Arduino Communication

Written by Mark Webster and ??

Summary
Learn how Arduino microcontrollers can communicate with other electronic devices using a wired connection, infrared beams, or radio waves.

Introduction

Microcontrollers like Arduinos are powerful by themselves and can be used in stand-alone applications that interact “intelligently” with their environment. However, when microcontrollers can communicate or network with other microcontrollers or computers another entire magnitude of capability emerges.

Serial

Serial communication sends bits sequentially through two wires, a transmit (TX) and a receive (RX) wire. The bits are sent at a different clock rate (called baud rate), and can have internal checksums (parity bits) to detect corruption of the information. The bits are grouped into 8 or 7 bit bytes which represent one ASCII character. Serial communication is one of the oldest ways computers and humans sent data back and forth.

USB UART

Most Arduino microcontroller boards have a hardware UART for serial communication over the USB. The hardware serial is referenced by the built-in Serial object.
In the setup() section of code initialize the Serial object to the desired baud rate, such as Serial.begin(9600). The maximum baud rate is 115200 with 9600 being the default.

Character strings can be sent back to the Arduino IDE using the various Serial.print() methods of the object. The different print() methods print an INT, a CHAR string, a FLOAT, or BYTE. The programmer does not need to use a different print() method for different data types.
For more sophisticated formatting use the C function “sprintf()” and then Serial.print() the resulting string.
To add a newline at the end of the print(), use Serial.println();

To read a byte from the hardware serial port first test if any characters are available to read, and read one character if something is in the pipeline. There is a buffer which can store 64

characters and the available() method returns the number of bytes. The standard if() test is true if anything more than 0 bytes are in the buffer.

https://www.arduino.cc/en/serial/available

```
if( Serial.available() ) {
        aByte = Serial.read();
}
```

The Arduino IDE Serial Monitor allows the user to read and write data to the Arduino, although any terminal program or Python serial program can also read and write data from the Arduino.

Note: the Arduino Mega has three hardware serial ports, although other Arduinos have one.

Software Serial

When more serial ports are desired, or no hardware UART can be used, or some electronic module needs serial data communication, the Arduino IDE offers a software serial port.

https://www.arduino.cc/en/Reference/softwareSerial

The software serial is slower since everything is done by the CPU, not in hardware. Often a maximum speed of 4200 baud is possible. Usually this is not a problem. Two pins are used, one for transmit (TX) and one for receive (RX)

If using two Arduino's connected by wires, the TX and RX pins must be reversed.

To use Software serial, include the SoftwareSerial.h file and create an instance of the software serial object. It must be told what pin is transmit and what pin is receive.

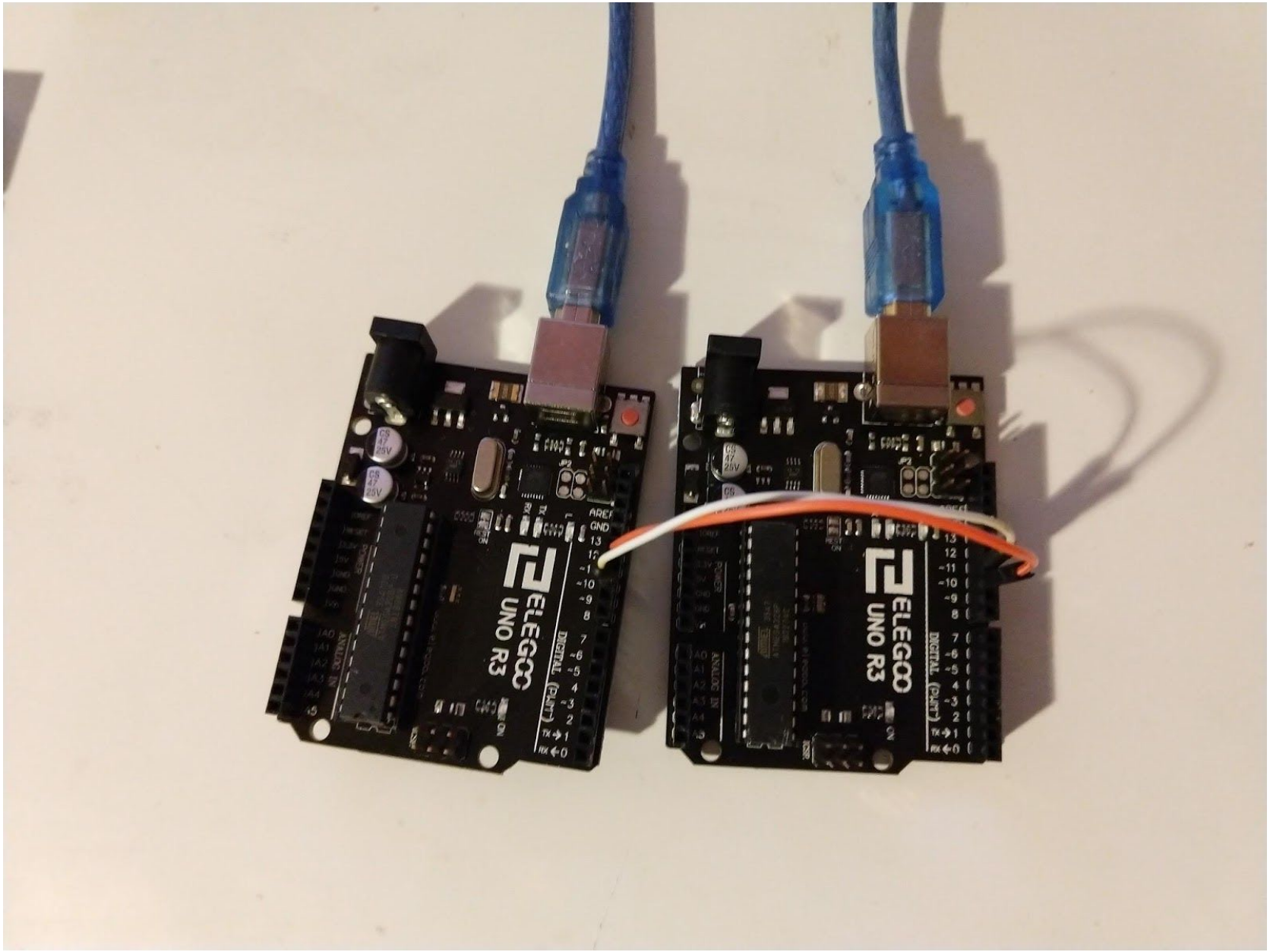
The software serial example included with the Arduino IDE indicates how to use the serial port. Here is the example code for SoftwareSerialExample

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
```

```
  }


  Serial.println("Goodnight moon!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(4800);
  mySerial.println("Hello, world?");
}

void loop() { // run over and over
  if (mySerial.available()) {
    Serial.write(mySerial.read());
  }
  if (Serial.available()) {
    mySerial.write(Serial.read());
  }
}
```

Example 1: Software Serial

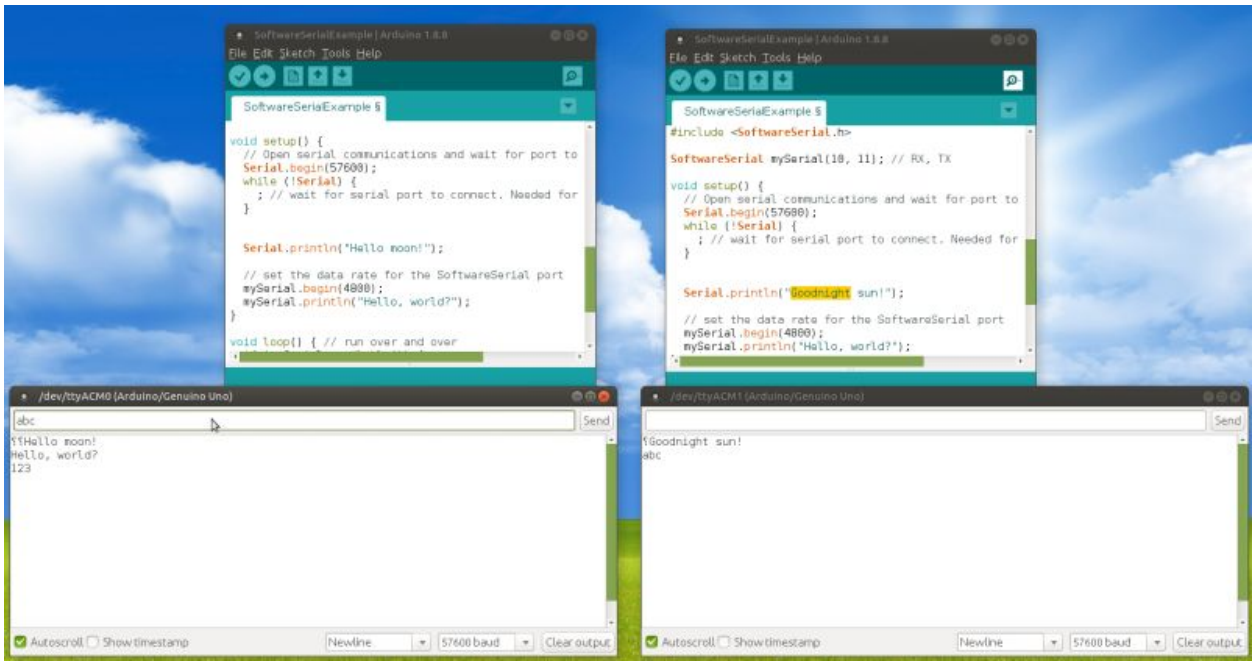
Hardware: Two Arduino Uno's, two USB cables, two m-m jumper wires. Connect the wires to pins 10 and 11 on each Arduino, but switch the wires. The wire on one Arduino pin 10 is connected to pin 11 on the other Arduino. And vise-versa.
Software: Two copies of the Arduino IDE running at the same time. (Not two windows of one IDE)


ELEGOO
UNO R3
ELEGOO
UNO R3

Connect both Arduino Uno's via USB to two different serial ports on the laptop or desktop. Open two instances of the Arduino IDE. Set the port in each IDE to a different port. For both Arduino's open the SoftwareSerialExample, which is probably under the Examples for Arduino Uno.

In both IDE's open the File -> Examples -> Software Serial -> SoftwareSerialExample.

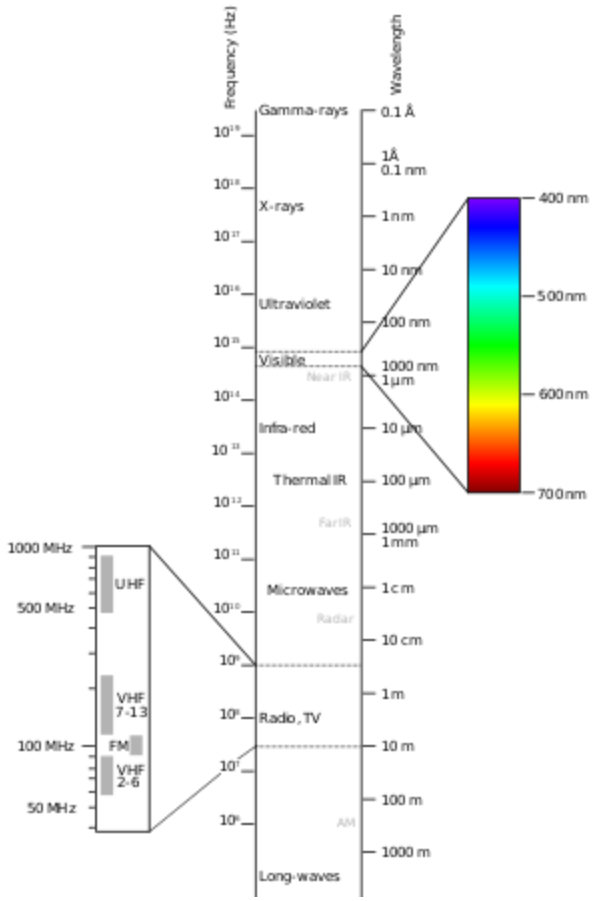
Edit one sketch to say "Goodnight sun", the other to say "hello moon". Compile and upload both sketches to the Arduino Unos. In each IDE, open the serial monitor, and set the baud rate to 57600.
Anything typed into one serial monitor (up to 3 characters long) will appear in the other monitor.



Electromagnetic Spectrum

Wireless communication between Arduinos or other microcontrollers is done with some frequency of electromagnetic radiation. The higher the frequency, the higher the possible data transfer rate. Lower frequency radio waves travel farther or penetrate walls more easily. So it is a trade-off.

The FCC allocates wireless communication on certain frequencies. For example 2.4 Ghz and 5 Ghz is allocated to Wifi, 2.45 Ghz to bluetooth, 433 Mhz, 27 and 49 Mhz to radio control toys, etc. The frequencies used in IoT (Internet of Things) uses one of the FCC allocated frequencies.



https://en.wikipedia.org/wiki/Electromagnetic_spectrum

The general categories of radiation are:

- Gamma radiation
- X-ray radiation
- Ultraviolet radiation
- Visible radiation
- Infrared radiation
- Terahertz radiation
- Microwave radiation
- Radio waves

Maker projects generally use visible light, infrared, microwave (for wifi or bluetooth), or radio waves.

Infrared

Infrared pulses are commonly used for nearby communication, for example in TV remote controls. An Arduino can receive and decode the IR transmitter signal, or can be used as a substitute to an IR remote control.

One popular explanation for an Arduino IR is at Sparkfun:
https://learn.sparkfun.com/tutorials/ir-communication/all

Two parts are needed: an infrared LED (transmitter) and an IR receiver diode. One can also use slightly more expensive IR breakout boards too as shown in this example:
http://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/

The signal from an IR remote control is encoded on a carrier signal. The IR receiver turns the signal into a series of hexadecimal bytes. The IRremote library is used to decode the bytes.
http://z3t0.github.io/Arduino-IRremote/

The IRremote library can be used to find the codes for any IR remote control device.

Example 2: IR remote

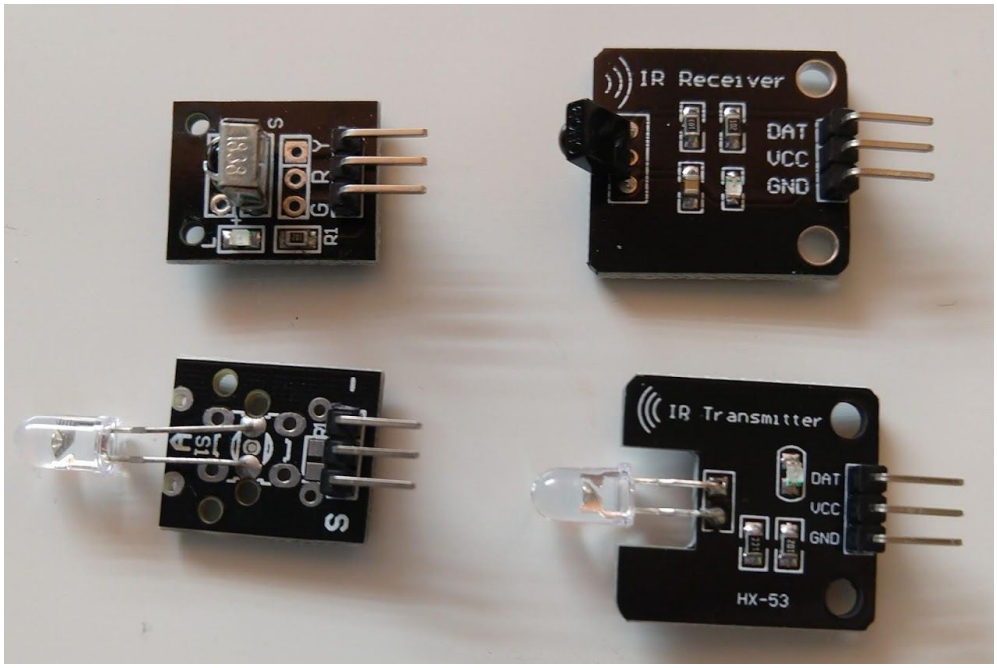
Hardware: Arduino Uno, IR receiver, 3 jumper wires, IR remote control
Software: Arduino IDE, Arduino-IRremote from Github
https://github.com/z3t0/Arduino-IRremote

Download and add the library using the Library Manager add ZIP file.

Set up an Arduino with an IR receiver breakout board. Point a remote control at the Arduino receiver and display the hex codes through the Serial Monitor.

Infrared sensors and transmitters can look just like transistors:

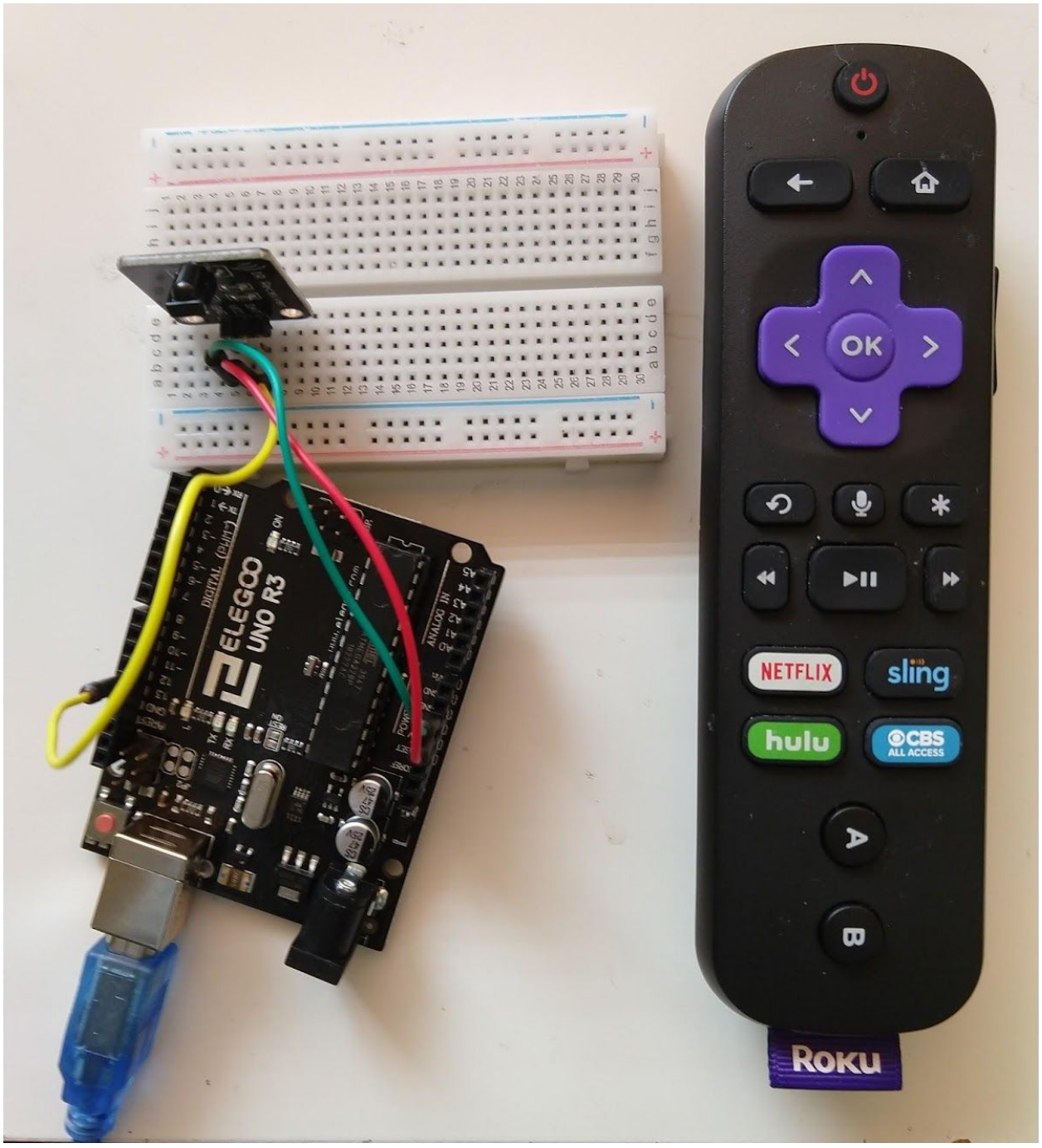
or the more commonly are soldered into modules for Arduino experimenters. Here is a photo of two different transmitter and receiver modules. There are only 3 pins to use, +5 V, ground, and a data pin.



The standard IRRemoteControl library can be installed and used with the Library Manager.

This example is simpler since it just uses the IR receiver module to decode control codes from an IR remote control.

Or a photo:

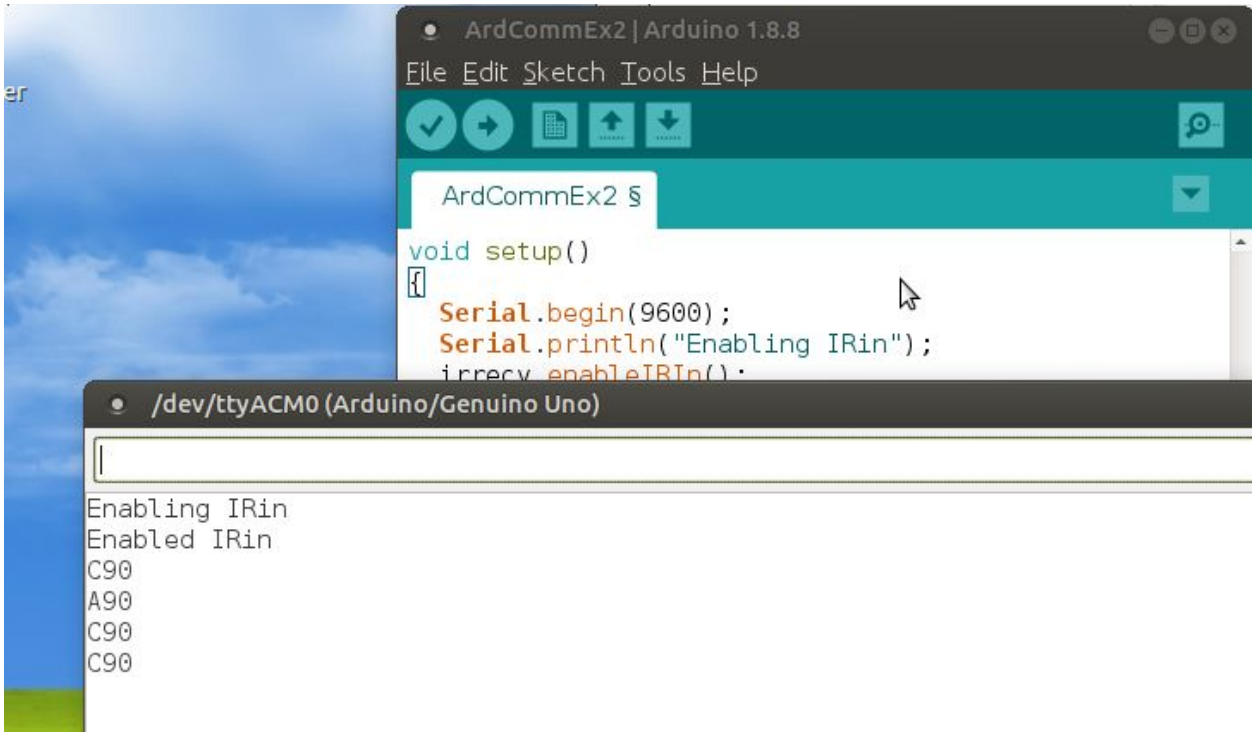


Connect the data line on the receiver to pin 12.
More information can be found at:
https://www.instructables.com/id/IR-REMOTE-DECODER-USING-ARDUINO/

Here is the Arduino code:
/* Arduino Communications IR decoder Example 2

```
 *  Pins are +5 V, Gnd, pin 12
 *
 * Uses the library https://github.com/z3t0/Arduino-IRremote
 * Install the ZIP using the Library Manager add a ZIP
 */

#include <IRremote.h>

int IRPIN = 12;

IRrecv irrecv(IRPIN);

decode_results result;

void setup()
{
  Serial.begin(9600);
  Serial.println("Enabling IRin");
  irrecv.enableIRIn();
  Serial.println("Enabled IRin");
}

void loop()
{
  if (irrecv.decode(&result))
  {
    Serial.println(result.value, HEX);
    irrecv.resume();
  }
  delay(500);
}
```

Point an IR remote control at the IR receiver and view the resulting codes with the Serial Monitor. Be sure the Serial Monitor is set to 9600 baud.



The hex code for each button on the remote control will be recorded. These values could be used in an Arduino program with an IR transmitter to create a custom remote control, or to duplicate another remote control.

Note: in environments with many lights that could be producing spurious IR signals, building a cave around the IR receiver will make the hex codes received more consistent.

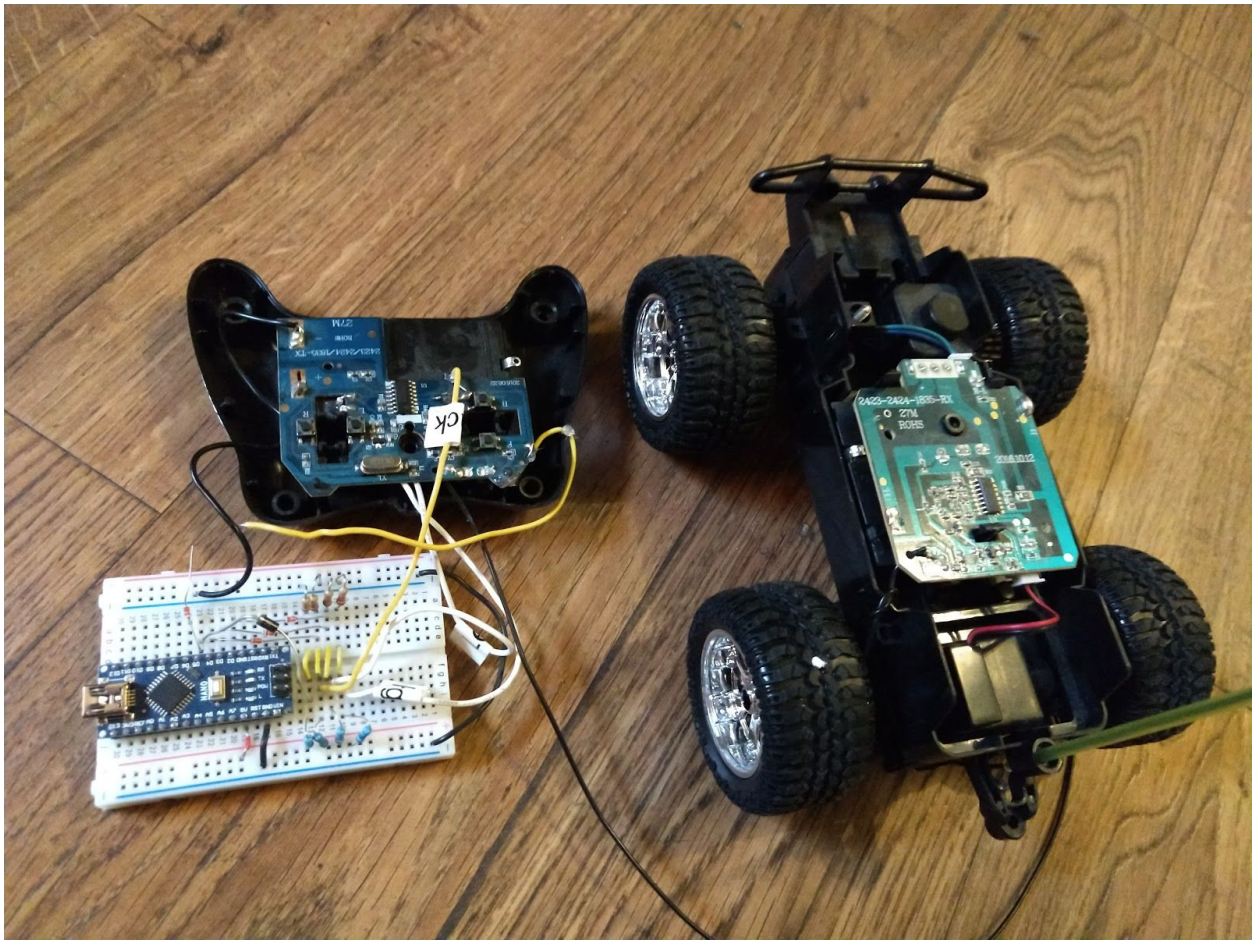
Radio

Much communication in the IoT world is through radio frequency. It can be short distance (meters) or long distance (kilometers). With RF transmission microcontrollers can be linked into mesh networks.

Most radio connections are point-to-point with one transmitter and one receiver. However, RF transmission spreads widely so many receivers can pick up the same signal. Often channels or encryptions are used to make the communication point-to-point.

27 and 49 Mhz

These are two frequencies used by most inexpensive radio controlled cars and toys. It is possible to create an electronic receiver or transmitter circuit and bit-bang out a signal from an Arduino. More commonly a commercial RC controller is purchased and hacked by desoldering and soldering components,



433 Mhz RF transceivers

The frequency band of 433 Mhz is common for garage door openers, baby monitors, even headphones. The range is typically 10 meters although specialized units reach 2 km.

Two Arduinos can communicate at 433 mhz if one has a transmitter and another a receiver module connected. These modules are very inexpensive, about $2 a pair on Amazon. Usually a library like RadioHead is used https://github.com/PaulStoffregen/RadioHead

Many RF remote controls use the 433 Mhz frequency so an Arduino can snoop on those frequencies and read codes. It can also transmit those codes to simulate a remote control. Some remote control appliance switches also use 433 Mhz. More info about remote controls is at:
https://randomnerdtutorials.com/decode-and-send-433-mhz-rf-signals-with-arduino/

The transmitter and receiver can also have a wire loop antenna soldered on for longer range.

(replace)

Example 3: 433 Mhz RF

Hardware: Two Arduino Unos, cable, 433 mhz transmitter and receiver, 6 jumper wires
Software: Radiohead ASK library, two instances of the Arduino IDE.

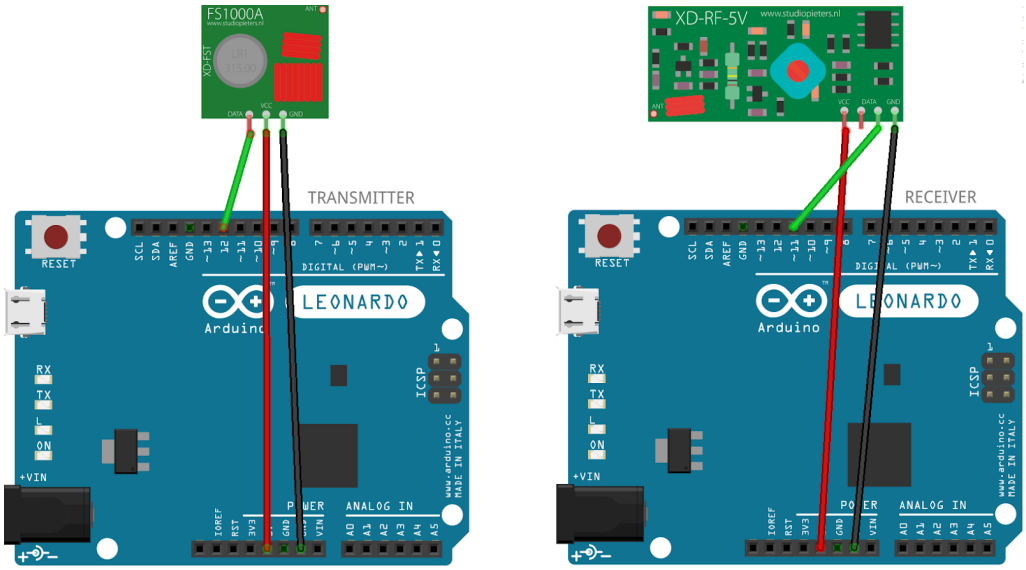
Install the Radiohead library from a ZIP file if not already installed.

Set up two Arduinos, one with an RF receiver and the other an RF transmitter. Either launch two copies of the Arduino IDE on one computer or use two different computers.

The two Arduinos communicate like over a serial port. Digital pin 12 is connected to the middle pin on the transmitter module, digital pin 11 is connected to the right inner pin on the receiver.

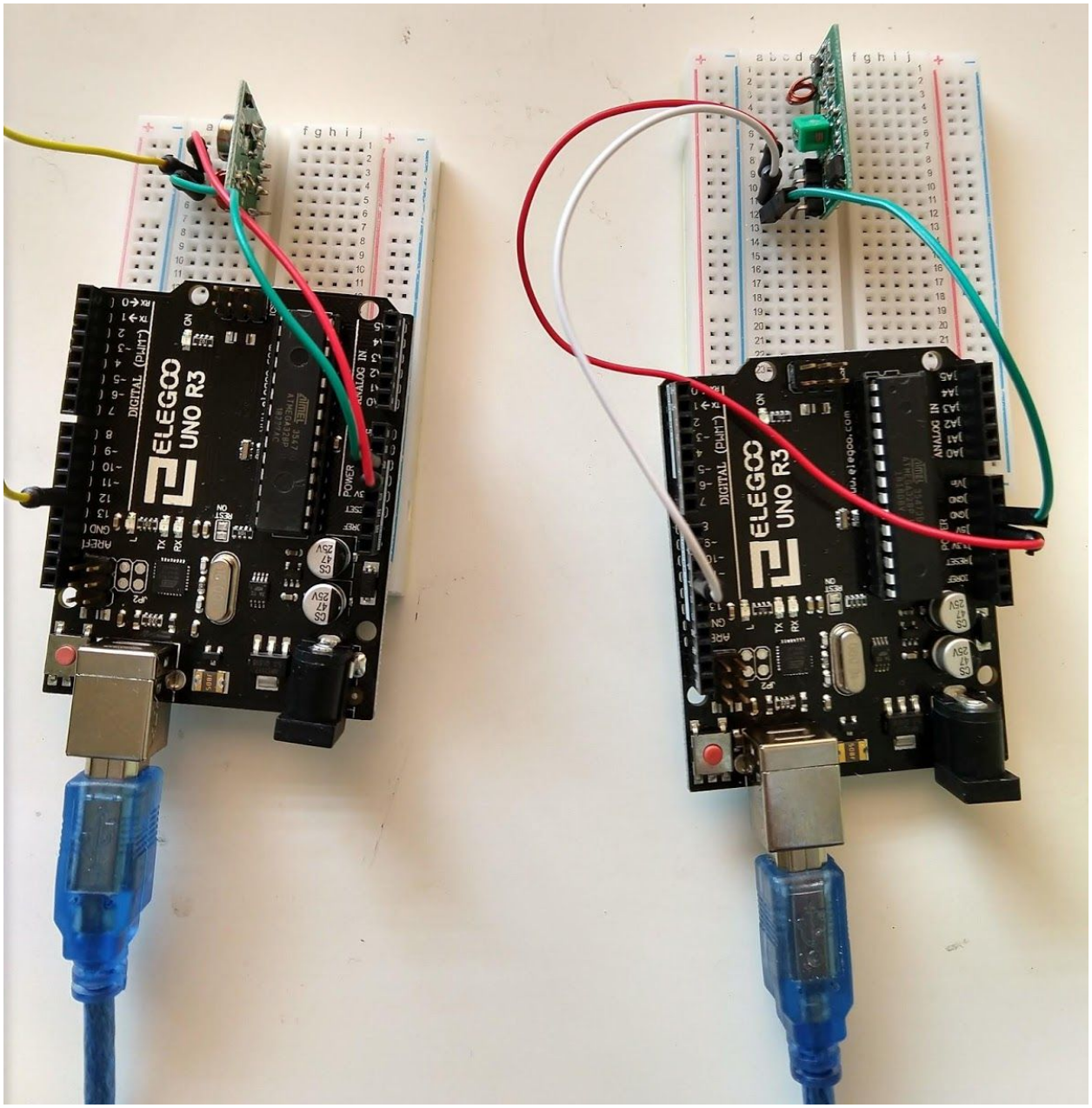
https://randomnerdtutorials.com/rf-433mhz-transmitter-receiver-module-with-arduino/

A Fritzing diagram looks like:



(replace)

A photograph looks like:

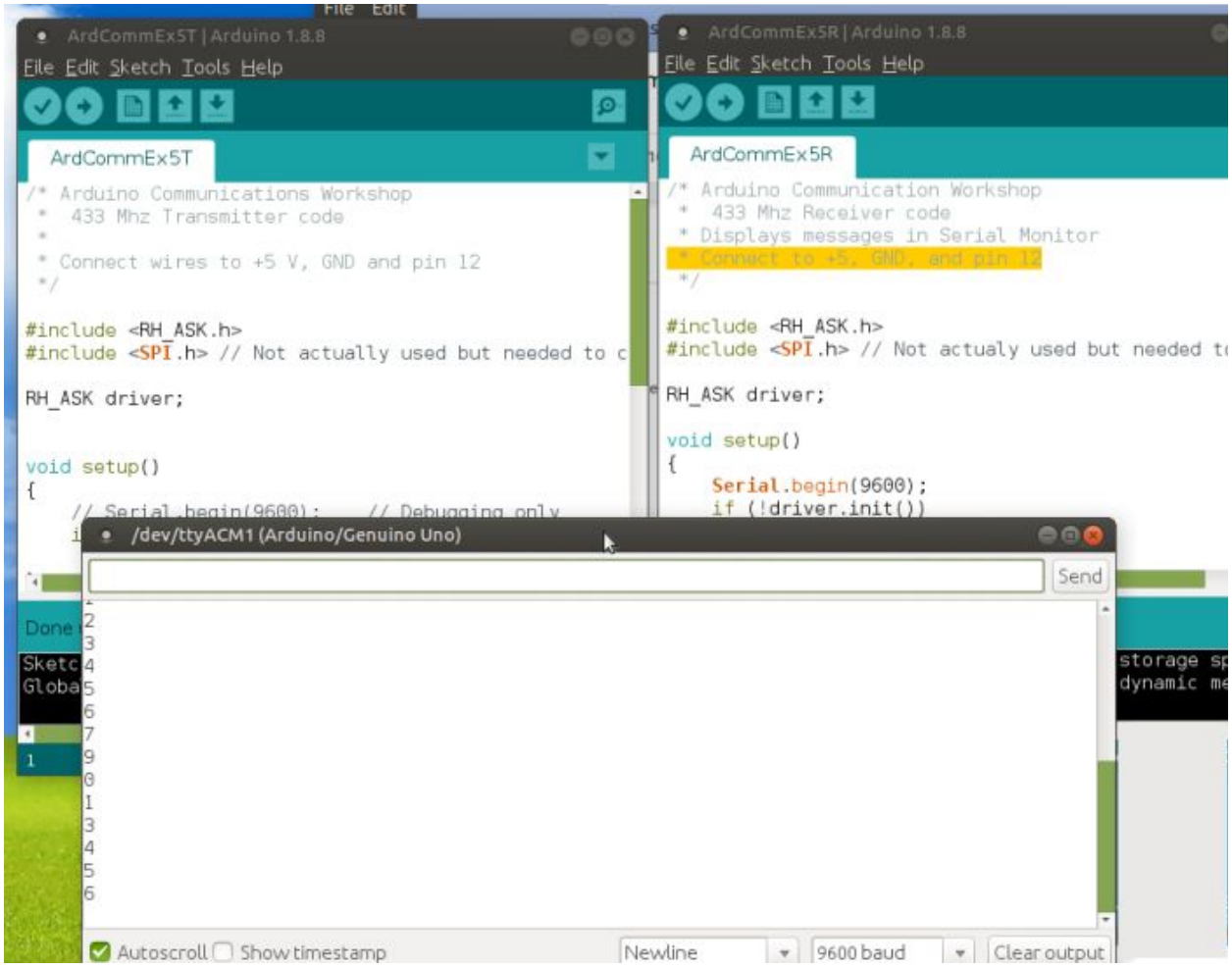



```
// ********************Transmitter Code ************************
/* Arduino Communications Workshop
 *  433 Mhz Transmitter code
 *
 * Connect wires to +5 V, GND and pin 12
 */

#include <RH_ASK.h>
#include <SPI.h> // Not actually used but needed to compile
```

```
RH_ASK driver;


void setup()
{
    if (!driver.init())
         Serial.println("init failed");
}

void loop()
{
    int i;
    char msg[10];

   for(i=0;i<10;i++){
           sprintf(msg,”%d”,i);
           driver.send((uint8_t *)msg, strlen(msg));
           driver.waitPacketSent();
           delay(1000);
  }

}


// **********************Receiver Code******************
/* Arduino Communication Workshop
 *  433 Mhz Receiver code
 * Displays messages in Serial Monitor
 *
 * Connect to +5, GND, and pin 11    (check RH_ASK.h for verification)
 */

#include <RH_ASK.h>
#include <SPI.h> // Not actually used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600);
    if (!driver.init())
         Serial.println("init failed");
```

```
}

void loop()
{
     int value;
    uint8_t buf[12];
    uint8_t buflen = sizeof(buf);
    if (driver.recv(buf, &buflen)) // Non-blocking
    {

      value = buf[0]-'0';
      // Message with a good checksum received, dump it.
      Serial.println(value);
    }
}
```

Open the serial monitor on the receiver IDE and view the numbers

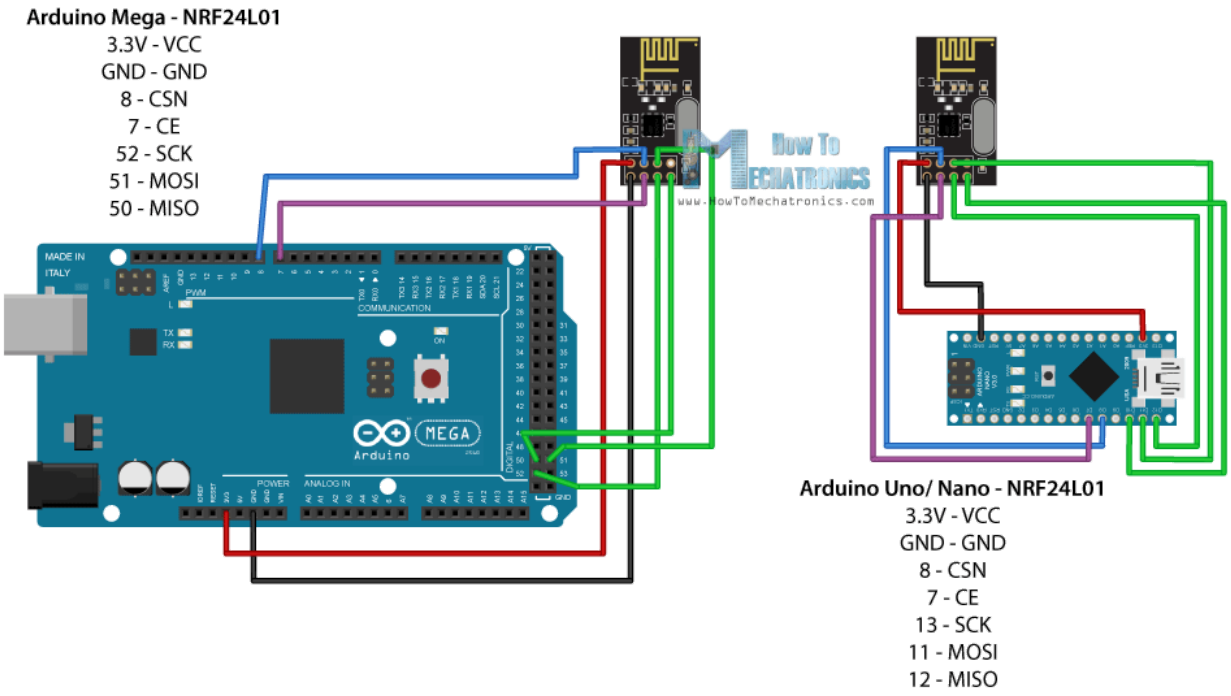


Note: in a rf noisy environment there can be garbage in the receive buffer. That is why only the first character of the buffer is used in this example.

2.4 Ghz. NRF24L01 transceiver

A slightly longer range and higher baud rate RF transmitter/receiver is the NRF24L01 transceiver. It is in the 2.4 Ghz range. The hardware connection is simple and a library handles all the messy bits.

https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/



(replace)
The RF24 library is needed. https://github.com/nRF24/RF24

Some nRF24L01 modules sell for about $4 and often come with an antenna which claims to increase the range to 800m, although that is under very good conditions. Usually 100 meter range is common.

XBEE

An expensive but well supported method for Arduinos to communicate wirelessly is the XBEE module. It is made by the *Digi Corporation*. The hardware comes with XCTU software to configure the systems. Low power modules cost about $27 and up to four are needed for two Arduinos

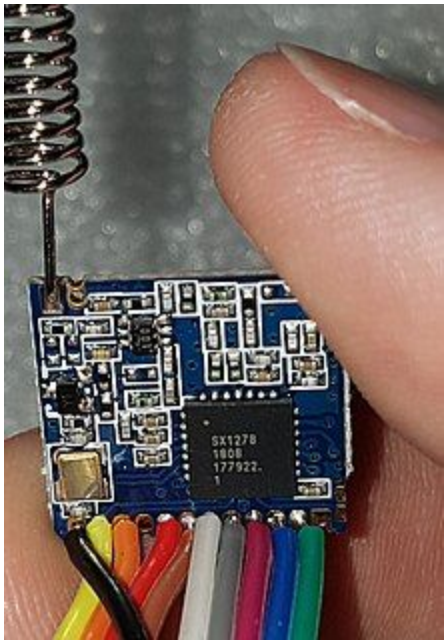
https://www.instructables.com/id/How-to-Use-XBee-Modules-As-Transmitter-Receiver-Ar/

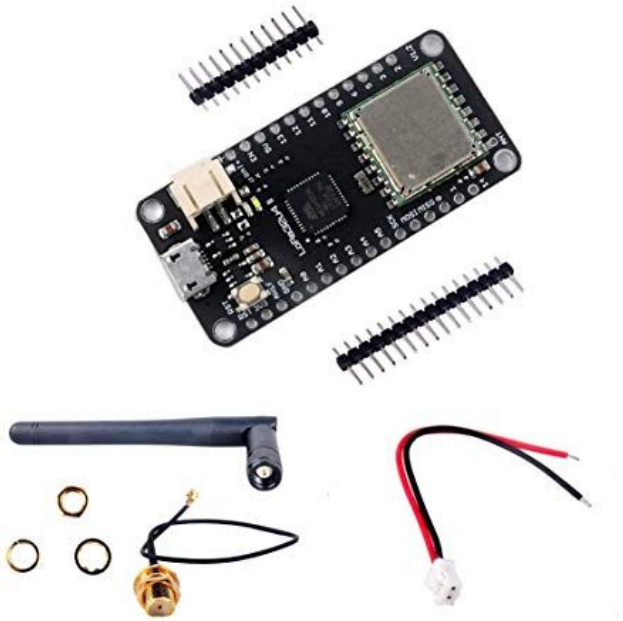
No example using the XBEE is done here due to the cost of the modules.

LoRa

For much longer communication ranges the LoRa technology has become popular. There are other systems (LPWAN) but they are less common. LoRa is a proprietary technology from Cycleo in France although the LoRa alliance is an open group.

LoRa is 1km with no antenna and can reach up to 100km with directional antennas and high power units. It uses the license free bands of 868 MHz (Europe) and 915 MHz (North America). Due to the low baud rates (below 50kb) Lo Ra is non-real time communication system. The LoRa transceiver modules cost around $25 on Amazon.



or


Adafruit has two LoRa modules for about $20. There are two frequencies offered, 433 mhz and 915 mhz LoRa modules. https://www.adafruit.com/product/3073, https://www.adafruit.com/product/3072

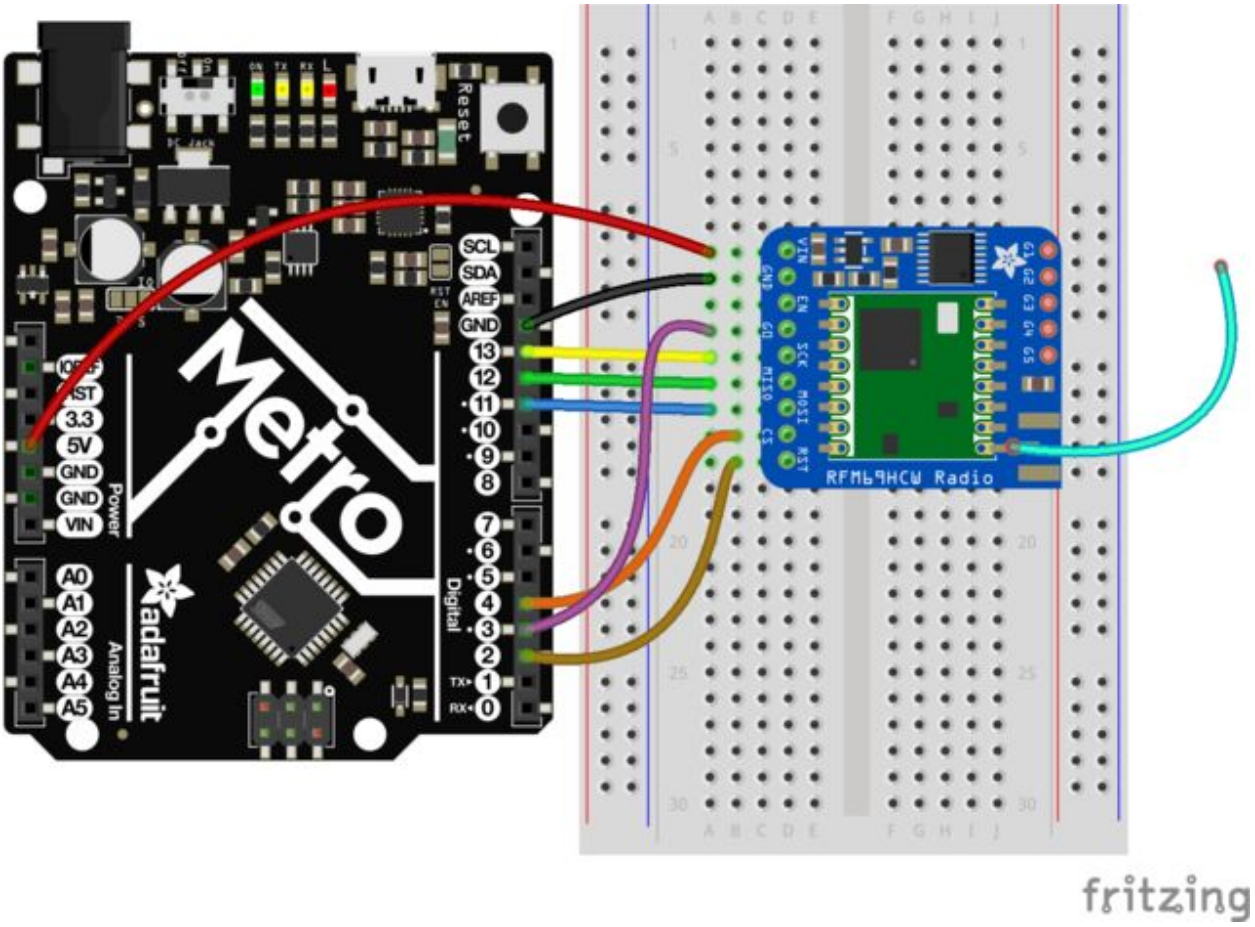
Adafruit has good tutorials and documentation. https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts/downloads

The LoRa modules are common enough that Arduino libraries have been developed, such as:

https://github.com/sandeepmistry/arduino-LoRa

Or two more common and powerful libraries for Arduinos:
LowPowerLab at https://github.com/LowPowerLab/RFM69
And RadioHead at http://www.airspayce.com/mikem/arduino/RadioHead/

Below is a Fritzing wiring diagram for the Metro Arduino Uno clone and the Adafruit LoRa module. The hardware is simple. The messy software communication is wrapped up in libraries.


Reset
SCL
SDA
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
TX
RX
Digital
IOREF
RST
3.3
5V
GND
GND
VIN
Power
A0
A1
A2
A3
A4
A5
Analog In
adafruit
Metro
RFM69HCW Radio
fritzing