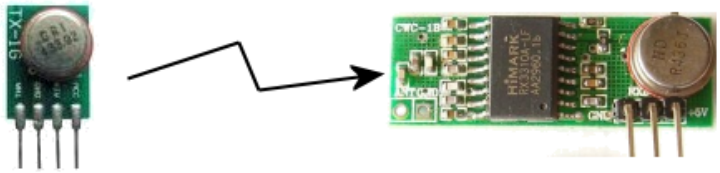
# Unidirectional Wireless RF Transmitting & Receiving



Digital in, digital out, with no wires!!! Yes, it can be that easy but there are some things you will find out about going wireless that you may not have counted with wired connection.
This application note will guide you through a simple RS232 link and illustrate how easy these devices are to use as long as you know how to handle a couple wireless phenomenon and plan your project to take these into account.

In Figure 1 below, we have illustrated the simplest RS232 link possible. It links a computer to a microcontroller using the 418/433MHz transmitter and the 418/433MHz receiver.
This circuit will work with most computer's RS232 ports because most use standard digital voltage levels to determine if a signal is a "1" or a "0".

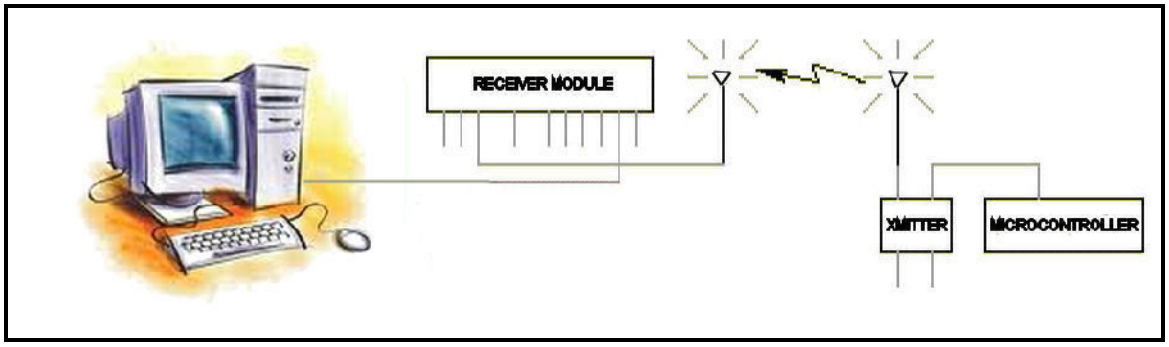



Figure 1: Simplest RF circuit (power & ground not shown).

In this example, we'll say the microcontroller is transmitting a packet of data with some sort of information. Something like:
<device ID><data byte 1><data byte 2><data byte 3><data byte 4><data byte 5>
Let's also put some ASCII readable data in the packet. Something like:

A12345

If a simple terminal program was used on the computer, one might see something like the following:

@#23lk 23 @# __(*& A12345 #@@23@#44

The information we are transmitting is seen but it is surrounded by garbage data. This garbage data is simply static the receiver is picking up, just as you pick up static on a television channel where there is no transmission. The transmitter only transmits the "1"s in your data packet. "0"s are simply periods of no transmission. The receiver is able to tell a "1" from a "0" when transmitted closely together but at other times, the receiver will try to make out data when there is none. Although this may seem counterproductive, it gives a much greater transmission distance since the receiver can constantly adjusts its gain, looking for data even when the signal is very weak.

One way of overcoming the static is to simply add a few unique bytes to the first of the packet that will identify good data to the program running on the computer. This is sometimes called a "header". For example, let's change our packet to be:

RFRFA12345

Now the computer would see something like:

@#23lk 23 @# __(*& RFRFA12345 #@@23@#44

It would be simple for a program to look for the "RFRF" pattern and then read in the device ID and data from within the packet.
This technique should work fine for many applications but we would also like to suggest using some sort of error detection and/or correction scheme that will ensure the integrity of the data. There are many different ways to go about doing this. The simplest is a checksum, where each byte of data is added together and the result is transmitted at the end of the packet. The receiver can then does the same addition, compare the result to the checksum, and have some confidence that the data was transmitted without interference. Remember, when using radio, there is always a possibility of interference from another radio source such as a garage door opener, nearby intercom, CB radio, etc….
A simple checksum for our example can be obtained by adding up the ASCII values for each byte in our packet. You can choose whether you wish to use the header in the calculation. If we leave out the header, our checksum is:

"A" + "1" + "2" + "3" + "4" + "5"
65 + 49 + 50 + 51 + 52 + 53 = 320

For this example, we will allocate only one byte at the end of the packet for the checksum information. A value of 320 "rolls-over" because one byte can only handle a value up to 255. The value after roll-over is the remainder of 320/256, or 64. 64 is the "at" symbol in the ASCII character set so our transmitted packet now looks like:

RFRFA12345@

This is our completed packet with a simple header and checksum.
Now, let’s have a quick review of how this packet is transmitted and received. Simply, the transmitter will preface every packet with "RFRF". This is done simply to allow the receiving device to recognize a packet of data from static. Then, the packet data is sent. In our example, this was a device id of "A" followed by the data bytes "12345". Finally, the transmitter sums up all the bytes from within the packet to get a checksum. In this example, the checksum was 320. Since we allocated only one byte to the checksum, we divided by 256 and took the remainder. The result was 64 or the "@" symbol in ASCII. The transmitter is constantly receiving data from the RF receiver. Initially, it looks for nothing but an "R". If it receives an "R", then it looks for an "FRF" as the next 3 characters it receives. If at any time it gets something different, it will go back to looking for the first "R". Once the header is received, it will load the packet data but not put it to use until receiving the checksum and comparing it with its own calculated checksum. If the packet received has a checksum matching the one transmitted, the data will be put to use.
The communication explained is unidirectional. While a bi-directional protocol can offer many added benefits, the unidirectional protocol described works fine when the transmitter retransmits its data and an occasional bad packet doesn’t greatly impact the operation of the system. This protocol offers means for avoiding the use of bad data while staying simple and inexpensive. It is a good protocol for use in applications such as wireless door sensors, motion sensors, and simple switches.