



COMMUNICATION PROTOCOLS

What are Protocols?



Google says :"A communication protocol is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of physical quantity."

Why do we need them?

- Since a lot of devices communicating with each other.
- Avoiding Interference with other communicating modules.
- Efficient and complete data transfer.



Numerous Communication Protocols Are Present.

UART(Universal Asynchronous Receiving and Transmission)

- GPS,Bluetooth Module(UART Interface),etc.

SPI(Serial Peripheral Index)

- Compass sensor,NRF etc.

I2C(Inter Integrated Circuit)

- BMP280, Certain Temperature sensors etc.

ZIGBEE

- Xbee, Amazon Echo Plus etc.

802.11(Wifi)

- ESP family(ESP 8266,ESP 32)

HTTPS,HTTP(Hyper Text Transfer Protocol)

- Internet uses this to transfer data across the world.

A blue printed circuit board (PCB) featuring a central GPS module with a QR code and the text 'NEO-M6' and 'u-blox'. The board is populated with various electronic components, including a yellow SMA connector, a small black integrated circuit, and several surface-mount components. The text 'GPS6MV2' is visible on the top left of the board. At the bottom, there are four gold-plated pins labeled 'VCC', 'RX', 'TX', and 'GND'.

UART(Universal Asynchronous Receiving and Transmitter or Transmission)

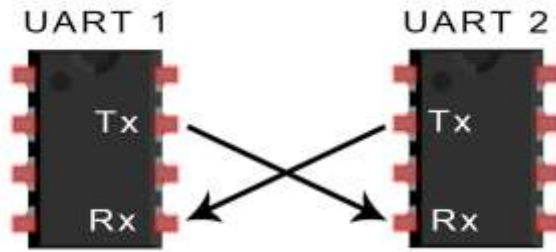


What is a clock ?

- The clock rate typically refers to the frequency at which the clock generator of a processor can generate pulses, which are used to synchronize the operations of its components.
- It is used as an indicator of the processor's speed.

- A universal asynchronous receiver-transmitter is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable.
- Converts the bytes it receives from the computer along parallel circuits into a single serial bit stream for outbound transmission.

! Which is just a device!



Which is the case in a computer.

What would be the case on a MC?

It's the same for MC as well!(But with low voltage levels and low baud rates)

The way the communication is taking place is the protocol.

Let us take an example of GPS Module

- GPS works by using a method called "triangulation" or "trilateration".
- It needs to get a message from at least three, preferably four satellites
- To "triangulate", a GPS receiver measures the distance between itself and each satellite. It can measure distance because it works out exactly how long it took for each satellite's message to arrive. (distance = time of arrival * speed of light)
- To measure travel time, GPS needs very accurate timing which it achieves with atomic clocks on board each satellite.
- Along with distance, the device needs to know exactly where the satellites are in space at any given time. This information is held inside the GPS receiver itself.
- Finally, because it knows exactly where the satellites are at that instant, by using some very clever mathematics, it can work out where it is on the ground.

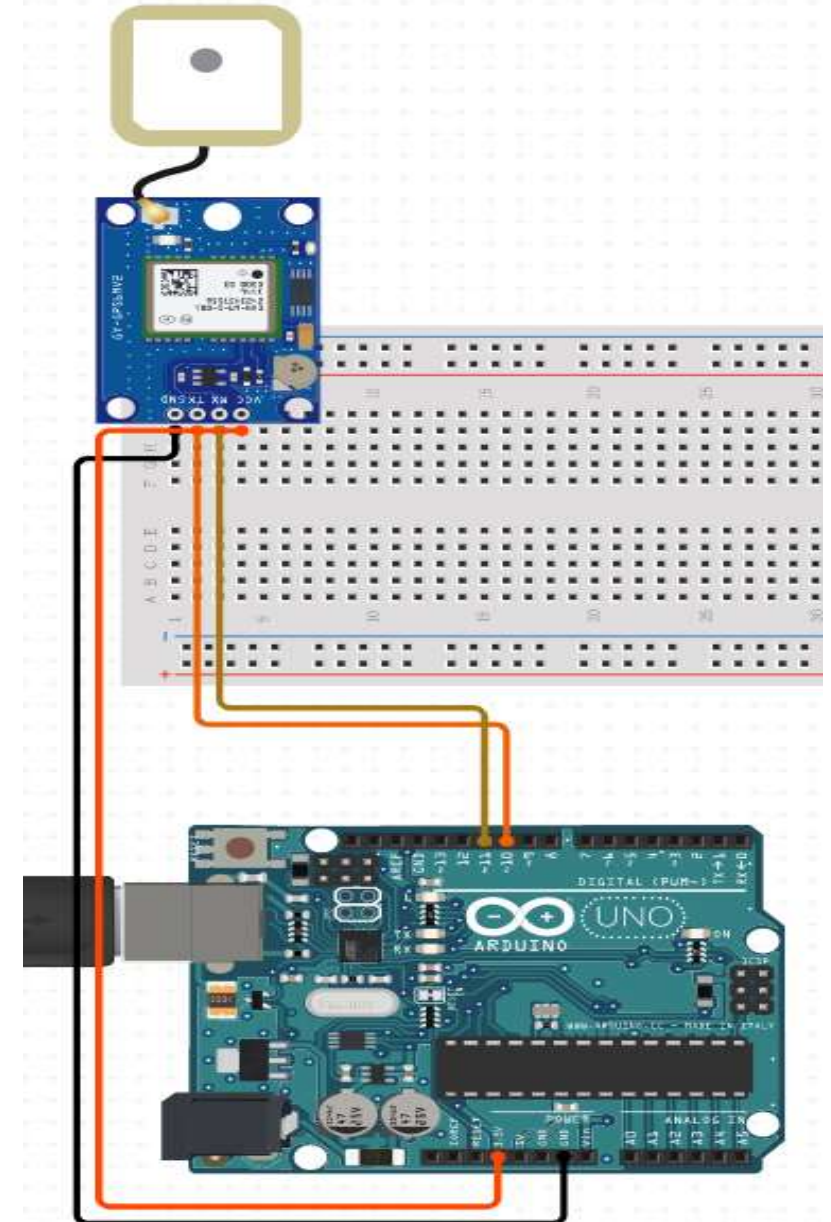
Pin config:

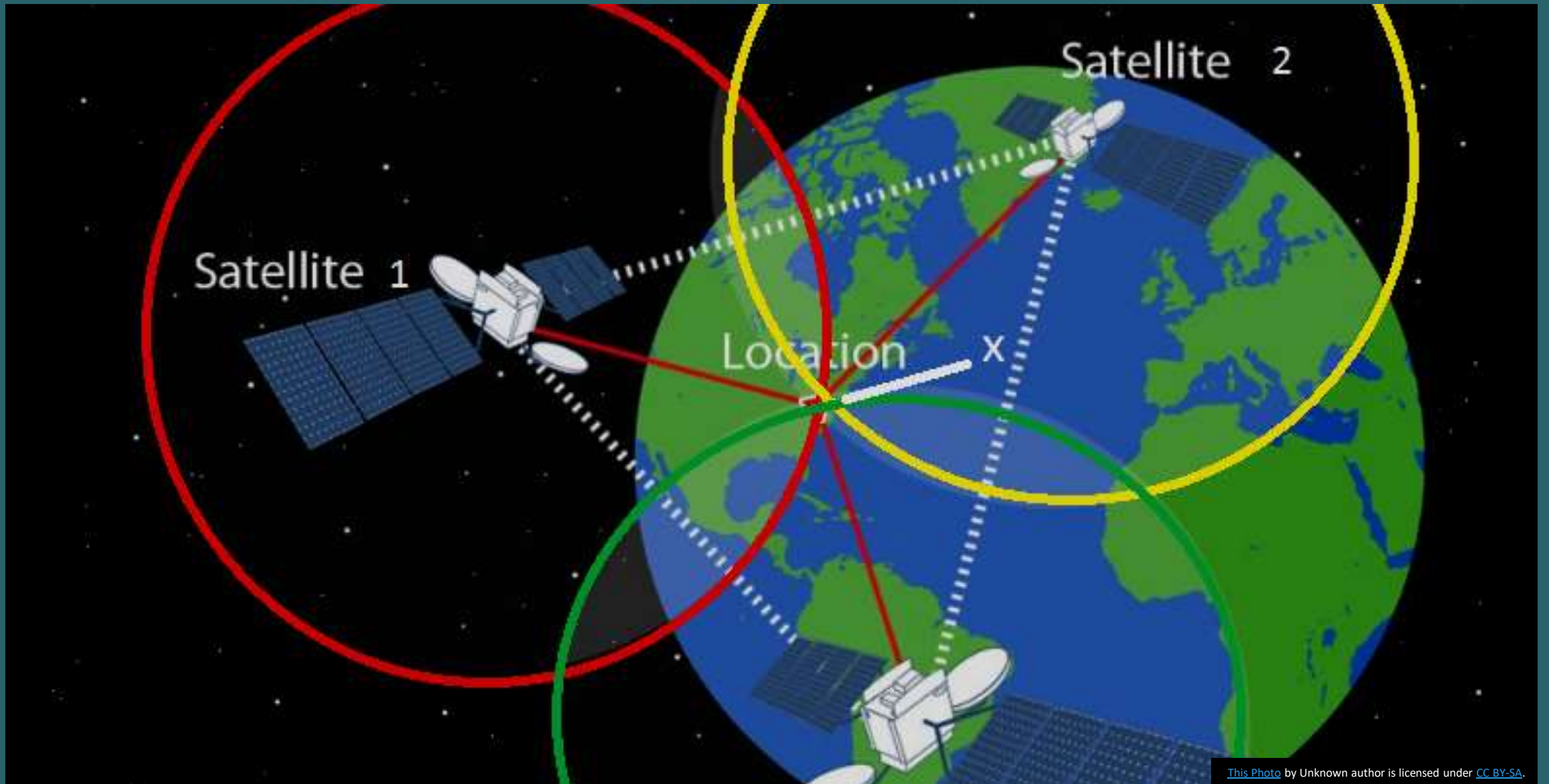
VCC- 5V

Gnd – 0V

Tx – UART transmit

Rx-UART receiver

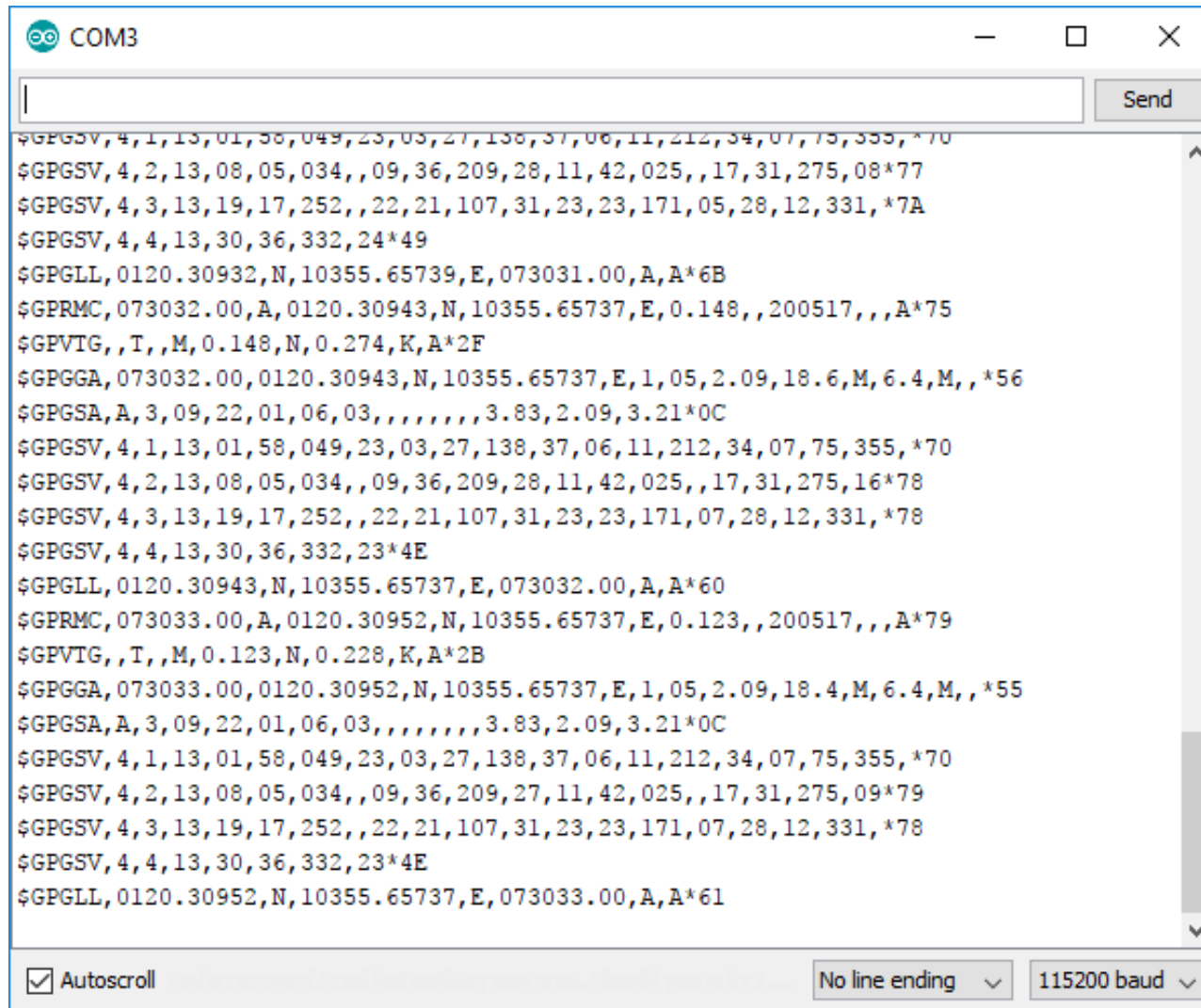




[This Photo](#) by Unknown author is licensed under [CC BY-SA](#).

Sample code for GPS

```
#include <SoftwareSerial.h>
static const int RXPin = 10, TXPin = 11;
static const GPSPBaud = 9600;
// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);
void setup()
{
    Serial.begin(9600);
    ss.begin(GPSPBaud);
}
void loop()
{
    // Output raw GPS data to the serial monitor
    while (ss.available() > 0){
        Serial.write(ss.read());
    }
}
```




```
$GPGSV,4,1,13,01,58,049,23,03,27,138,37,06,11,212,34,07,75,355,*70
$GPGSV,4,2,13,08,05,034,,09,36,209,28,11,42,025,,17,31,275,08*77
$GPGSV,4,3,13,19,17,252,,22,21,107,31,23,23,171,05,28,12,331,*7A
$GPGSV,4,4,13,30,36,332,24*49
$GPGLL,0120.30932,N,10355.65739,E,073031.00,A,A*6B
$GPRMC,073032.00,A,0120.30943,N,10355.65737,E,0.148,,200517,,A*75
$GPVTG,,T,,M,0.148,N,0.274,K,A*2F
$GPGGA,073032.00,0120.30943,N,10355.65737,E,1,05,2.09,18.6,M,6.4,M,,*56
$GPGSA,A,3,09,22,01,06,03,,,,,,,,,3.83,2.09,3.21*0C
$GPGSV,4,1,13,01,58,049,23,03,27,138,37,06,11,212,34,07,75,355,*70
$GPGSV,4,2,13,08,05,034,,09,36,209,28,11,42,025,,17,31,275,16*78
$GPGSV,4,3,13,19,17,252,,22,21,107,31,23,23,171,07,28,12,331,*78
$GPGSV,4,4,13,30,36,332,23*4E
$GPGLL,0120.30943,N,10355.65737,E,073032.00,A,A*60
$GPRMC,073033.00,A,0120.30952,N,10355.65737,E,0.123,,200517,,A*79
$GPVTG,,T,,M,0.123,N,0.228,K,A*2B
$GPGGA,073033.00,0120.30952,N,10355.65737,E,1,05,2.09,18.4,M,6.4,M,,*55
$GPGSA,A,3,09,22,01,06,03,,,,,,,,,3.83,2.09,3.21*0C
$GPGSV,4,1,13,01,58,049,23,03,27,138,37,06,11,212,34,07,75,355,*70
$GPGSV,4,2,13,08,05,034,,09,36,209,27,11,42,025,,17,31,275,09*79
$GPGSV,4,3,13,19,17,252,,22,21,107,31,23,23,171,07,28,12,331,*78
$GPGSV,4,4,13,30,36,332,23*4E
$GPGLL,0120.30952,N,10355.65737,E,073033.00,A,A*61
```

☒ Autoscroll No line ending 115200 baud

Could not understand it Right!(Mixed up values of Altitude, LONG, LAT ,Time etc.)

Use TinyGPS++(A Library) to understand what it means .

If you use Tiny gps you can get Output similar to this, for the examples present on library examples.



COM3

Send

Location: 1.338830,103.927528 Date/Time: 5/20/2017 07:20:25.00
Location: 1.338830,103.927528 Date/Time: 5/20/2017 07:20:25.00
Location: 1.338830,103.927528 Date/Time: 5/20/2017 07:20:25.00
Location: 1.338830,103.927528 Date/Time: 5/20/2017 07:20:25.00
Location: 1.338830,103.927528 Date/Time: 5/20/2017 07:20:25.00
Location: 1.338830,103.927528 Date/Time: 5/20/2017 07:20:25.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338811,103.927536 Date/Time: 5/20/2017 07:20:26.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00
Location: 1.338773,103.927536 Date/Time: 5/20/2017 07:20:27.00

< >

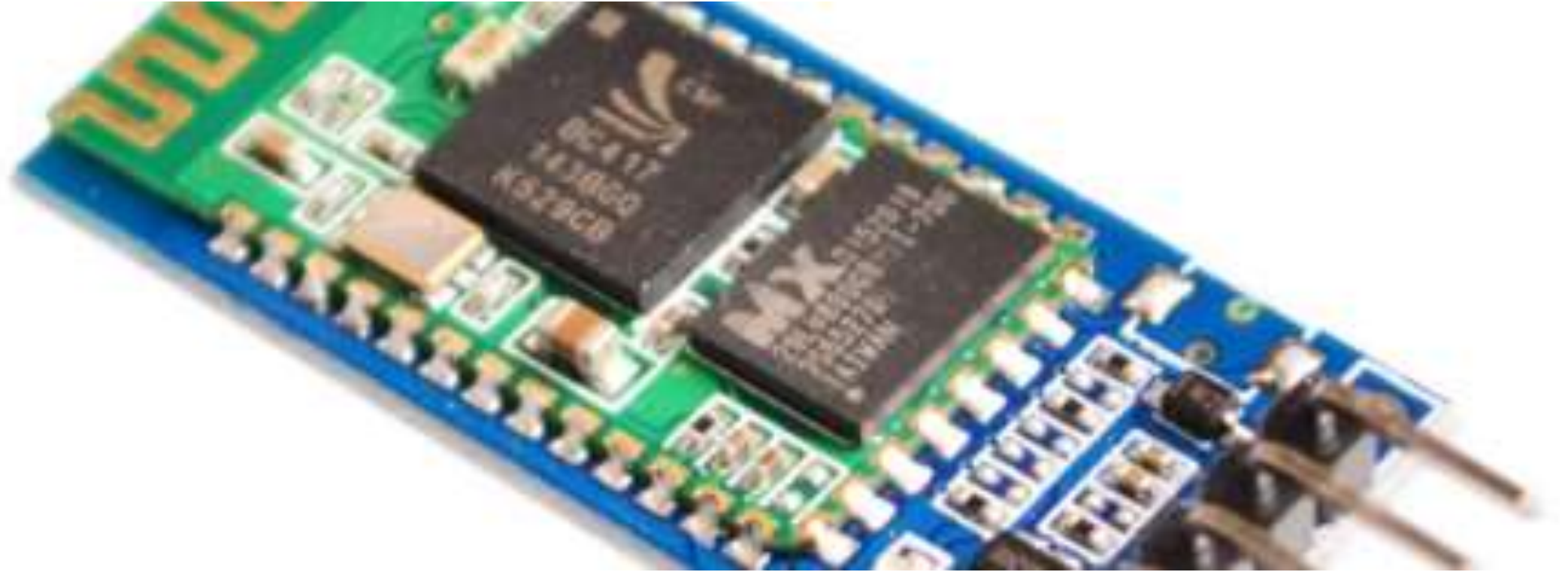
☒ Autoscroll

No line ending

115200 baud

UART is for hardware communication but what are the protocols used to get data for the GPS?

NMEA 0183 protocol !



Bluetooth protocols

Bluetooth also uses UART for hardware communication !

Let's take an example:

SIGNAL FOR THE MODULE
COMES FROM MOBILE
APP(bluetooth present in
the mobile)

```
#include <SoftwareSerial.h>
static const int RXPin = 4, TXPin = 3;
static const bluetoothBaud = 9600;
// The serial connection to the bluetooth(hc05) device
SoftwareSerial ss(RXPin, TXPin);
void setup()
{
    Serial.begin(9600);
    ss.begin(bluetoothbaud);
}
void loop()
{
    // Output bluetooth to the serial monitor
    while (ss.available() > 0){
        Serial.write(ss.read());
    }
}
```


Code looks similar ! (with GPS)

- So all the modules having UART as their communication has the same code (!! still there are libraries for the raw data to get parsed like Tinygps++ !!)

WHERE ARE BLUETOOTH PROTOCOLS HERE, IF ITS UART?

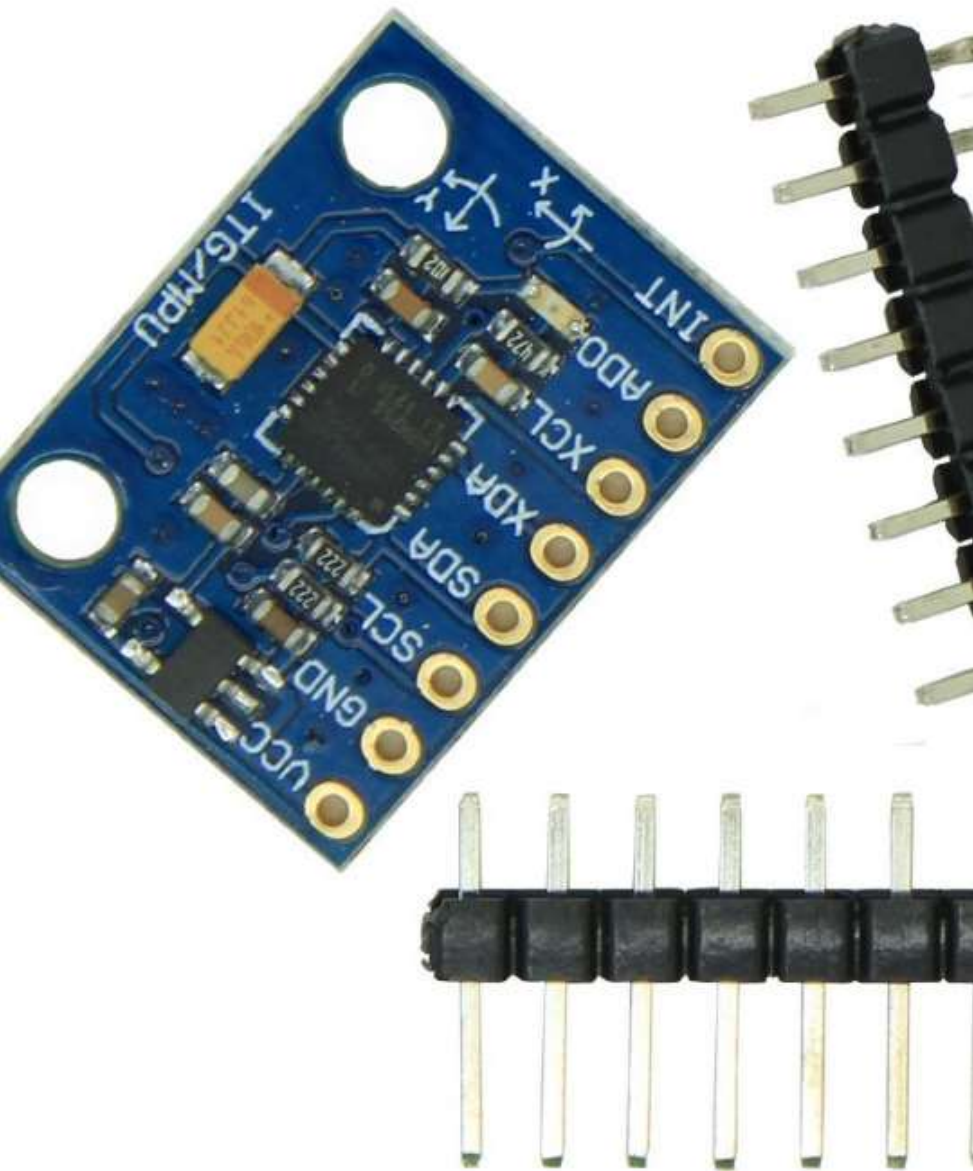
- They are for communication with other bluetooth modules!
- There are many bluetooth protocols designed by respective companies for their own advantages.



Other UART using modules present out there

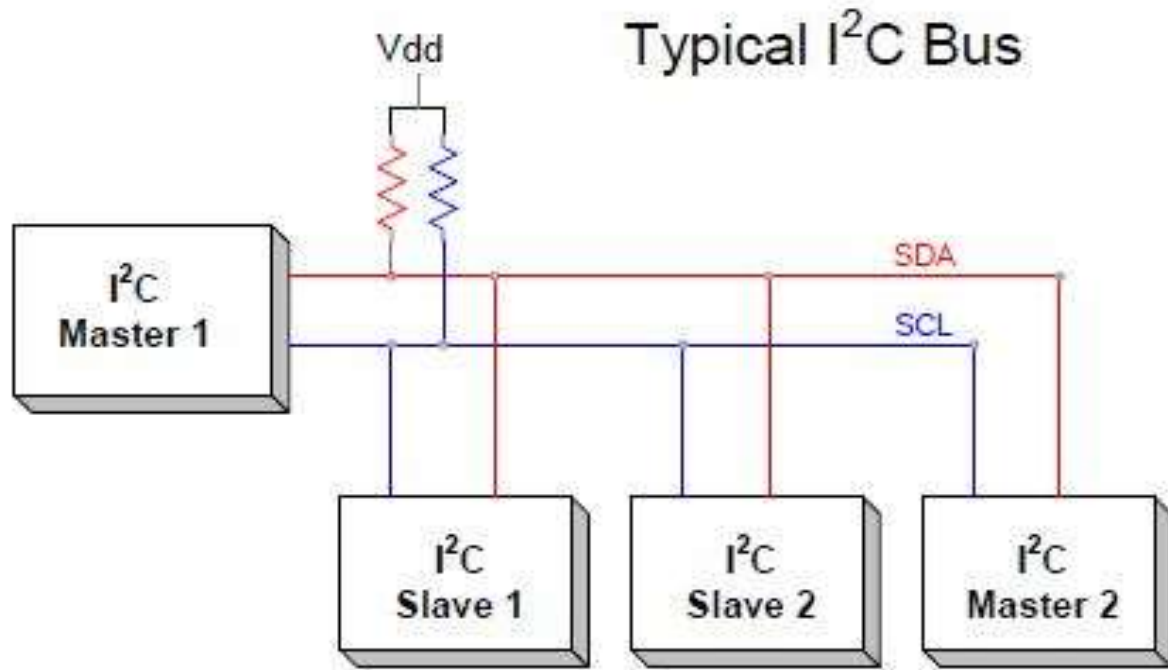
- All other GPS modules, Bluetooth modules, and RFID card reader modules to your Raspberry Pi, Arduino, or other microcontrollers.





I2C

Inter Integrated communication



- I2C uses only 2 wires, one for the clock (SCL) and one for the data (SDA).
- That means that master and slave send data over the same wire, again controlled by the master who creates the clock signal.
- I2C doesn't use separate Slave Selects to select a particular device, but has addressing.

Example Masters: All MCs can act as Masters and some sensors can act as slave as well as masters(Bluetooth)

Let us take a simple example of an accelerometer(used to find the acceleration with all the three axes)

Pin config:

VCC: 5V or 3.3V pin

GND: Ground pin

SCL (Serial clock line): Responsible for the primary I2C communication

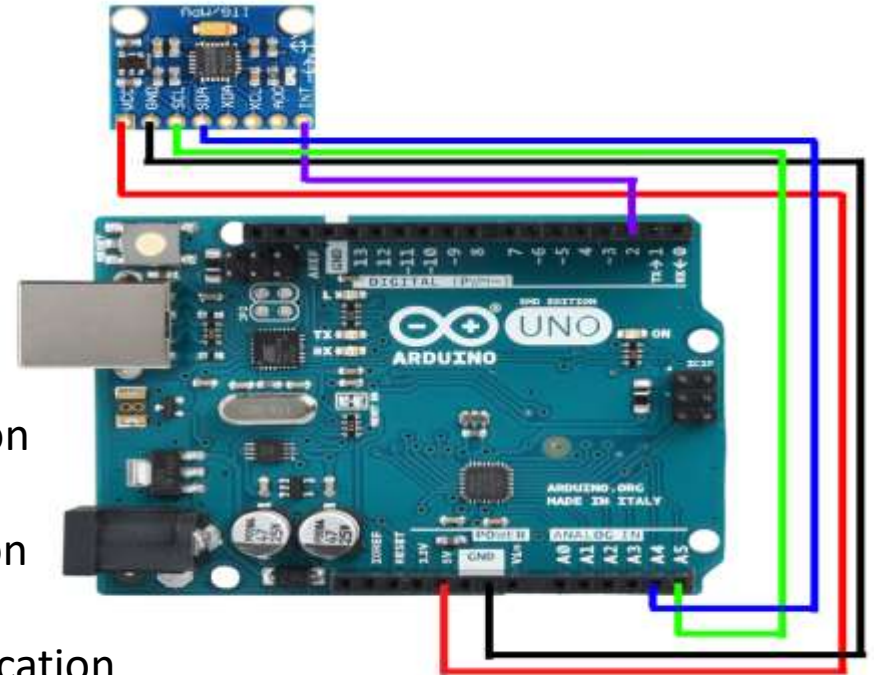
SDA (Serial data line): Responsible for the primary I2C communication

XCA (Auxiliary clock line): Responsible for the Auxiliary I2C communication

XDL (Auxiliary data line): Responsible for the Auxiliary I2C communication

ADO: Responsible for slave or master interface

INT: Interrupt Pin



How they work?

The piezoelectric effect is the most common form of accelerometer and uses microscopic crystal structures that become stressed due to accelerative forces. These crystals create a voltage from the stress, and the accelerometer interprets the voltage to determine velocity and orientation.

The capacitance accelerometer senses changes in capacitance between microstructures located next to the device. If an accelerative force moves one of these structures, the capacitance will change and the accelerometer will translate that capacitance to voltage for interpretation.

The diagram shows the formula for capacitance, $C = \epsilon_r \epsilon_0 \frac{A}{d}$, centered within a light green rectangular box. Four red curved arrows point from descriptive text labels to the variables in the formula: one from 'permeability of the material between plates' to ϵ_r , one from 'dielectric constant $\epsilon_0 \approx 8.854 \times 10^{-12} \text{ F} \cdot \text{m}^{-1}$ ' to ϵ_0 , one from 'Area between parallel plates' to A , and one from 'distance between plates' to d .

permeability of the material between plates

Area between parallel plates

distance between plates

dielectric constant $\epsilon_0 \approx 8.854 \times 10^{-12} \text{ F} \cdot \text{m}^{-1}$

$$C = \epsilon_r \epsilon_0 \frac{A}{d}$$

Simple code:

```
#include <Wire.h>      // Including the wire library
#include <MPU6050.h>    // Including the MPU6050 library
MPU6050 accelerometer; // Initializing a variable name accelerometer of type
MPU6050

void setup()
{
  Serial.begin(115200); // Starting the serial communication at 115200 baud rate.
  Serial.println("Initializing the MPU6050 sensor. Wait for a while");
  delay(2000);

  while(!accelerometer.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)) // Checking whether
  //the mpu6050 is sensing or not

  {
    Serial.println("Failed to read from the sensor. Something is Wrong!");
    delay(500);
  }
}
```

```
void loop()
{
  Vector sensor_data = accelerometer.readNormalizeAccel();
  // Reading the acceleration values from the sensor

  int pitch_value = -(atan2(sensor_data.XAxis,
    sqrt(sensor_data.YAxis*sensor_data.YAxis
    + sensor_data.ZAxis*sensor_data.ZAxis))*180.0)/M_PI;
  // Calculating the pitch value

  Serial.print(" Pitch = ");
  Serial.print(pitch_value); // Printing the pitch values on the Serial Monitor

  int roll_value = (atan2(sensor_data.YAxis,
    sensor_data.ZAxis)*180.0)/M_PI; // Calculating the Raw value

  Serial.print(" Roll = ");

  Serial.println(roll_value); // printing the Roll values on the serial
  monitor
  delay(10);
}
```

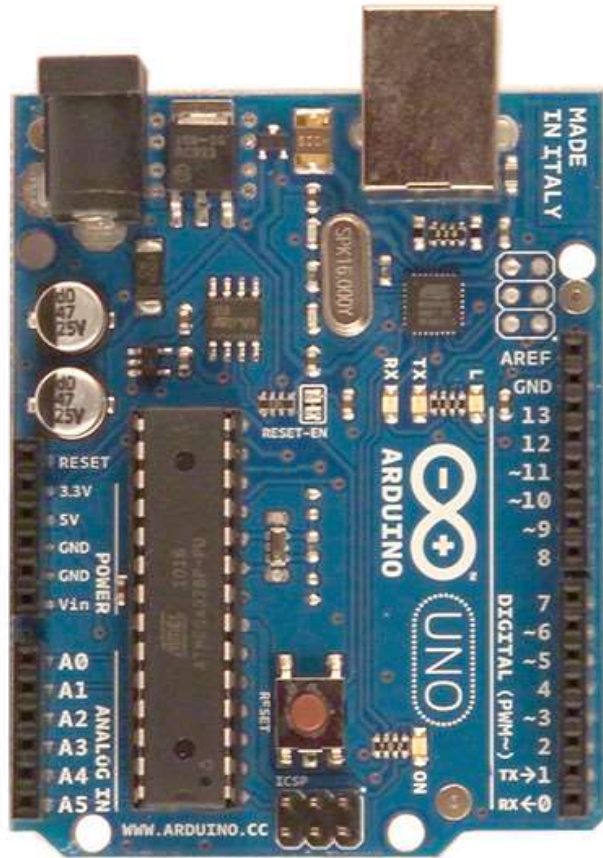

Dual, 256-Position, I2 C-Compatible Digital Potentiometer	Analog Devices
AD5248	Dual, 256-Position, I2 C-Compatible Digital Potentiometer
AD5251	Dual 64-Position I2 C Nonvolatile Memory Digital Potentiometers
AD5252	Dual 256-Position I2C Nonvolatile Memory Digital Potentiometers
ADS1115	4-channel 16-bit ADC
ADXL345	3-axis accelerometer
AK8975	3-axis magnetometer
AM2315	Humidity/Temp sensor

Still out there are many I2C devices present do your research!

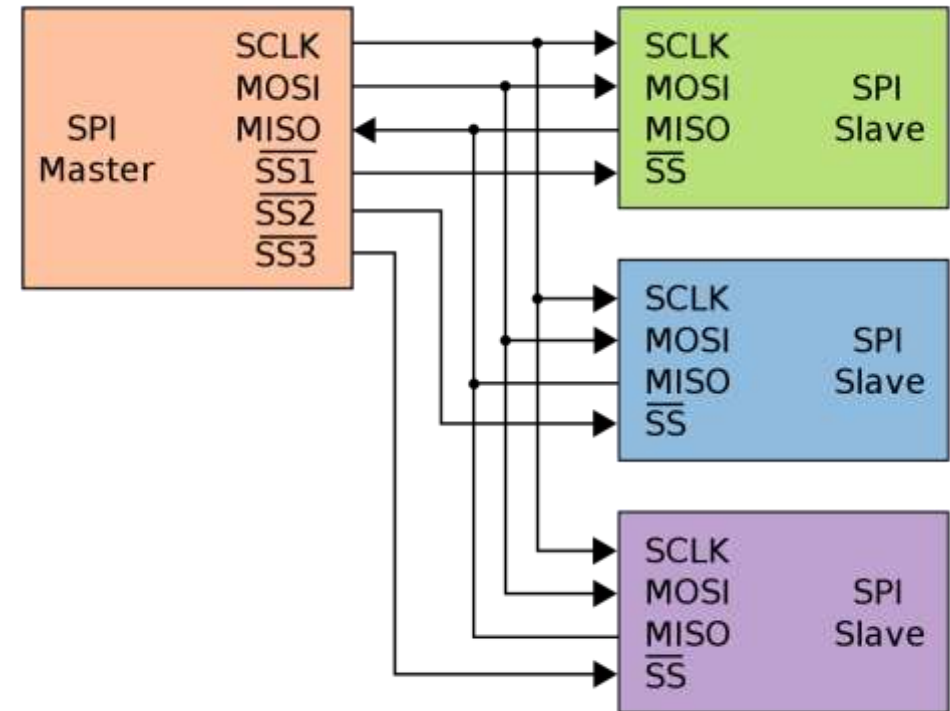
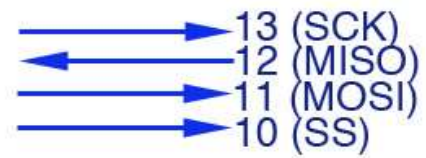


SPI

Serial Peripheral Index



SPI pins



Basic specifics:

- SPI is a common communication protocol used by many different devices. For example, SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all use SPI to communicate with microcontrollers.
- The master sends data to the slave bit by bit, in serial through the MOSI line. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with the most significant bit first.
- The slave can also send data back to the master through the MISO line in serial. The data sent from the slave back to the master is usually sent with the least significant bit first.

Pin configuration:

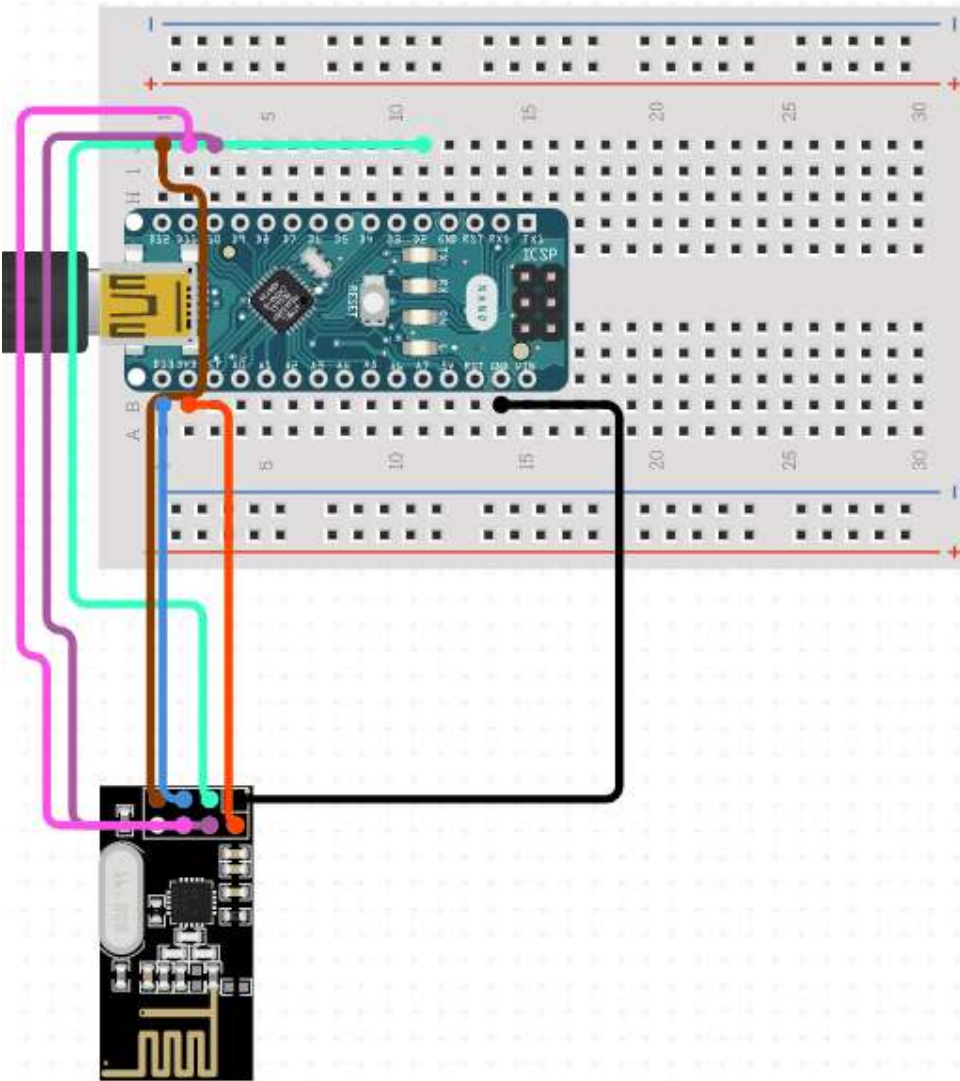
Pin Number	Pin Name	Abbreviation	Function
1	Ground	Ground	Connected to the Ground of the system
2	Vcc	Power	Powers the module using 3.3V
3	CE	Chip Enable	Used to enable SPI communication
4	CSN	chip Select Not	This pin has to be kept high always, else it will disable the SPI
5	SCK	Serial Clock	Provides the clock pulse using which the SPI communication works
6	MOSI	Master Out Slave In	Connected to MOSI pin of MCU, for the module to receive data from the MCU
7	MISO	Master In Slave Out	Connected to MISO pin of MCU, for the module to send data from the MCU
8	IRQ	Interrupt	It is an active low pin and is used only if interrupt is required



Let us take an example of nrf module which uses this protocol:

- Let's take a closer look at the NRF24L01 transceiver module. It uses the 2.4 GHz band and it can operate with baud rates from 250 kbps up to 2 Mbps.
- If used in open space and with lower baud rate its range can reach up to 100 meters.
- The power consumption of this module is just around 12mA during transmission, which is even lower than a single LED !

Simple program:



Transmitter code
(obviously we need atleast two
modules to communicate)

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
```

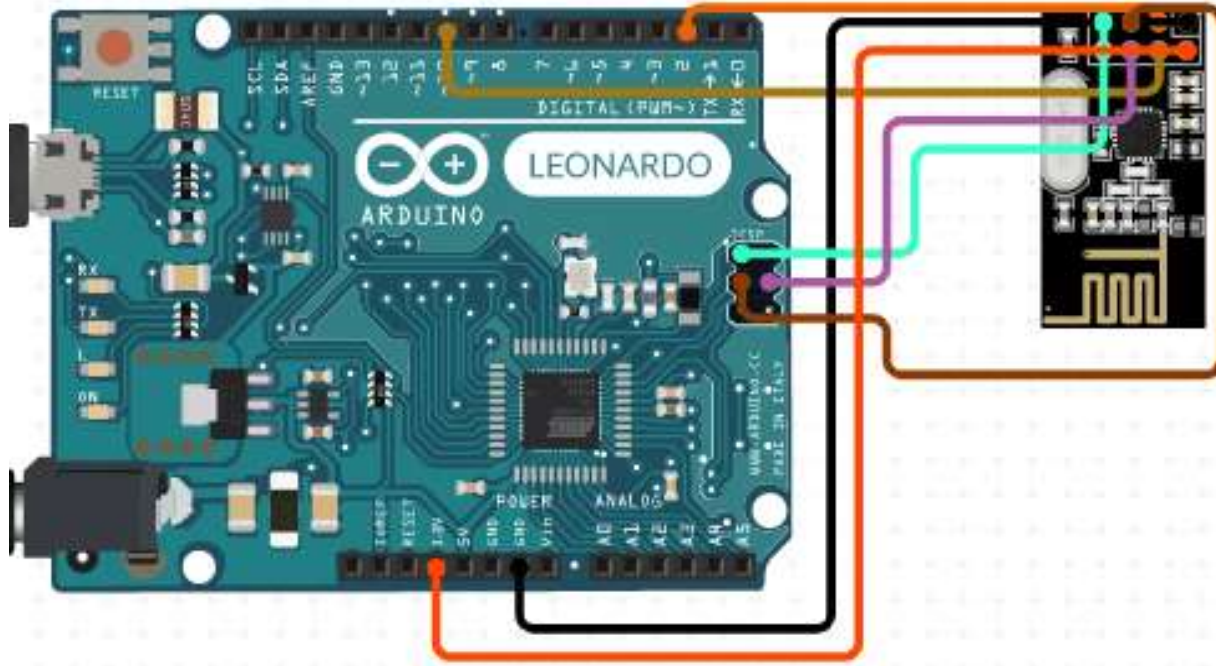
```
RF24 radio(2, 10); // CE, CSN
```

```
const byte address[6] = "00001";
```

```
void setup() {
  radio.begin();
  radio.openWritingPipe(address); //set communication for
  transmitter and receiver
  radio.setPALevel(RF24_PA_MIN); //set amplifier level
  radio.stopListening();
}
```

```
void loop() {
  const char text[] = "Hello World";
  radio.write(&text, sizeof(text));
  delay(1000);
}
```

Receiver code:



```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(2, 10); // CE, CSN

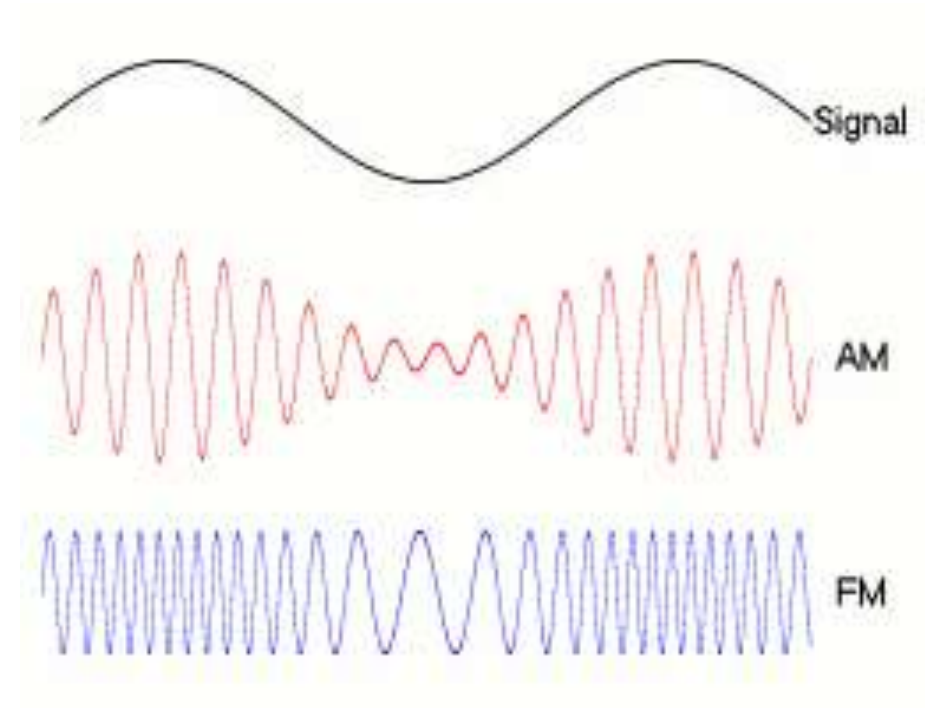
const byte address[6] = "00001";

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN); // min
  amplifier level
  radio.startListening(); // so that it doesn't
  // transmit
}

void loop() {
  if (radio.available()) {
    char text[32] = "";
    radio.read(&text, sizeof(text));
    Serial.println(text);
  }
}
```

**We discussed about the hardware communication !
But what about the over the air communication protocol!**

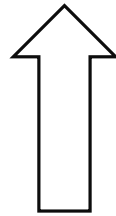
Simple modulation and demodulation techniques are employed for the communication purposes



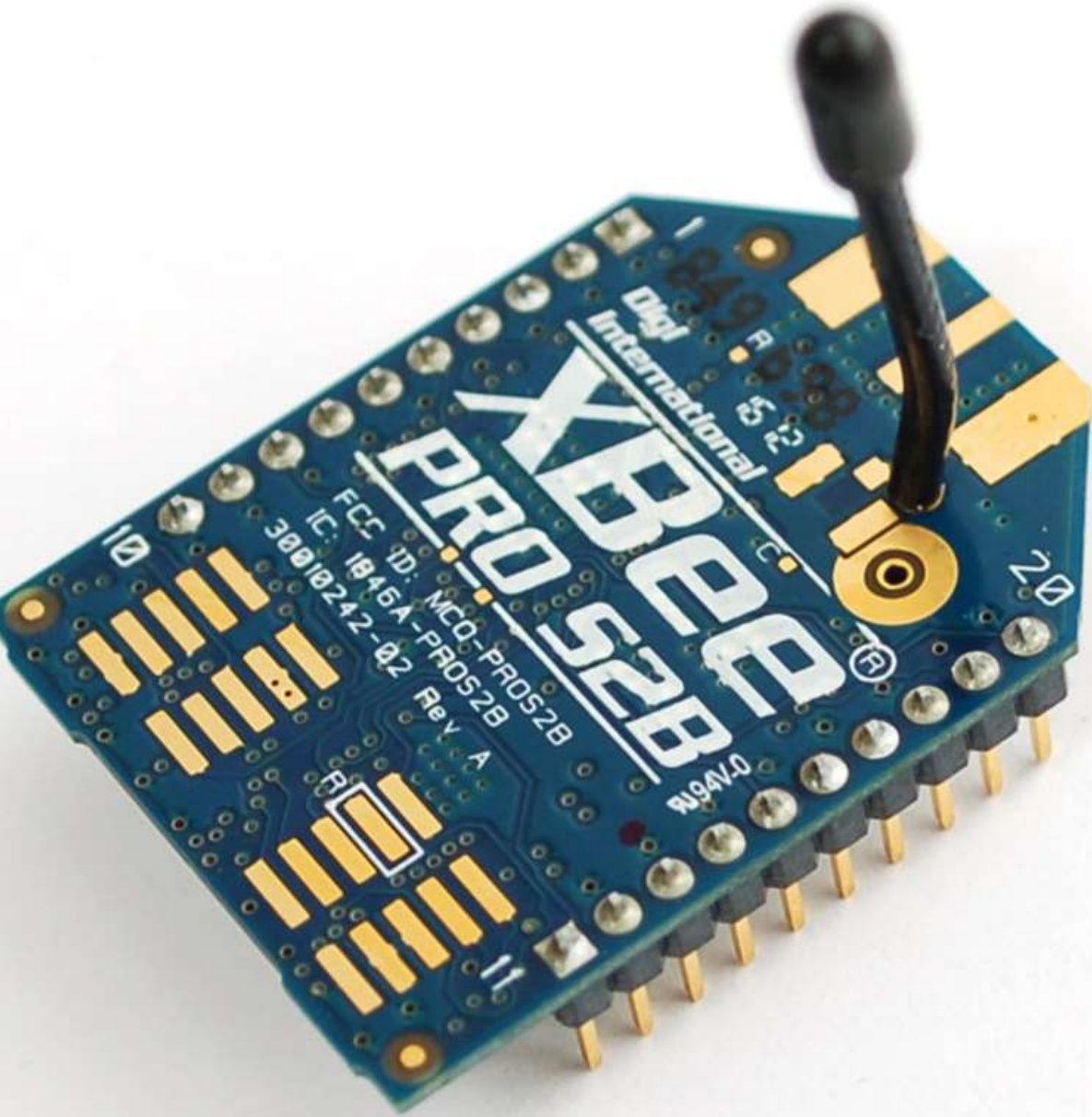
This is the simple example of how SPI used!

There are many other
modules which use spi as a
communication channel
explore them!

<https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html>



Check this site for more insight!



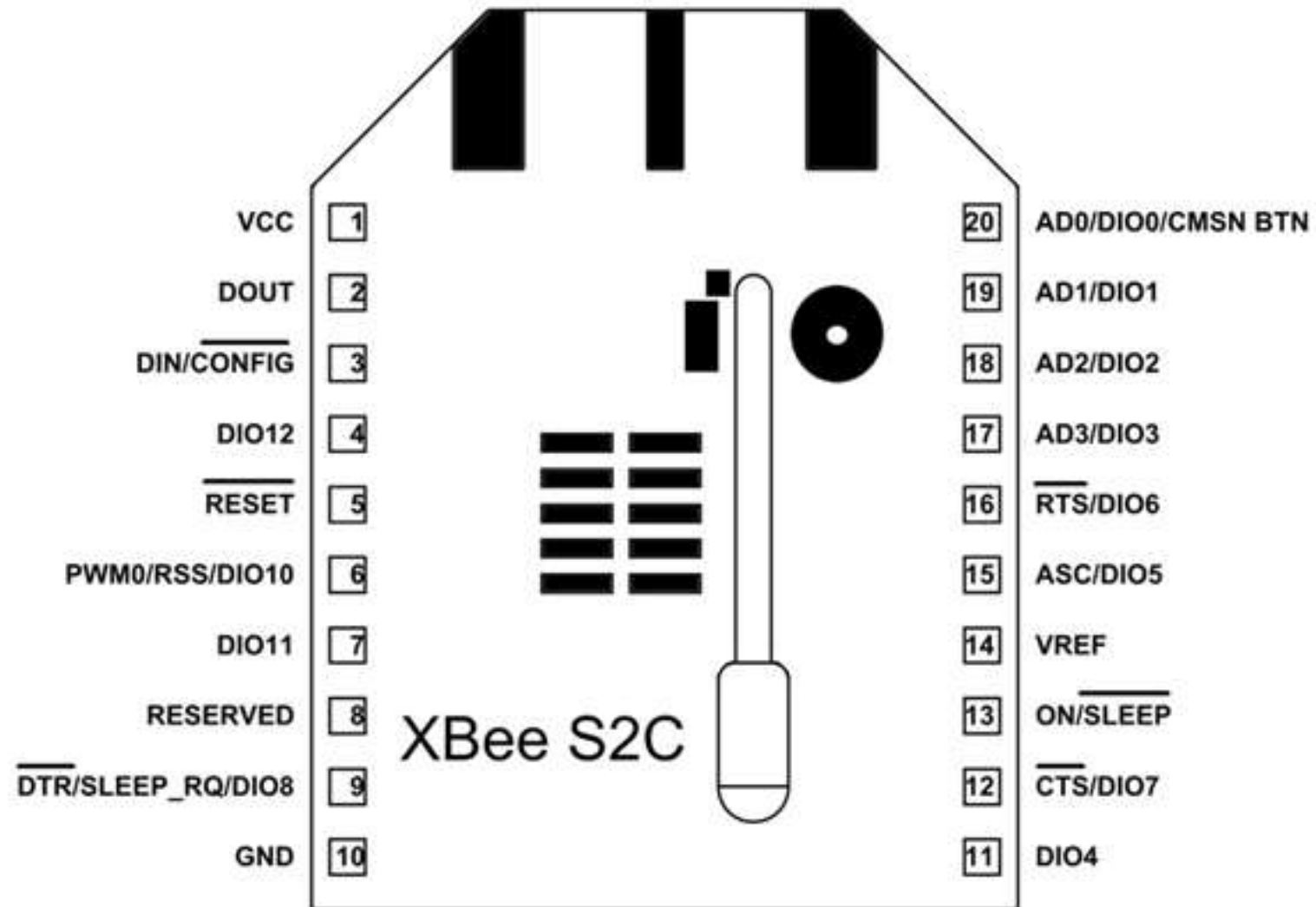
Zigbee
protocols

- Zigbee is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios, such as for home automation.
- The Zigbee protocol was designed to provide an easy-to-use wireless data solution characterized by secure, reliable wireless network architectures.



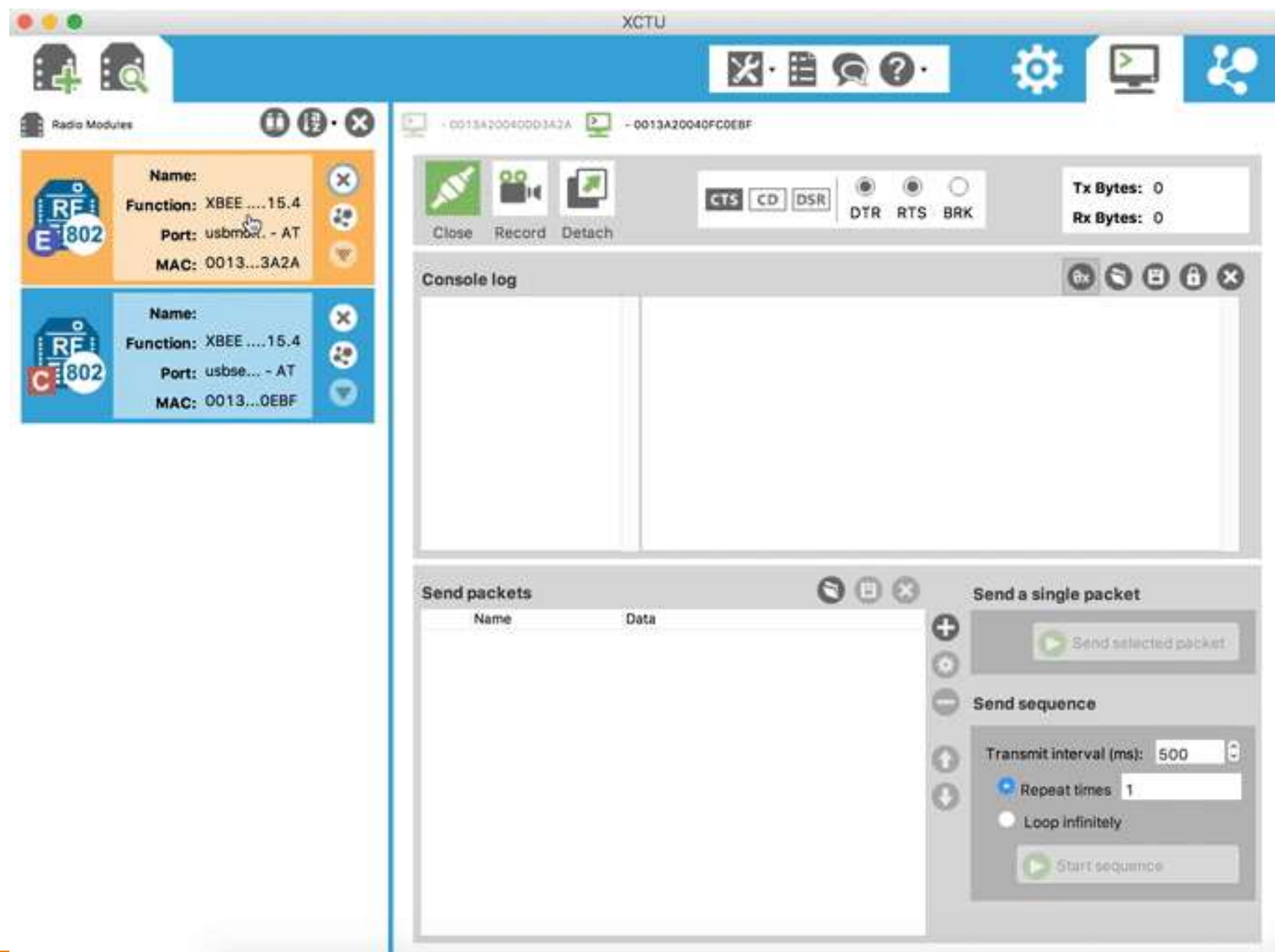
There are so many advantages with these protocols such as encryption of data ,low latency and many more..

We shall concentrate on X bee(A module using this Protocol)

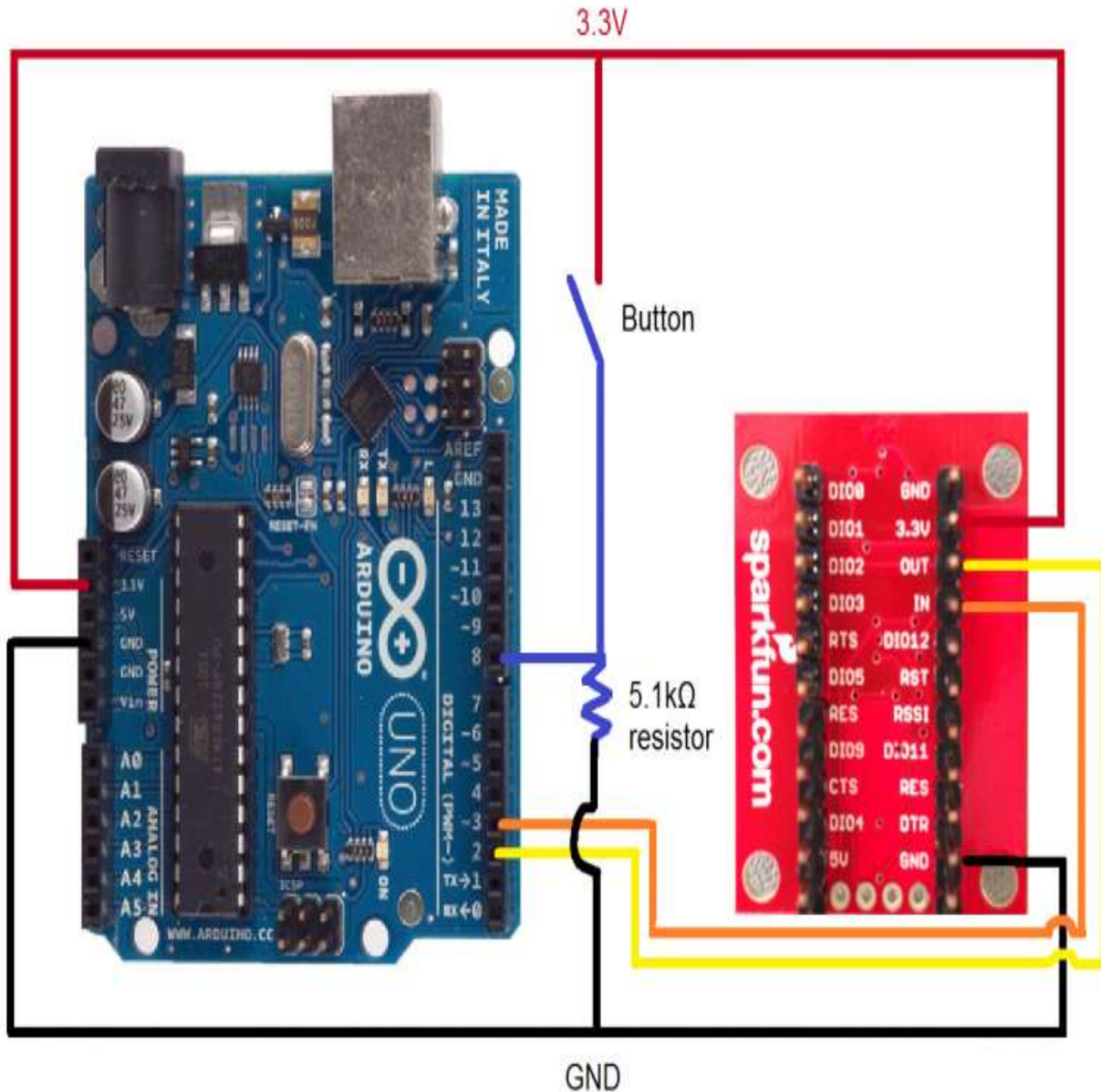


Pin	Name	Description
1	VCC	Power supply
2	DOUT/DIO13	UART data out pin (TXD)/GPIO
3	DIN/CONFIG / DIO14	UART data in pin (RXD)/ GPIO
4	DIO12/SPI_MISO	GPIO/ Master Input-Slave Output pin of SPI interface
5	RESET	Module Reset pin
6	RSS PWM /DIO10	RX Signal Strength Indicator pin / GPIO
7	PWM1/DIO11	Pulse Width Modulator/GPIO
8	RESERVED	Do not connect
9	DTR/SLEEP_RQ/ DIO8	Pin Sleep Control line /GPIO
10	GND	Ground
11	DIO4/ SPI_MOSI	GPIO/Master Output-Slave Input pin of SPI interface
12	CTS/DIO7	Clear-to-send flow control/GPIO
13	ON_SLEEP/DIO9	Device status indicator/GPIO
14	VREF	Voltage Reference for ADC
15	ASC/DIO5	Associate Indicator/GPIO
16	RTS/DIO6	Request to send flow control/ GPIO
17	AD3/DIO3/SPI_SSEL	Analog input/GPIO/SPI slave select
18	AD2 /DIO2/SPI_CLK	Analog input/GPIO/SPI clock
19	AD1/DIO1/SPI_ATTN	Analog input/GPIO/SPI attention
20	AD0/DIO0/C	Analog input/GPIO/ Commissioning button

XCTU software
to set the
configuration
of the
Xbee module !



A example using this module



/* Input-side (button) **Transmitting side!**
Arduino code */

```
#include "SoftwareSerial.h"  
// RX: Arduino pin 2, XBee  
pin DOUT. //TX: Arduino pin  
3, XBee pin DIN
```

```
SoftwareSerial XBee(2, 3);  
int BUTTON = 8;  
void setup() {  
  // Baud rate MUST match  
  XBee settings (as set in  
  XCTU) pinMode(BUTTON, INPUT);  
  XBee.begin(9600); }
```

```
void loop() {  
  if (digitalRead(BUTTON) ==  
  HIGH)  
  {  
    XBee.write('H'); delay(50);  
  }  
}
```



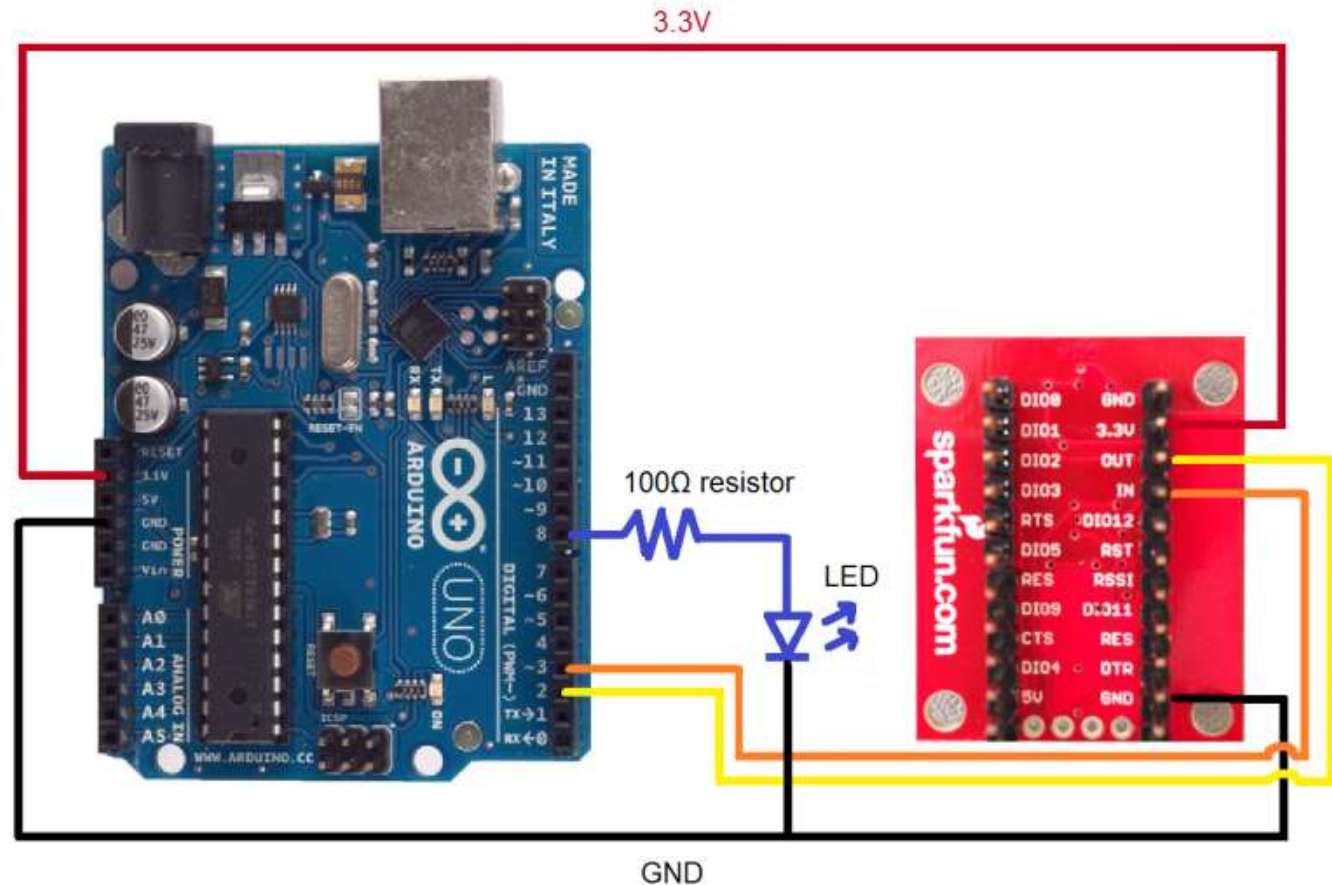
```

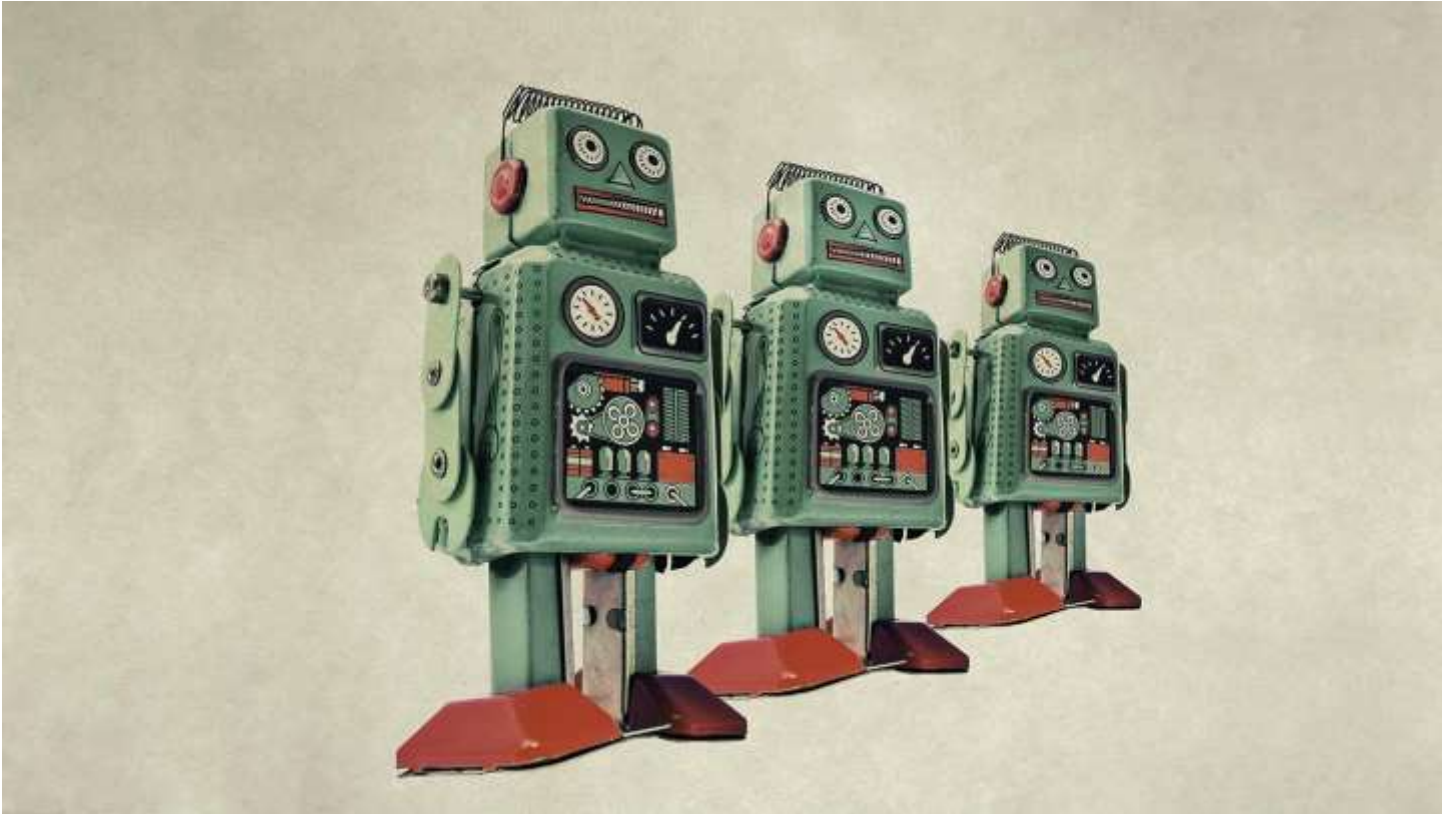
/* Output-side (LED) Arduino code */
#include "SoftwareSerial.h" // RX:
Arduino pin 2, XBee pin DOUT. TX:
Arduino pin 3, XBee pin DIN
SoftwareSerial XBee(2, 3);
int LED = 9;
void setup() { // Baud rate MUST match
XBee settings (as set in XCTU)
XBee.begin(9600);
pinMode(LED, OUTPUT); }
void loop() {
  if (XBee.available()){
char c = XBee.read();
if (c == 'H') {
digitalWrite(LED, HIGH); delay(50);
} else {
  digitalWrite(LED, LOW);
}
} else {
  digitalWrite(LED, LOW);
}
}

```

Receiving side !

If you set up everything correctly then data should transmit





There are still
many more
modules to
work on !
Explore them
according to
your project!