는 아두이노 코드 업로드가 되지 않습니다. 따라서 업로드시에는 XBee 를 제거한 뒤에 업로드하고 업로드가 완료되면 XBee 를 다시 연결합니다.)

#### **XBee ----- 아두이노 나노**

GND ----- GND

5V ----- 5V

DOUT ----- RX

DIN ----- TX

#### **아두이노 코드**

```
int dir1PinA = 2;
int dir2PinA = 3;
int dir1PinB = 4;
int dir2PinB = 5;

int speedPinA = 9;
int speedPinB = 10;
byte incomingByte;
void setup()
{
  Serial.begin(9600);
  pinMode(dir1PinA, OUTPUT);
  pinMode(dir2PinA, OUTPUT);
  pinMode(dir1PinB, OUTPUT);
  pinMode(dir2PinB, OUTPUT);
  pinMode(speedPinA, OUTPUT);
  pinMode(speedPinB, OUTPUT);
  digitalWrite(dir1PinA, LOW);
  digitalWrite(dir2PinA, LOW);
  digitalWrite(dir1PinB, LOW);
  digitalWrite(dir2PinB, LOW);
  analogWrite(speedPinA, 200);
  analogWrite(speedPinB, 200);

}

void loop()
{
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    if (incomingByte == 'W')
    {
      digitalWrite(dir1PinA, HIGH);
      digitalWrite(dir2PinA, LOW);
      digitalWrite(dir1PinB, HIGH);
      digitalWrite(dir2PinB, LOW);
    }
    if (incomingByte == 'D')
    {
      digitalWrite(dir1PinA, LOW);
      digitalWrite(dir2PinA, LOW);
    }
  }
}
```

```

digitalWrite(dir1PinB, HIGH);
digitalWrite(dir2PinB, LOW);
}
if (incomingByte == 'A')
{
digitalWrite(dir1PinA, HIGH);
digitalWrite(dir2PinA, LOW);
digitalWrite(dir1PinB, LOW);
digitalWrite(dir2PinB, LOW);
}
if (incomingByte == 'S')
{
digitalWrite(dir1PinA, LOW);
digitalWrite(dir2PinA, LOW);
digitalWrite(dir1PinB, LOW);
digitalWrite(dir2PinB, LOW);
}
if (incomingByte == 'X')
{
    digitalWrite(dir1PinA, LOW);
    digitalWrite(dir2PinA, HIGH);
    digitalWrite(dir1PinB, LOW);
    digitalWrite(dir2PinB, HIGH);
}
}
}

```

아두이노 코드를 업로드 한 후에 XBee 를 연결하고, 그 다음에는 모터 드라이버와 아두이노의 연결 및 모터 드라이버와 RC 탱크의 두개의 모터를 연결합니다.

#### 아두이노 나노 ----- 모터 드라이버

D9 ----- ENA

D2 ----- IN1

D3 ----- IN2

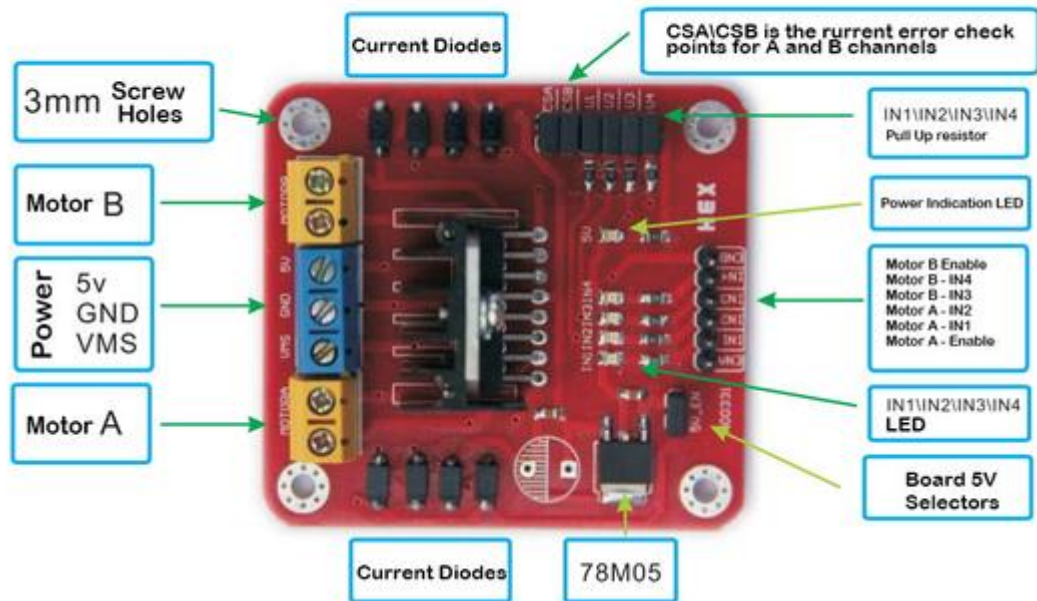
D4 ----- IN3

D5 ----- IN4

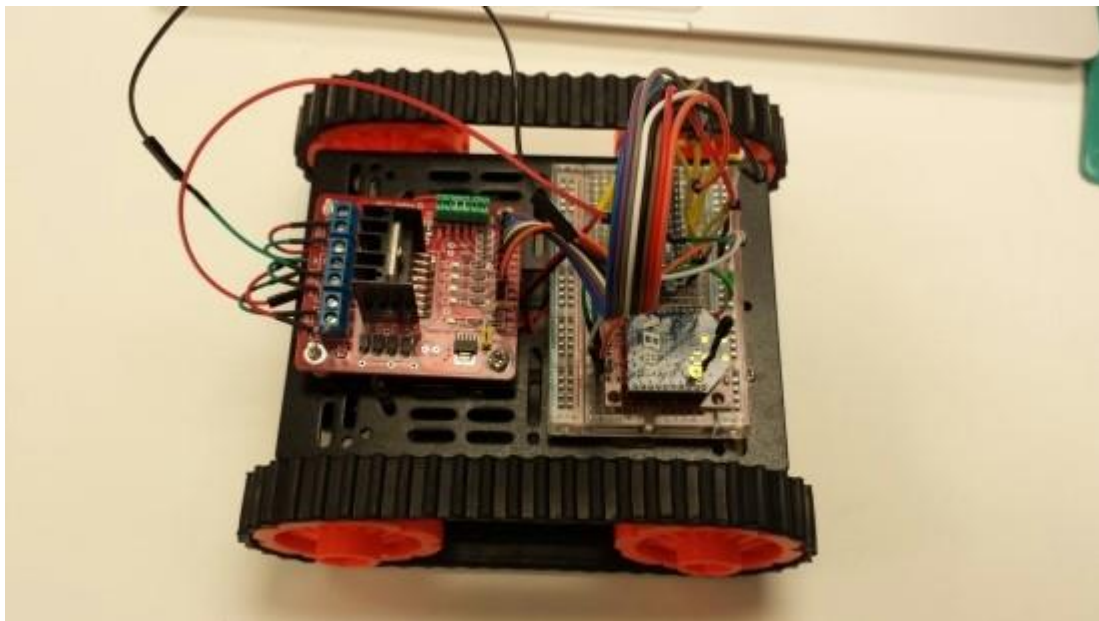
D10 ----- ENB

모터로 들어가는 두개의 선은 왼쪽의 Motor A 혹은 Motor B 출력 단자를 사용하면 됩니다.





배터리의 전원은 아두이노의 VIN과 GND에 그리고 모터 드라이버의 GND와 VMS에 연결합니다. 즉, VIN과 VMS는 연결되어 있고, 아두이노의 GND와 모터 드라이버의 GND도 연결되어 있습니다.



RC 탱크부를 연결한 후에는 XBee 발신부를 컴퓨터의 USB에 연결한 후에 아두이노의 시리얼 모니터링을 통해서 문자를 보내봅니다.

## 6. 프로젝트 2: RSSI를 이용한 XBee간 거리 센싱하기 (AT-API)

RSSI는 Received Signal Strength Indicator의 약자로서 신호의 세기를 의미합니다. 이 신호의 세기는 거리가 멀어지면 세기가 약해지고, 이를 통해서 거리를 측정하는 것이 가능합니다. 물론, RSSI 값은 주변 환경의 노이즈에 취약하고, 안테나의 방사 패턴이 일정하지 않다는 단점이 있지만, 초음파센서 혹은 적외선센서와 달리 어느 방향에서도 거리 측정이 가능하다는 장점이 있습니다.

필요한 준비물로는

XBee Series 1 안테나형 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=377&category=006002](http://mechasolution.com/shop/goods/goods_view.php?goodsno=377&category=006002)

XBee 레귤레이터 보드 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?&goodsno=16](http://mechasolution.com/shop/goods/goods_view.php?&goodsno=16)

아두이노 나노 호환보드 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=1&category=001001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=1&category=001001)

브레드보드 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=7&category=027001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=7&category=027001)

점퍼케이블

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=673&category=024001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=673&category=024001)

USB 미니 케이블 2개

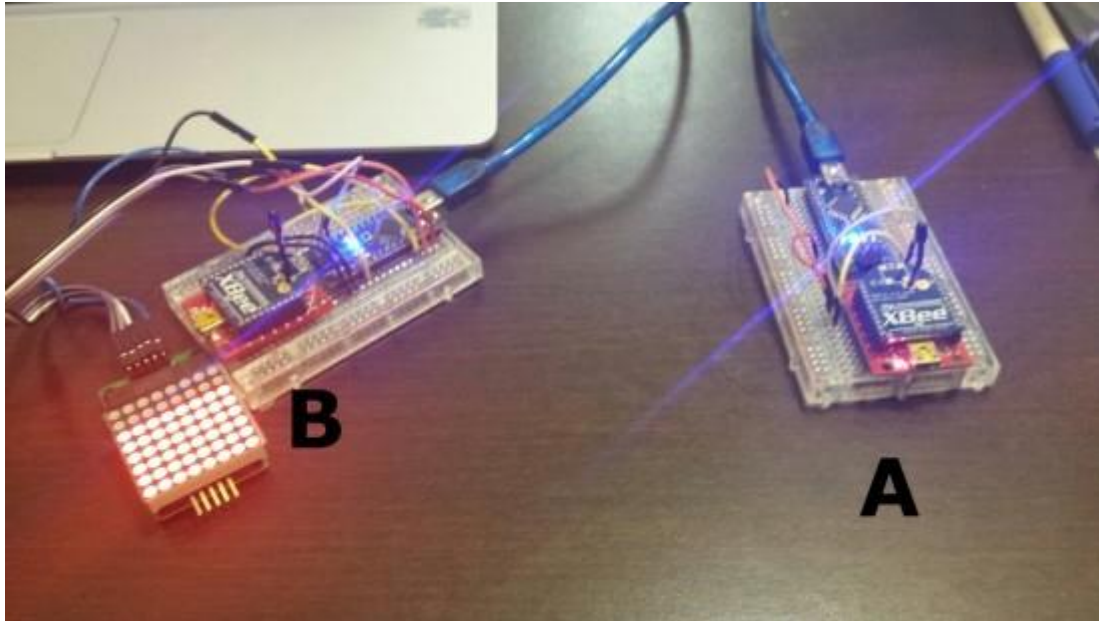
[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=8&category=024001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=8&category=024001)

MAX7219 도트 매트릭스 키트

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=120&category=009001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=120&category=009001)

### XBee 세팅

다음의 그림에서 오른쪽을 'A', 왼쪽을 'B'라 편의상 칭하겠습니다.



먼저, X-CTU 프로그램을 이용하여 XBee를 다음과 같이 설정합니다.

A 모듈

XB24 - XBEE 802.15.4

DL - 0

MY - 1

BD - 57600

AP - 0

B모듈

XB24 - XBEE 802.15.4

MY - 0

BD - 57600

AP - 1

B 모듈에는 MAX7219 도트매트릭스 키트를 통해서 거리를 대략적으로 디스플레이하였으며, 아두이노와의 연결법은 다음과 같습니다.

**아두이노 나노 ----- 도트매트릭스 키트**

5V ----- VCC

GND ----- GND

D8 ----- DIN

D9 ----- CS

D10 ----- CLK

#### 아두이노 소스코드 (A모듈)

```
void setup(){
    Serial.begin(57600);
}

void loop(){
    Serial.print("ID:001");
    delay(300);
}
```

#### 아두이노 소스코드 (B모듈)

```
unsigned char i;
unsigned char j;
/*Port Definitions*/
int Max7219_pinCLK = 10;
int Max7219_pinCS = 9;
int Max7219_pinDIN = 8;

int vol;

unsigned char disp1[9][8]={
{0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0},//0
{0x1,0x1,0x1,0x1,0x1,0x1,0x1,0x1},//1
{0x3,0x3,0x3,0x3,0x3,0x3,0x3,0x3},//2
{0x7,0x7,0x7,0x7,0x7,0x7,0x7,0x7},//3
{0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF},//4
{0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F},//5
{0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F},//6
{0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F},//7
{0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF},//8
};

void Write_Max7219_byte(unsigned char DATA)
{
    unsigned char i;
```

```

digitalWrite(Max7219_pinCS,LOW);
for(i=8;i>=1;i--)
{
    digitalWrite(Max7219_pinCLK,LOW);
    digitalWrite(Max7219_pinDIN,DATA&0x80);// Extracting a bit data
    DATA = DATA<<1;
    digitalWrite(Max7219_pinCLK,HIGH);
}
}

void Write_Max7219(unsigned char address,unsigned char dat)
{
    digitalWrite(Max7219_pinCS,LOW);
    Write_Max7219_byte(address);          //address , code of LED
    Write_Max7219_byte(dat);              //data , figure on LED
    digitalWrite(Max7219_pinCS,HIGH);
}

void Init_MAX7219(void)
{
    Write_Max7219(0x09, 0x00);           //decoding : BCD
    Write_Max7219(0x0a, 0x03);           //brightness
    Write_Max7219(0x0b, 0x07);           //scanlimit ; 8 LEDs
    Write_Max7219(0x0c, 0x01);           //power-down mode : 0 , normal mode : 1
    Write_Max7219(0x0f, 0x00);           //test display : 1 ; EOT , display : 0
}

void setup(){
    Serial.begin(57600);
    pinMode(Max7219_pinCLK,OUTPUT);
    pinMode(Max7219_pinCS,OUTPUT);
    pinMode(Max7219_pinDIN,OUTPUT);
    delay(50);
    for(i=1;i<9;i++)
    {
        Write_Max7219(i,disp1[0][i-1]);
    }
}

```

```

    Init_MAX7219();
}

byte rssi=0x00;
void loop(){
    if(Serial.available()){
        byte in = Serial.read();
        if(in == (byte) 0x7E){
            int i=0;
            while(true){
                if(Serial.available()){
                    in= Serial.read();
                    if(i==5)rssi=in;
                    Serial.print(in,HEX);
                    Serial.print(" ");
                    i++;
                }
                if(i>13)break;
            }
            Serial.print("/ ");
            Serial.print("RSSI:");
            Serial.print(rssi,HEX);
            Serial.println(" ");
        }
    }
    if(rssi>0x30){
        for(i=1;i<9;i++)
        {
            Write_Max7219(i,disp1[0][i-1]);
        }
    }
    else if(rssi>0x2B){
        for(i=1;i<9;i++)
        {
            Write_Max7219(i,disp1[2][i-1]);
        }
    }
    else if(rssi>0x27){
        for(i=1;i<9;i++)

```

```

    {
        Write_Max7219(i,disp1[4][i-1]);
    }
}
else if(rssi>0x22){
for(i=1;i<9;i++)
    {
        Write_Max7219(i,disp1[6][i-1]);
    }
}
else if(rssi>0x1C){
for(i=1;i<9;i++)
    {
        Write_Max7219(i,disp1[8][i-1]);
    }
}
else{

}
}
}

```

유튜브 동영상 링크: [http://youtu.be/2q\\_9i6XssUg](http://youtu.be/2q_9i6XssUg)

## 7. 프로젝트 3: RSSI를 이용한 XBee간 거리 센싱하기 (API-API)

AT-API 통신을 통한 RSSI는 간편하고 두 XBee간의 거리를 측정하기에는 문제가 없지만, 다수의 RSSI를 제어하기 어렵다는 점과 양방향으로 값을 얻기 어렵다는 단점이 있습니다. 이번 프로젝트는 API-API 통신을 통해서 XBee간의 거리를 측정하는 프로젝트를 구현해 보겠습니다.

필요한 준비물로는

XBee Series 1 안테나형 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=377&category=006002](http://mechasolution.com/shop/goods/goods_view.php?goodsno=377&category=006002)

XBee 레귤레이터 보드 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?&goodsno=16](http://mechasolution.com/shop/goods/goods_view.php?&goodsno=16)

아두이노 나노 호환보드 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=1&category=001001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=1&category=001001)

브레드보드 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=7&category=027001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=7&category=027001)

점퍼케이블

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=673&category=024001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=673&category=024001)

USB 미니 케이블 2개

[http://mechasolution.com/shop/goods/goods\\_view.php?goodsno=8&category=024001](http://mechasolution.com/shop/goods/goods_view.php?goodsno=8&category=024001)

## XBee 세팅

X-CTU를 통해 XBee는 9600으로 baudrate를 맞춘 후에 AP=2로 설정합니다.

그리고, XBee의 주소 값을 찾아서 적어둡니다. X-CTU에 보면

0x0013a200, 0x40b5de73와 같은 주소가 있는데, 이것은 XBee에서 API 통신을 할 때, 상대방

주소로 사용되게 됩니다.

XBee 라이브러리는 다음의 링크에서 다운로드 받을 수 있습니다.

<http://mechasolution.com/shop/board/view.php?id=mechatutorial&no=12>

## 아두이노 코드 (RX)

```
#include <XBee.h>

XBee xbee = XBee();
XBeeResponse response = XBeeResponse();
// create reusable response objects for responses we expect to handle
Rx16Response rx16 = Rx16Response();
Rx64Response rx64 = Rx64Response();

int statusLed = 11;
int errorLed = 12;
int dataLed = 10;
int rssiLed = 9;

uint8_t option = 0;
uint8_t data = 0;
uint8_t rssi = 0;
//getRssi();

void flashLed(int pin, int times, int wait) {
    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(wait);
        digitalWrite(pin, LOW);
    }
}
```



```

        if (i + 1 < times) {
            delay(wait);
        }
    }
}

void setup() {
    pinMode(statusLed, OUTPUT);
    pinMode(errorLed, OUTPUT);
    pinMode(dataLed, OUTPUT);
    pinMode(rssiLed, OUTPUT);
    // start serial
    Serial.begin(9600);
    xbee.setSerial(Serial);

    flashLed(statusLed, 3, 50);
}

// continuously reads packets, looking for RX16 or RX64
void loop() {

    xbee.readPacket();

    if (xbee.getResponse().isAvailable()) {
        // got something

        if (xbee.getResponse().getApId() == RX_16_RESPONSE || xbee.getResponse().getApId() ==
RX_64_RESPONSE) {
            // got a rx packet

            if (xbee.getResponse().getApId() == RX_16_RESPONSE) {
                xbee.getResponse().getRx16Response(rx16);
                option = rx16.getOption();
                data = rx16.getData(0);
                rssi = rx16.getRssi();
            } else {
                xbee.getResponse().getRx64Response(rx64);
                option = rx64.getOption();
                data = rx64.getData(0);
                rssi = rx16.getRssi();
            }

            // TODO check option, rssi bytes
            flashLed(statusLed, 1, 10);

            // set dataLed PWM to value of the first byte in the data
            analogWrite(dataLed, data);
            analogWrite(rssiLed, rssi);
            Serial.println(rssi);
        } else {
            // not something we were expecting
            flashLed(errorLed, 1, 25);
        }
    } else if (xbee.getResponse().isError()) {
        //nss.print("Error reading packet. Error code: ");
        //nss.println(xbee.getResponse().getErrorCode());
        // or flash error led
    }
}

```

```
}
```

## 아두이노 코드 (TX)

```
#include <XBee.h>

/*
This example is for Series 1 XBee
Sends a TX16 or TX64 request with the value of analogRead(pin5) and checks the status response
for success
Note: In my testing it took about 15 seconds for the XBee to start reporting success, so I've added a
startup delay
*/

XBee xbee = XBee();

unsigned long start = millis();

// allocate two bytes for to hold a 10-bit analog reading
uint8_t payload[] = { 0, 0 };

// with Series 1 you can use either 16-bit or 64-bit addressing

// 16-bit addressing: Enter address of remote XBee, typically the coordinator
//Tx16Request tx = Tx16Request(0x1874, payload, sizeof(payload));

// 64-bit addressing: This is the SH + SL address of remote XBee
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x40b5de73);
// unless you have MY on the receiving radio set to FFFF, this will be received as a RX16 packet
Tx64Request tx = Tx64Request(addr64, payload, sizeof(payload));

TxStatusResponse txStatus = TxStatusResponse();

int pin5 = 0;

int statusLed = 11;
int errorLed = 12;

void flashLed(int pin, int times, int wait) {

    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(wait);
        digitalWrite(pin, LOW);

        if (i + 1 < times) {
            delay(wait);
        }
    }
}

void setup() {
    pinMode(statusLed, OUTPUT);
    pinMode(errorLed, OUTPUT);
    Serial.begin(9600);
    xbee.setSerial(Serial);
}
```

```

void loop() {

    // start transmitting after a startup delay.  Note: this will rollover to 0 eventually so not best way to
    handle
    if (millis() - start > 1000) {
        // break down 10-bit reading into two bytes and place in payload
        pin5 = analogRead(5);
        pin5 = 2000;
        payload[0] = pin5 >> 8 & 0xff;
        payload[1] = pin5 & 0xff;

        xbee.send(tx);

        // flash TX indicator
        flashLed(statusLed, 1, 100);
    }

    // after sending a tx request, we expect a status response
    // wait up to 5 seconds for the status response
    if (xbee.readPacket(5000)) {
        // got a response!

        // should be a znet tx status
        if (xbee.getResponse().getApiId() == TX_STATUS_RESPONSE) {
            xbee.getResponse().getZBTxStatusResponse(txStatus);

            // get the delivery status, the fifth byte
            if (txStatus.getStatus() == SUCCESS) {
                // success.  time to celebrate
                flashLed(statusLed, 5, 50);
            } else {
                // the remote XBee did not receive our packet.  is it powered on?
                flashLed(errorLed, 3, 500);
            }
        }
    }
    } else if (xbee.getResponse().isError()) {
        //nss.print("Error reading packet.  Error code: ");
        //nss.println(xbee.getResponse().getErrorCode());
        // or flash error led
    } else {
        // local XBee did not provide a timely TX Status Response.  Radio is not configured properly or
        connected
        flashLed(errorLed, 2, 50);
    }

    delay(1000);
}

```