

# KLP / KLPA Module Walkthrough

Spark Fun Electronics - Jeff Gray

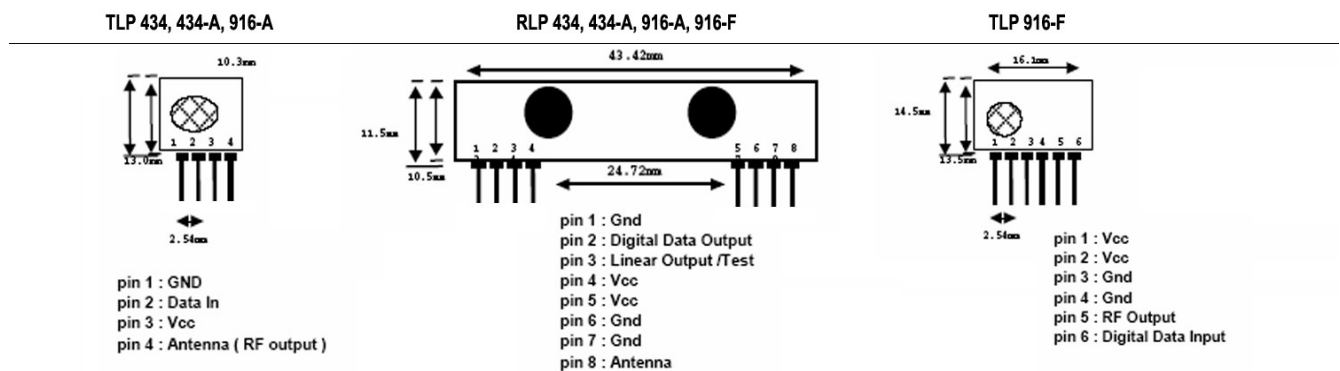
## Introduction

These laipac modules provide a simple, straight-forward transmitter/receiver pairs for all of your low-cost wireless project needs. These integrated circuits range in data-rates and electronic characteristics as outlined below.

Laipac Model	Spark Fun SKU	Frequency Range	Data Rate	Operating Voltage (Transmitter)	Operating Voltage (Receiver)
TLP/RLP 434	RF-KLP	315, 418, and 434 MHz	2,400 bps	2 - 12V	4.5 – 5.5 V
TLP/RLP 434-A	RF-KLPA	315, 418, and 434 MHz	4,800 bps	2 - 12V	3.3 – 6V
TLP/RLP 916-A	RF-KLP916-ASK	915 MHz	200,000 bps	2 - 12V	2.7 – 5 V
TLP/RLP 916-F	RF-KLP916-FSK	915 MHz	40,000 bps	2 - 12V	2.7 – 5.5V

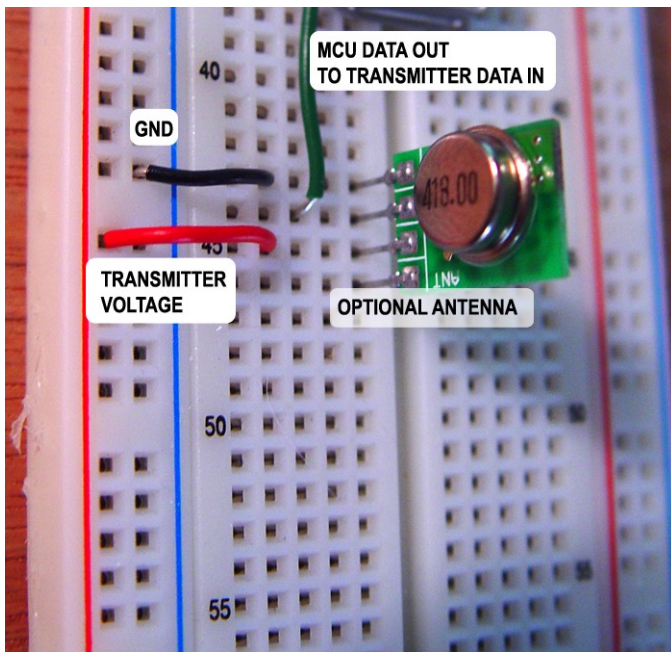
## Pin Diagrams and Wiring Information

Along with this information, here are the pin diagrams for the transmitter and receiver units. All four models of receivers use the same pin configuration, but the TLP 916-F transmitter is an odd man out, having extra pins to accommodate for its slightly larger board size.



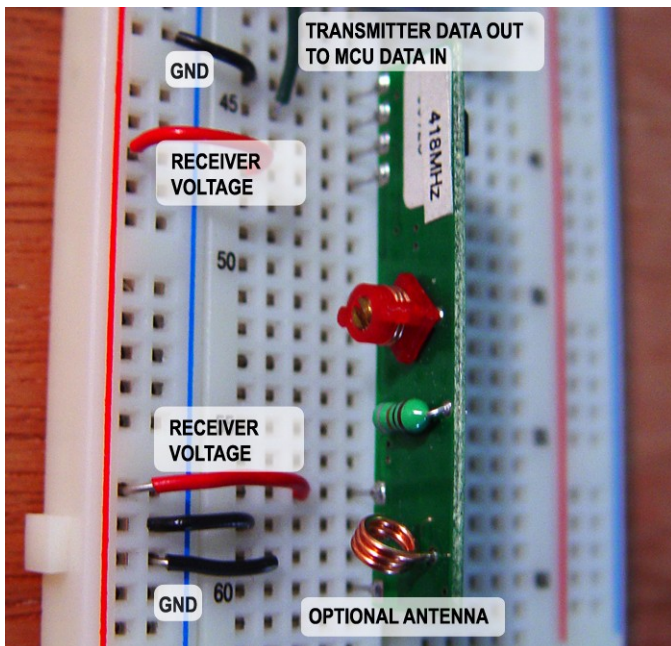
These devices are simple pass-through integrated circuits. Meaning, you set up your baud-rate (as long as its within an acceptable range of whatever pair of devices you are using), and then start sending bytes to the transmitter. Quite simply, it just sends your data out the transmitter and the receiver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

Based on these specifications, here's some sample images and code to get you started, followed by some things to consider while using these devices.



Three of the four transmitter modules have four pins, while the other has six. The six-legged version simply has redundant pins of power and ground, but otherwise remains the same.

Starting from the top, Pin 1 goes to ground. Pin 2 is the data input from the microcontroller of your choice. It transmits one byte at a time, at the given baud rate of the unit you're using. Pin 3 is the supply voltage, which will commonly be 5 volts. We'll explore other options later. Finally, Pin 4 is the optional antenna we'll look at later.



The receiver modules all share common pin assignments, as follows (from the top). Pin 1 goes to ground. Pin 2 sends the incoming byte from the transmitter out to the microcontroller. Pin 3 needs no connection, but can be taken to ground. Pins 4 and 5 go to the receiver voltage. Pins 6 and 7 go to ground, leaving Pin 8 as the optional antenna pin. Quite simply, that's all there is to it!

## Sample Code

On the following page, 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.

<i>Pic Basic Pro</i>	<i>Arduino 0004</i>
<pre> ***** /* 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 </pre>	<pre> /* * 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); } </pre>
<pre> ***** /* 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 </pre>	<pre> /* * 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() &gt; 0) {     incomingByte = Serial.read();     Serial.println(incomingByte, DEC);   }    incomingByte = 0; } </pre>

## 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 an supply voltage range of 2 – 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, but Spark Fun Electronics also sells antennas specifically designed for this as well. Experiment with different antenna shapes and configurations based on the shape and make of your project for best results.

- **Different Supply Voltages for Receivers**

As the chart at the beginning of this document shows, each receiver has different ranges for acceptable voltage. This is handy for use with 3.3 volt microcontrollers and low power ICs. Keep in mind that the RLP 434 only has a range of 4.5 – 5.5 volts, making it the exception in this case.

- **Counters for Debugging**

I found, when experiencing weird results with the pair, to implement a counter in my transmitter code as shown in the above 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).*

Best of luck in your wireless project!