# Introduction to the RFM69HW Transceiver

November 05, 2015 by Benjamin Crabtree (/author/benjamin-crabtree)

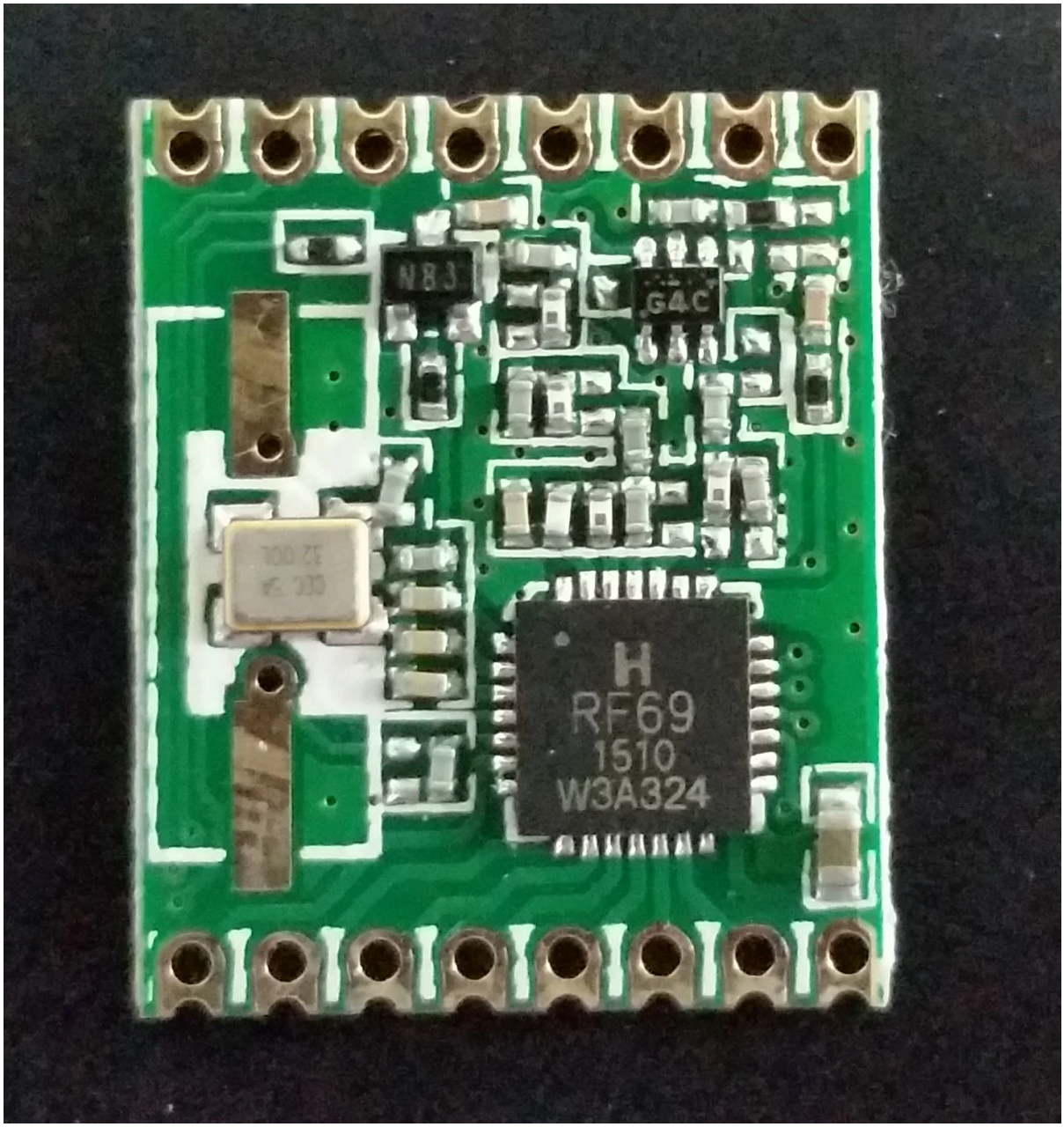
## RFM69HW Tranceivers

Engineers are designing more and more circuits that make use of some form of wireless communication to achieve a level of convenience and connectivity not feasible with wired options. In the designer's quest to go wireless, a few options present themselves which perform different features. 2.4GHz transceivers can send and receive large amounts of data and can operate with very small antennas, but they suffer from a comparatively short range with regard to lower frequency transceivers as well as a level of complexity and cost that might be overkill for many projects. For projects where low throughput is acceptable, the RFM69HW series transceivers provide a less complex solution which can interface with everything from the humble PIC up to a modern desktop PC and everything in between. The RFM transceiver can be purchased in 433MHz, 868MHz, or 915Mhz license-free ISM band for about $4, and can operate using FSK, GFSK, MSK, GMSK and OOK modulations. The power consumption is extremely low, yet these transceivers can communicate over several hundred meters given adequate antennae. These features make this device an excellent candidate for adding wireless connectivity to battery powered or remote projects for a very low cost investment. This project will be part one of a series introducing a range of applications for the RFM transceivers; in this case we will be setting up the transceivers to wirelessly enact the time-honored embedded systems equivalent of "Hello World!"- the "Blinky" program.

## What You Need

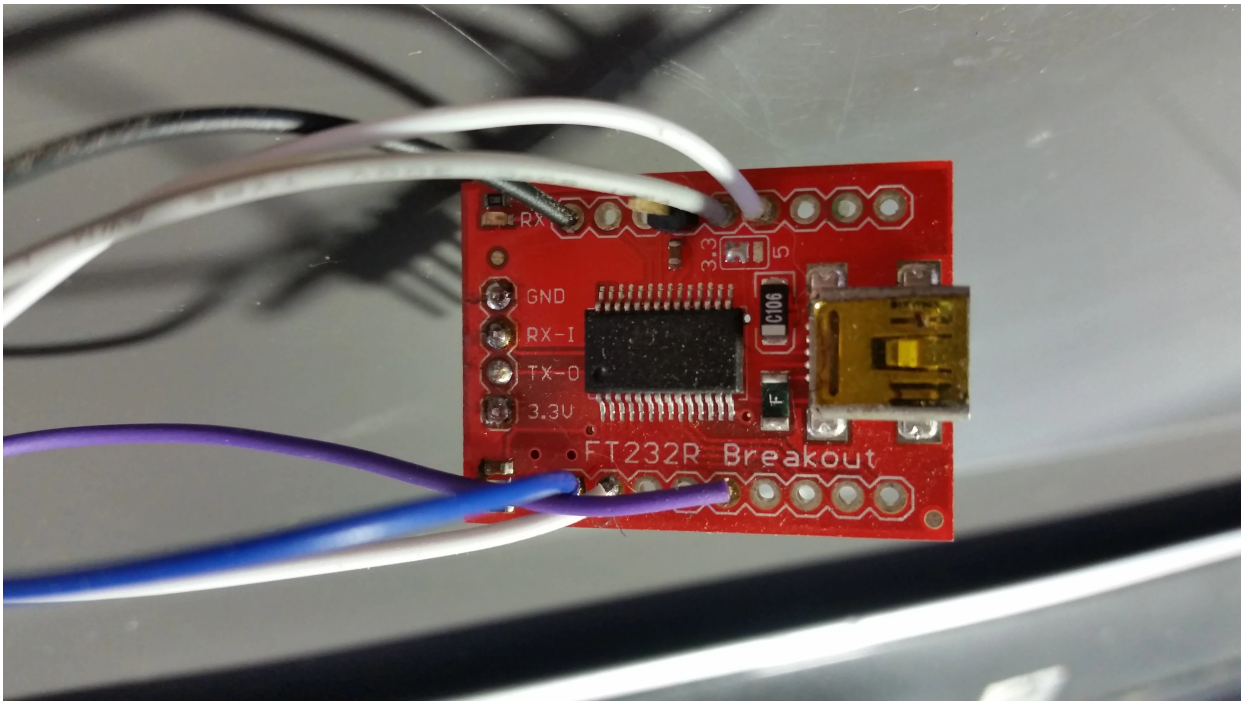
For teaching purposes, we will be implementing this code using Atmel328 (http://www.atmel.com/devices/atmega328p.aspx#buy) microprocessors running the Arduino bootloader. I happen to have a stack of 16Mhz Pro Mini V2 (http://www.digikey.com/product-detail/en/DEV-11113/1568-1055-ND/5140820?WT.mc_id=IQ_7595_G_pla5140820&wt.srch=1&wt.medium=cpc&WT.srch=1) ’s as seen below which will work after jumpering the 3.3 Volt selector, with Arduino you can use pretty much any board and get the same result- however the board you choose must be able to run on 3.3 Volts as the RFM chip can only operate on a maximum of 3.3 Volts. If you absolutely must use a 5 Volt Arduino you will need a logic level converter (https://www.adafruit.com/products/735?gclid=Cj0KEQiA0-GxBRDWsePx0pPtp4sBEiQACuTLNnc58W1LNIAKLkU3vTSc9IyrnWLzgD1VedSAcLOMQncaAqTO8P8HAQ).


You also need the RFM chip itself, with the 915MHz 69HW (http://www.hoperf.com/rf/fsk_module/RFM69HW.htm) model pictured below.




It is not necessary to use this identical model; you may choose other models or frequencies, but you will need to tweak the code to accommodate this. The code will be commented in these places for ease of modification.

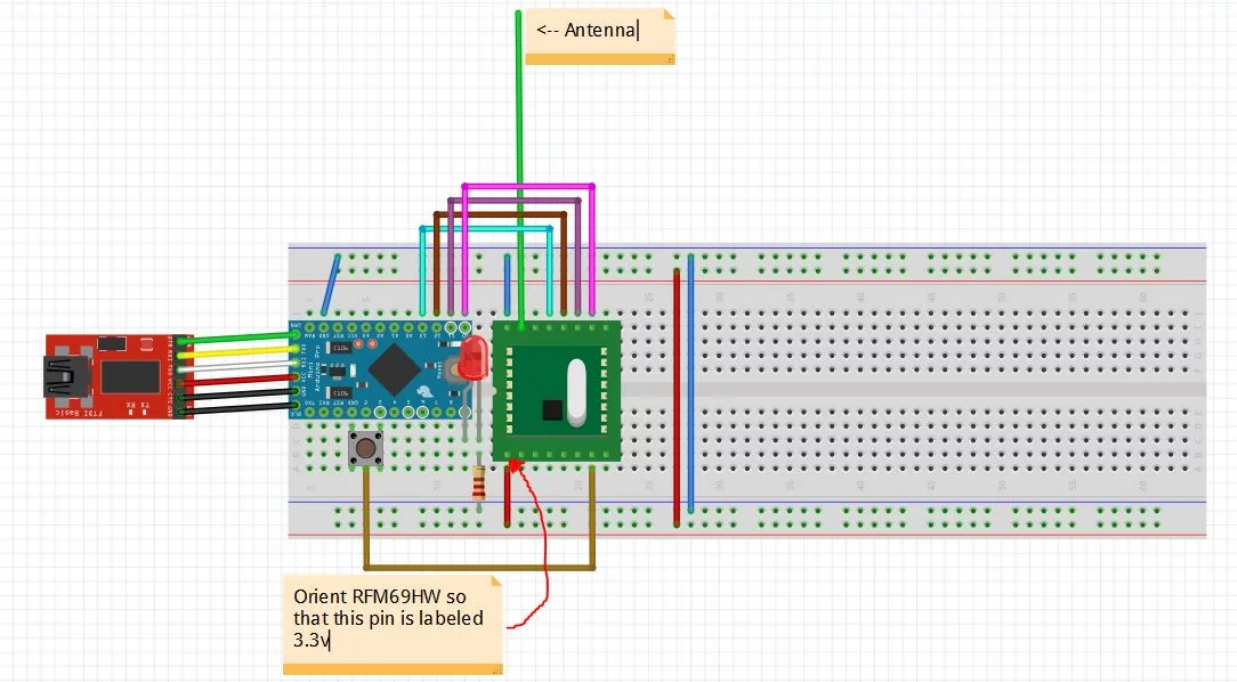
You will also need a method of programming and powering the boards; if you are using an UNO or other Arduino with built-in USB then you have this requirement covered. In my case I need to use FTDI serial breakout boards, as the Pro Mini's are only populated with the processor itself and a few housekeeping components.



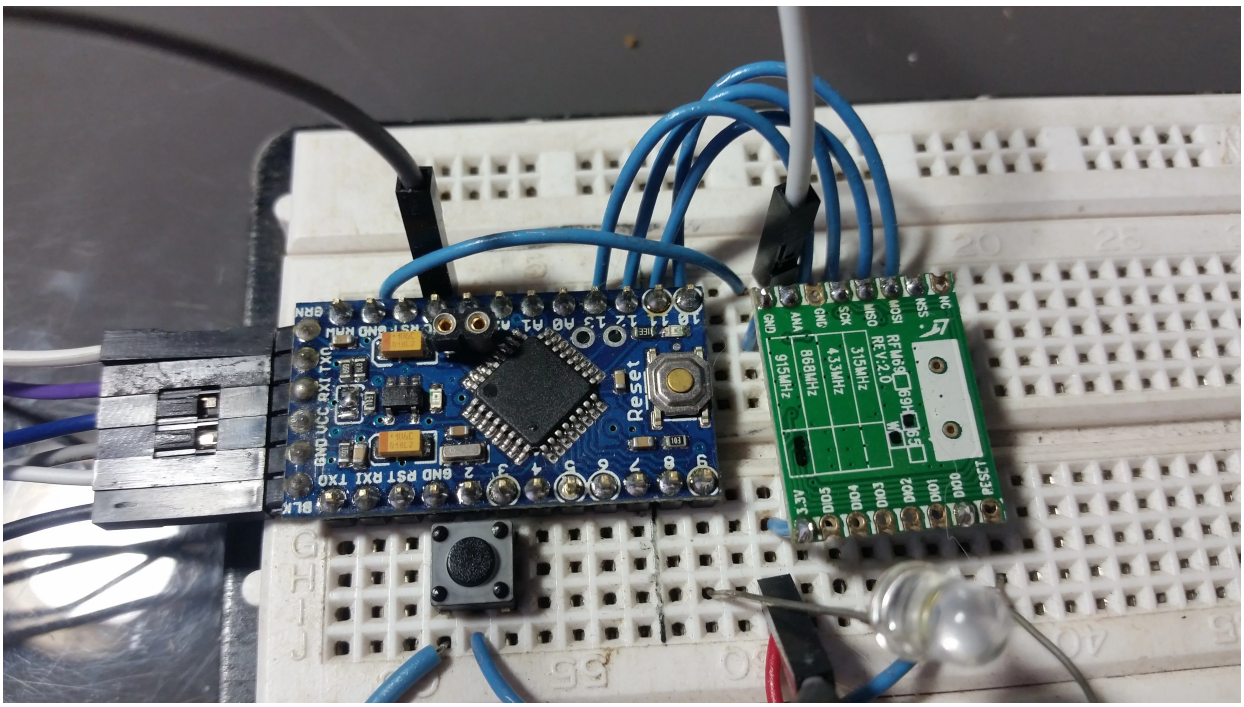

In addition to these main components, you will of course need an LED, a resistor (for LED ballast- choose accordingly), a momentary push button, jumper wire, and prototyping boards.

# Setup

Now that we have all the parts, we can assemble the circuit. Below is the wiring diagram:




The momentary button should be bridging pin 3 and ground, the LED should be attached to pin 9, and the data pins NSS, MOSI, MISO, and SCK should connect to pins 10, 11, 12, 13 in that order. The antenna can be a real antenna, or just a piece of un-terminated wire. Make two of this identical circuit and either device can be set up as the sender or receiver. Your circuits should look something like this:



# Run The Code

With circuits in hand, it’s time to do some programming. You will need two libraries written to run the RFM chips on an Arduino, RFM69.h and LowPower.h; they can be obtained from https://www.github.com/lowpowerlab/rfm69 (https://www.github.com/lowpowerlab/rfm69) and https://github.com/lowpowerlab/lowpower (https://github.com/lowpowerlab/lowpower) respectively. Install these libraries and copy the following code into your IDE:

```
//RFM69HW Blinky Code
//This program is a highly simplified version of the program by Felix from LowPow
erLab
//A button is pushed on the sender unit which toggles an LED on the receiver
#include     //get it here: https://www.github.com/lowpowerlab/rfm69 (https://ww
w.github.com/lowpowerlab/rfm69)
#include
#include  //get library from: https://github.com/lowpowerlab/lowpower (https://gi
thub.com/lowpowerlab/lowpower)

//*****************************************************************************
**************
// ********** IMPORTANT SETTINGS - YOU MUST CHANGE/ONFIGURE TO FIT YOUR HARDWARE
 *************
//*****************************************************************************
**************
//This part of the code simply sets up the parameters we want the chip to use
// these parameters allow you to have multiple networks, channels, and encryption
 keys
#define NETWORKID     100  //the same on all nodes that talk to each other
#define RECEIVER      1    //unique ID of the gateway/receiver
#define SENDER        2    // you could for example, have multiple senders
#define NODEID        RECEIVER  //change to "SENDER" if this is the sender node
 (the one with the button)
//Select your frequency by uncommenting
//#define FREQUENCY     RF69_433MHZ
//#define FREQUENCY     RF69_868MHZ
#define FREQUENCY     RF69_915MHZ
#define ENCRYPTKEY    "sampleEncryptKey" //exactly the same 16 characters/bytes o
n all nodes!
#define IS_RFM69HW    //uncomment only for RFM69HW! Remove/comment if you have RF
M69W!
//*****************************************************************************
**************
#define SERIAL_BAUD   9600

//This part defines the LED pin and button pin
#define LED             9 //LED on D9
#define BUTTON_INT      1 //user button on interrupt 1 (D3)
#define BUTTON_PIN      3 //user button on interrupt 1 (D3)
#define RX_TOGGLE_PIN   7 //GPIO to toggle on the RECEIVER

RFM69 radio;

// the setup contains the start-up procedure and some useful serial data
void setup() {
  Serial.begin(SERIAL_BAUD);
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
```

```
#ifdef IS_RFM69HW
  radio.setHighPower(); //only for RFM69HW!
#endif
  radio.encrypt(ENCRYPTKEY);
  char buff[50];
  sprintf(buff, "\nListening at %d Mhz...", FREQUENCY==RF69_433MHZ ? 433 : FREQUE
NCY==RF69_868MHZ ? 868 : 915);
  Serial.println(buff);
  Serial.flush();
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  attachInterrupt(BUTTON_INT, handleButton, FALLING);

  pinMode(RX_TOGGLE_PIN, OUTPUT);

}

//******** THIS IS INTERRUPT BASED DEBOUNCING FOR BUTTON ATTACHED TO D3 (INTERRUP
T 1)
#define FLAG_INTERRUPT 0x01
volatile int mainEventFlags = 0;
boolean buttonPressed = false;
void handleButton()
{
  mainEventFlags |= FLAG_INTERRUPT;
}

byte LEDSTATE=LOW; //LOW=0
void loop() {
  //******** THIS IS INTERRUPT BASED DEBOUNCING FOR BUTTON ATTACHED TO D3 (INTERR
UPT 1)
  if (mainEventFlags & FLAG_INTERRUPT)
  {
    LowPower.powerDown(SLEEP_120MS, ADC_OFF, BOD_ON);
    mainEventFlags &= ~FLAG_INTERRUPT;
    if (!digitalRead(BUTTON_PIN)) {
      buttonPressed=true;
    }
  }

  if (buttonPressed)
  {
    Serial.println("Button pressed!");
    buttonPressed = false;

    if (radio.sendWithRetry(RECEIVER, "All About Circuits", 18)) //target node I
d, message as string or byte array, message length
      delay(100);

  }
```

```
  //check if something was received (could be an interrupt from the radio)
  if (radio.receiveDone())
  {
    //print message received to serial
    Serial.print('[');Serial.print(radio.SENDERID);Serial.print("] ");
    Serial.print((char*)radio.DATA);
    Serial.print("   [RX_RSSI:");Serial.print(radio.RSSI);Serial.print("]");
    Serial.println();


    if(LEDSTATE==LOW)
      LEDSTATE=HIGH;
    else LEDSTATE=LOW;
    digitalWrite(LED, LEDSTATE);
    digitalWrite(RX_TOGGLE_PIN, LEDSTATE);


    //check if sender wanted an ACK
    if (radio.ACKRequested())
    {
      radio.sendACK();
      Serial.print(" - ACK sent");
    }
  }

  radio.receiveDone(); //put radio in RX mode
  Serial.flush(); //make sure all serial data is clocked out before sleeping the
 MCU
  LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_ON); //sleep Arduino in low power mod
e (to save battery)
}
```

**Download Code (/login/?redirect=https//www.allaboutcircuits.com/projects/intr**

The code is commented to describe its operation, but here is a basic breakdown of what is going on:

1. Include the libraries

These are the libraries linked to earlier, in addition to the standard SPI.h library for serial communication.

### 2. Define the constants

Tell the chip who it is, what network it is on, what its encryption code is, whether it is the sender or receiver, which of a possible set of receivers/senders the unit is, etc.

### 3. Initialize the radio

This is the housekeeping step where all those parameters we just defined get applied, in addition to setting up the serial monitor for debugging purposes.

### 4. Go to sleep and wait for an interrupt

Using an interrupt on the button and putting the unit into sleep helps to save power while the unit is idling. Interrupts are a highly efficient tool to wake a device upon an event, and allow a device to do other things in the meantime before and after the interrupt takes place.

### 5. Transmit upon an interrupt flag

This will send a message to the receiver after the sender's button is pushed. In this case the message is "All About Circuits."

### 6. Receive

The receiver checks and confirms that it has received a message. It then prints the message to serial, and also prints the RSSI (Received Signal Strength Indicator) to let you know how strong the signal was in dBm. The receiver then toggles the LED to let the user know a message was received.

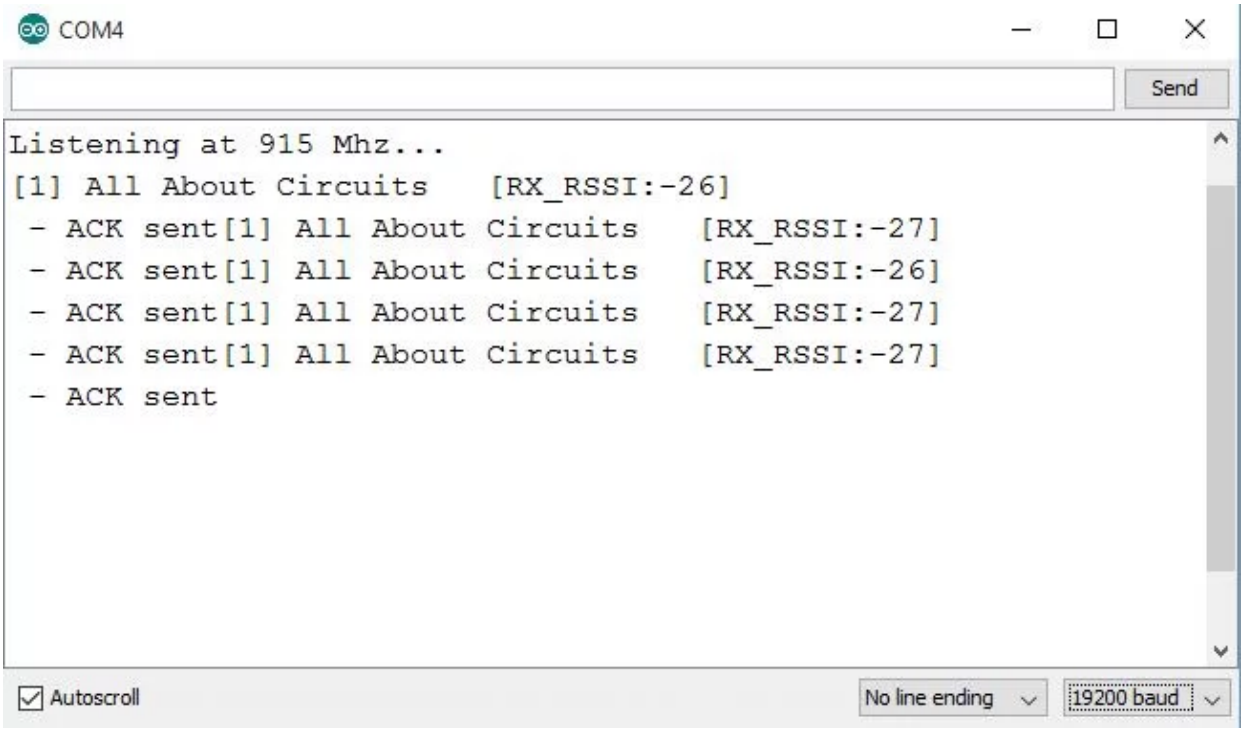
### 7. Acknowledge

Send an ACK bit back to the sender to let it know that the message was received.

### 8. Sleep

Put the device back into a low power sleep mode to conserve battery and wait for another interrupt.

If you hook up your receiving unit to your computer to use the serial monitor terminal you should see this after pressing the button a few times:




In addition, the LED should blink on and off as you press the button. Readers with sharp eyes may have noticed something interesting about this serial monitor box: the baud rate is set to 19200 while the baud rate in the code is set to 9600. This is because the Pro Mini I am using is running at 16MHz with