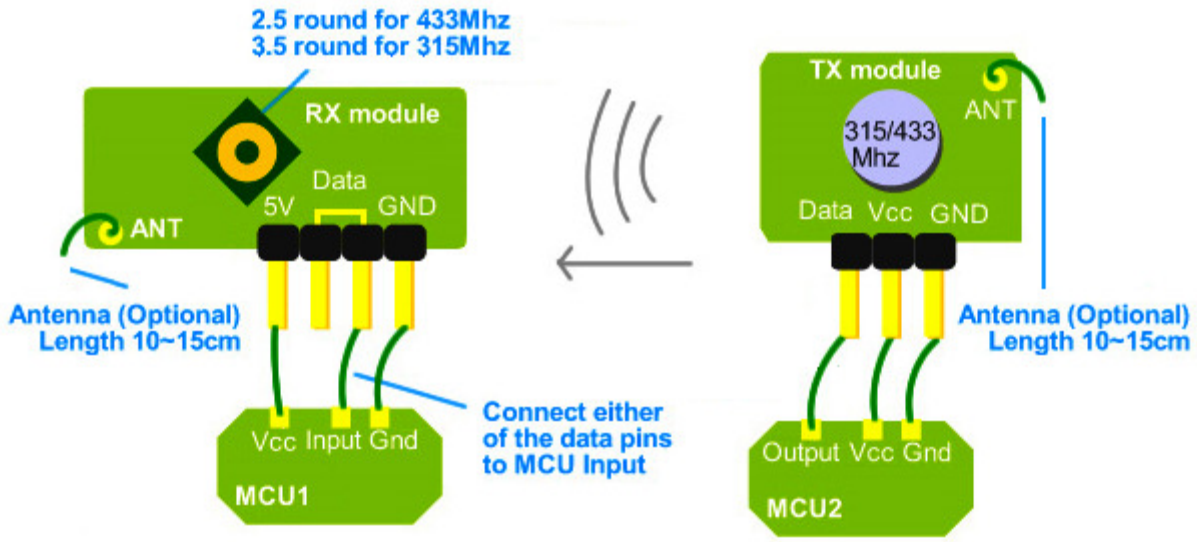
These modules provide a simple, straight-forward transmitter/receiver pairs for all of your low-cost wireless project needs.

These devices are simple pass-through integrated circuits. Meaning, you set up your baud-rate (as long as its within an acceptable range of whatever pair of devices you are using), and then start sending bytes to the transmitter. Quite simply, it just sends your data out the transmitter and the receiver grabs it, acting as if you had a wired serial connection between them, minus the wire!

- Frequency: 433.92 MHz.
- Modulation: ASK
- Receiver Data Output: High - 1/2 Vcc, Low - 0.7 V
- Transmitter Input Voltage: 3 - 12 V (high voltage = more transmitting power)
- Receiver Input Voltage : 4.5 - 5.5 V
- Transmitter working current: 20 - 28 mA
- Receiver static current:(mA): 4 mA
- Receiver sensitivity: -105 dBm
- Transmitter output power:16 dBm (40 mW)
- Transmitting range (work at 5 V): 40m indoor, and 100m in open air
- Transfer rate: <10 Kbps
- Size: Transmitter 19 mm x 19 mm x 8 mm, receiver 30 mm x 14 mm x 7 mm



Demostration scheme of 433/315Mhz RF kit

Tips and Considerations

Signal Strength (Transmitter Voltage and Antennas)

While the above establishes a wireless connection between two separate circuits, there are two considerations which will expand the range and efficiency of these two devices. First of all, each transmitter device has a supply voltage range of 3 – 12 volts. The higher the voltage, the stronger the signal strength. Commonly, your microcontroller and other circuitry may be running at 5 volts, so you may need to route a separate line of power, either from a non-voltage resisted incoming voltage, or a separate power supply sharing a common ground. Since these devices take very little amperage, consider using a 12 volt power supply, and running voltage directly from it to the transmitter, while isolating the other circuitry's voltage through a voltage regulator.

Also, the use of an optional antenna will increase the effectiveness of your wireless communication; a simple wire will do the trick. Experiment with different antenna shapes and configurations based on the shape and make of your project for best results.

Counters for Debugging

I found, when experiencing weird results with the pair, to implement a counter in my transmitter code as shown in the below examples. The benefit of this is it gives you a pattern you can follow visually, whereas an analog input or a combination of bits may be harder to troubleshoot. Especially when tuning the lower quality devices with a small screwdriver, its nice to be able to see a pattern of numbers scroll past in the serial window. Once you have successfully tuned the units and examined the data flow and protocol you want to send from device to device, remove the debugging serial-to-computer commands and you're ready to build up the code from there. *(note: In most occasions, I don't recommend tuning the devices, as they are generally tuned when shipped, but you may need to do so for one reason or another).*

Sample Code

On the following lines, I've provided two different sets of transmitter and receiver code for the microcontrollers you may be using in your project. One set is provided in Pic Basic Pro for the PIC microcontroller users, while the other is provided using the Arduino environment for AVR microcontrollers.

Pic Basic Pro

```
'*****************************************
'* Simple Transmitter (2400 baud)
'* For PIC18F452
'*****************************************
DEFINE OSC 20
counter var byte
counter = 0
OUTPUT PORTC.6
main:
' 16780 is 2400 non-inverted
' send out to transmitter
SEROUT2 PORTC.6,16780, [counter]
counter = counter + 1
PAUSE 10
goto main


'*****************************************
'* Simple Receiver (2400 baud)
'* For PIC18F452
'*****************************************
DEFINE OSC 20
inbyte var byte
OUTPUT PORTC.6
INPUT PORTD.1
main:
' 16780 is 2400 non-inverted
' Receive on Pin Port D1
SERIN2 PORTD.1,16780, [inbyte]
' debug to computer serial, just to see it
SEROUT2 PORTC.6, 16468, [inbyte]
' clear out inbyte. You see repeating zeros
' if you are losing signal
inbyte = 0
goto main
```

Arduino 0004

```
/*
```

```
* Simple Transmitter Code
* This code simply counts up to 255
* over and over
* (TX out of Arduino is Digital Pin 1)
*/
byte counter;
void setup(){
//2400 baud for the 434 model
Serial.begin(2400);
counter = 0;
}
void loop(){
//send out to transmitter
Serial.print(counter);
counter++;
delay(10);
}
```

```
/*
* Simple Receiver Code
* (TX out of Arduino is Digital Pin 1)
* (RX into Arduino is Digital Pin 0)
*/
int incomingByte = 0;
void setup(){
//2400 baud for the 434 model
Serial.begin(2400);
}
void loop(){
// read in values, debug to computer
if (Serial.available() > 0) {
incomingByte = Serial.read();
Serial.println(incomingByte, DEC);
}
incomingByte = 0;
}
```