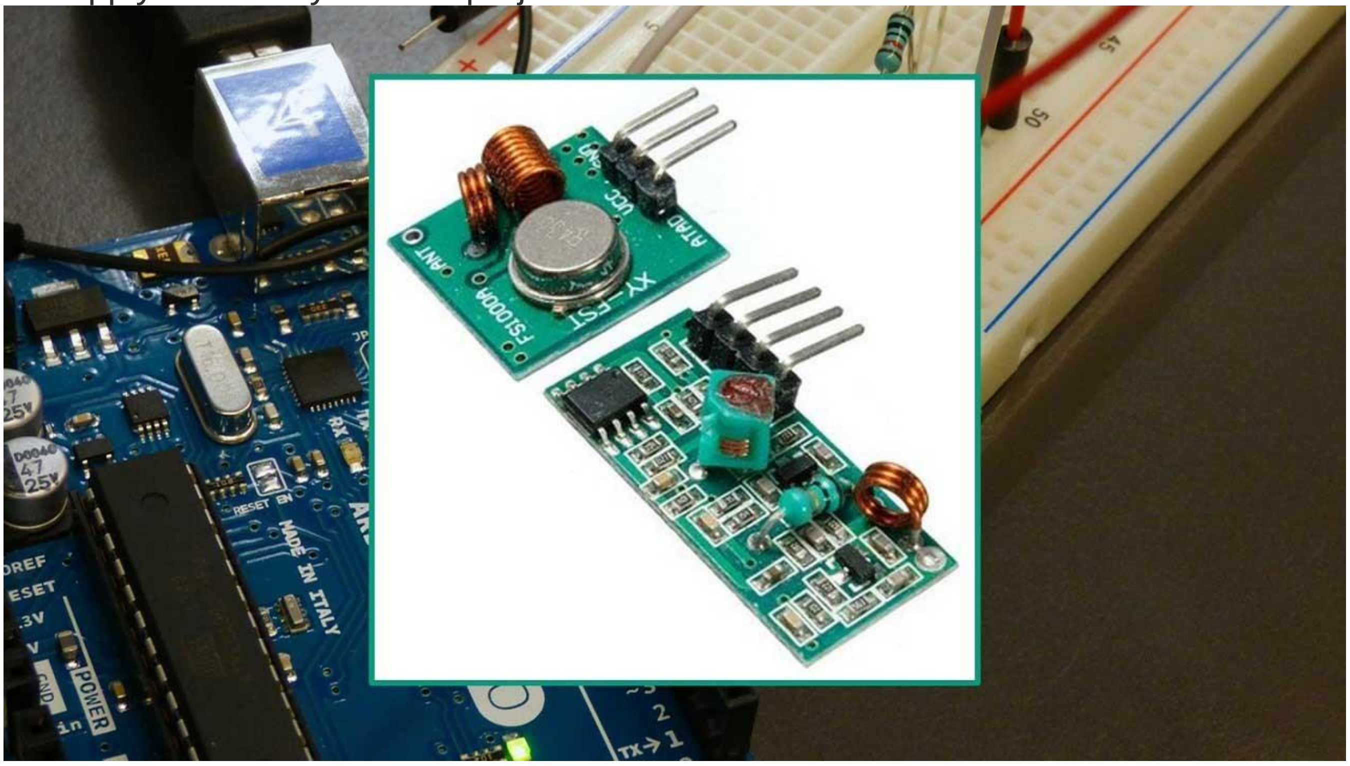
Complete Guide for RF 433MHz Transmitter/Receiver Module With Arduino

This post is a guide for the popular RF 433MHz Transmitter/Receiver modules with Arduino. We'll explain how they work and share an Arduino project example that you can apply to use in your own projects.



Description

Throughout this tutorial we'll be using the FS1000A transmitter and corresponding receiver, but the instructions provided also work with other 433MHz transmitter/receiver modules that work in a similar fashion. These RF modules are very popular among the Arduino tinkerers and are used on a wide variety of applications that require wireless control. These modules are very cheap and you can use them with any microcontroller (MCU), whether it's an Arduino, ESP8266, or ESP32.

Specifications RF 433MHz Receiver

- Frequency Range: 433.92 MHz
- Modulation: ASK

- Input Voltage: 5V
- Price: $1 to $2

Specifications RF 433MHz Transmitter

- Frequency Range: 433.92MHz
- Input Voltage: 3-12V
- Price: $1 to $2

Arduino with RF 433MHz Transmitter/Receiver Modules

In this section we build a simple example that sends a message from an Arduino to another using 433 MHz. An Arduino board will be connected to a 433 MHz transmitter and will send the "Hello World!" message. The other Arduino board will be connected to a 433 MHz receiver to receive the messages.

You need the following components for this example:

- 2x Arduino – read Best Arduino Starter Kits
- RF 433MHz Receiver/Transmitter
- Breadboard
- Jumper wires

You can use the preceding links or go directly to MakerAdvisor.com/tools to find all the parts for your projects at the best price!

Installing the RadioHead Library

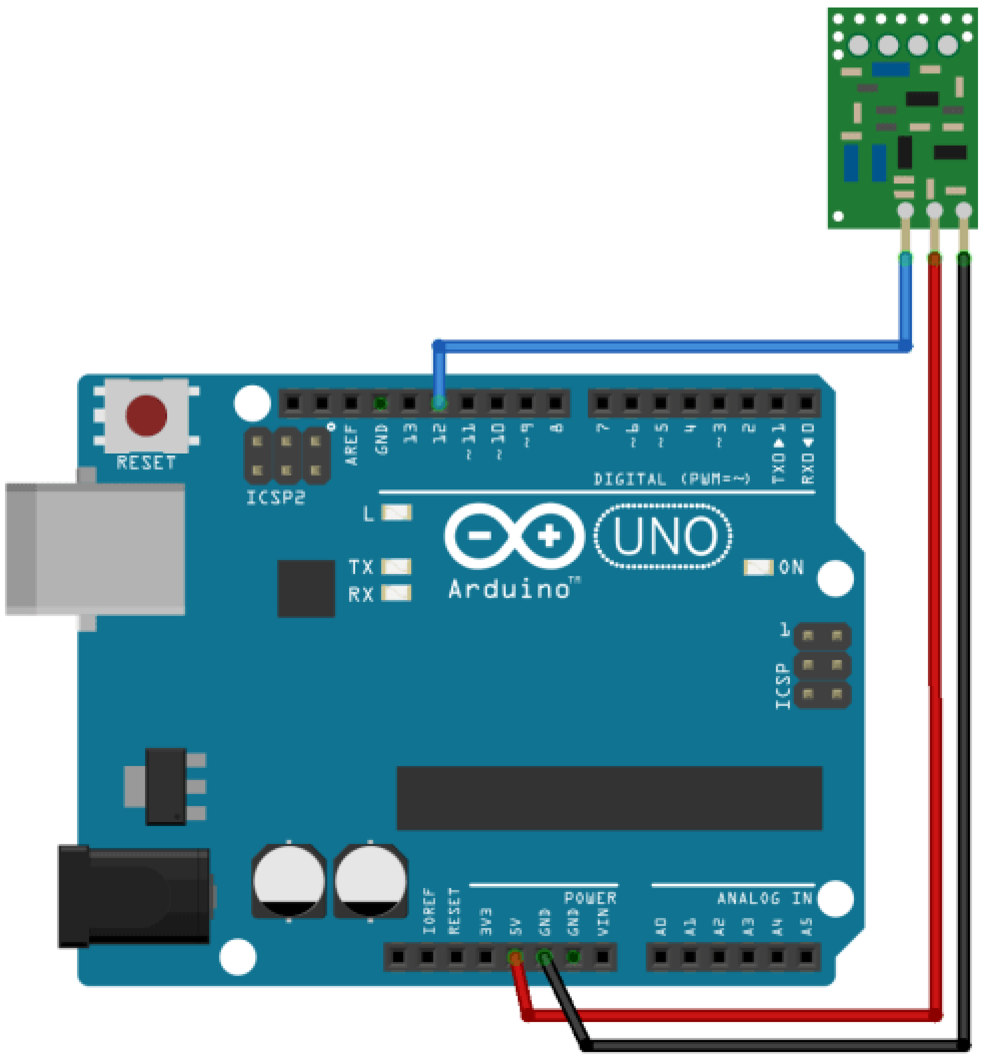
The RadioHead library provides an easy way to work with the 433 MHz transmitter/receiver with the Arduino. Follow the next steps to install that library in the Arduino IDE:

1. Click here to download the RadioHead library. You should have a .zip folder in your **Downloads** folder.
2. Unzip the **RadioHead** library.
3. Move the **RadioHead** library folder to the Arduino IDE installation **libraries** folder.
4. Restart your Arduino IDE

The RadioHead library is great and it works with almost all RF modules in the market. You can read more about the RadioHead library here.

Transmitter Circuit

Wire the transmitter module to the Arduino by following the next schematic diagram.



Note: always check the pinout for the transmitter module you're using. Usually, there are labels next to the pins. Alternatively, you can also take a look at your module's datasheet.

Transmitter Sketch

Upload the following code to the Arduino board.

```
#include
#include  // Not actually used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600);       // Debugging only
    if (!driver.init())
         Serial.println("init failed");
}

void loop()
{
    const char *msg = "Hello World!";
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(1000);
}
```

How the transmitter sketch works

First, we need to include the *RadioHead ASK* library.

```
#include <RH_ASK.h>
```

This library needs the SPI library to work. So, we also need to include the SPI library.

```
#include <SPI.h>
```

After that, we create a *RH_ASK* object called *driver*.
In the *setup()* we initialize the *RH_ASK* object by using the *init()* method. We use an *if*statement and a print in the Serial Monitor for debugging purposes.

```
Serial.begin(9600); // Debugging only

if (!driver.init())

    Serial.println("init failed");
```

In the *loop()*, we write and send our message. Here, our message is saved on the *msg*variable. Please note that **the message needs to be a char type**.

```
const char *msg = "Hello World!";
```

This message contains the “Hello World!” message, but you can send anything you want as long as it is in char format.

Finally, we send our message as follows:

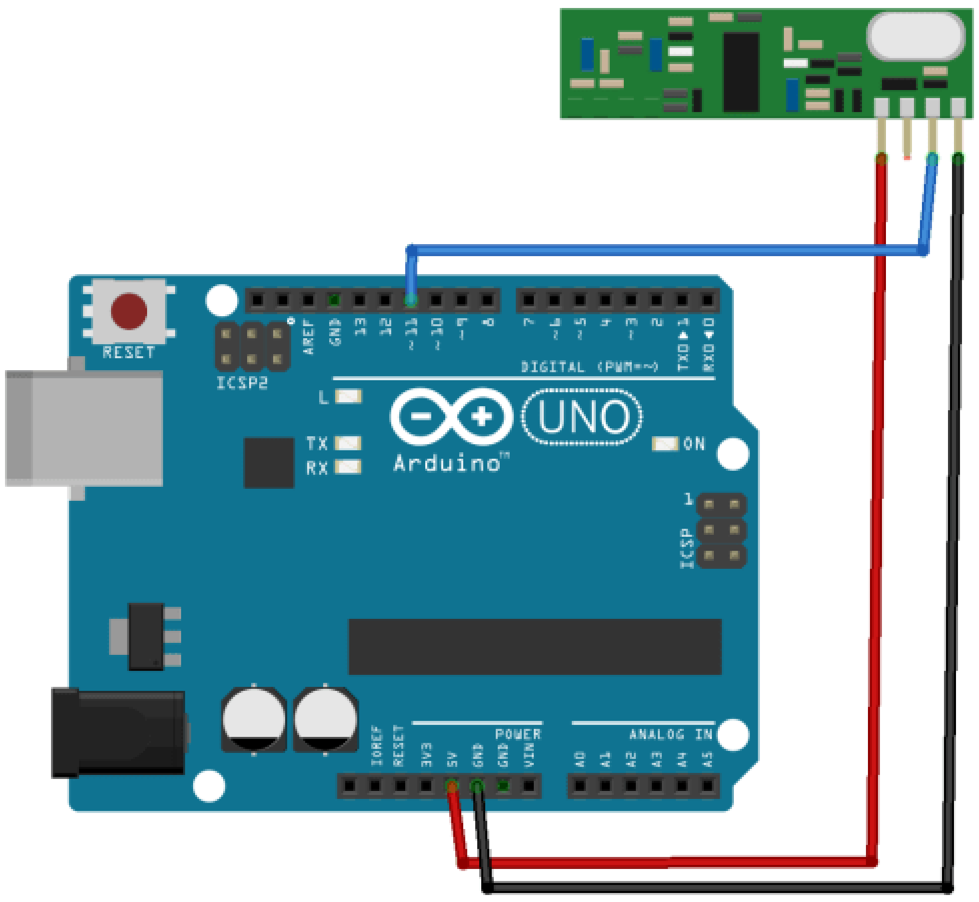
```
driver.send((uint8_t *)msg, strlen(msg));

driver.waitPacketSent();
```

The message is being sent every second, but you can adjust this delay time.

```
delay(1000);
```

Receiver Circuit

Wire the receiver module to another Arduino by following the next schematic diagram.



Note: always check the pinout for the receiver module you're using. Usually, there are labels next to the pins. Alternatively, you can also take a look at your module's datasheet.

Receiver Sketch

Upload the code below to the Arduino connected to the receiver.

```
#include
#include  // Not actualy used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600);     // Debugging only
    if (!driver.init())
         Serial.println("init failed");
}

void loop()
{
    uint8_t buf[12];
    uint8_t buflen = sizeof(buf);
    if (driver.recv(buf, &buflen)) // Non-blocking
    {
      int i;
      // Message with a good checksum received, dump it.
      Serial.print("Message: ");
      Serial.println((char*)buf);
    }
}
```

How the receiver sketch works

Similarly to the previous sketch, you start by including the necessary libraries:

```
#include <RH_ASK.h>

#include <SPI.h>
```

You create a *RH_ASK* object called *driver*:

```
RH_ASK driver;
```

In the *setup()*, we initialize the *RH_ASK* object.

```
void setup(){

    Serial.begin(9600); // Debugging only

    if (!driver.init())

    Serial.println("init failed");

}
```

In the *loop()*, we need to set a buffer that matches the size of the message we'll receive. "Hello World!" has 12 characters. You should adjust the buffer size accordingly to the message you'll receive (spaces and punctuation also count).

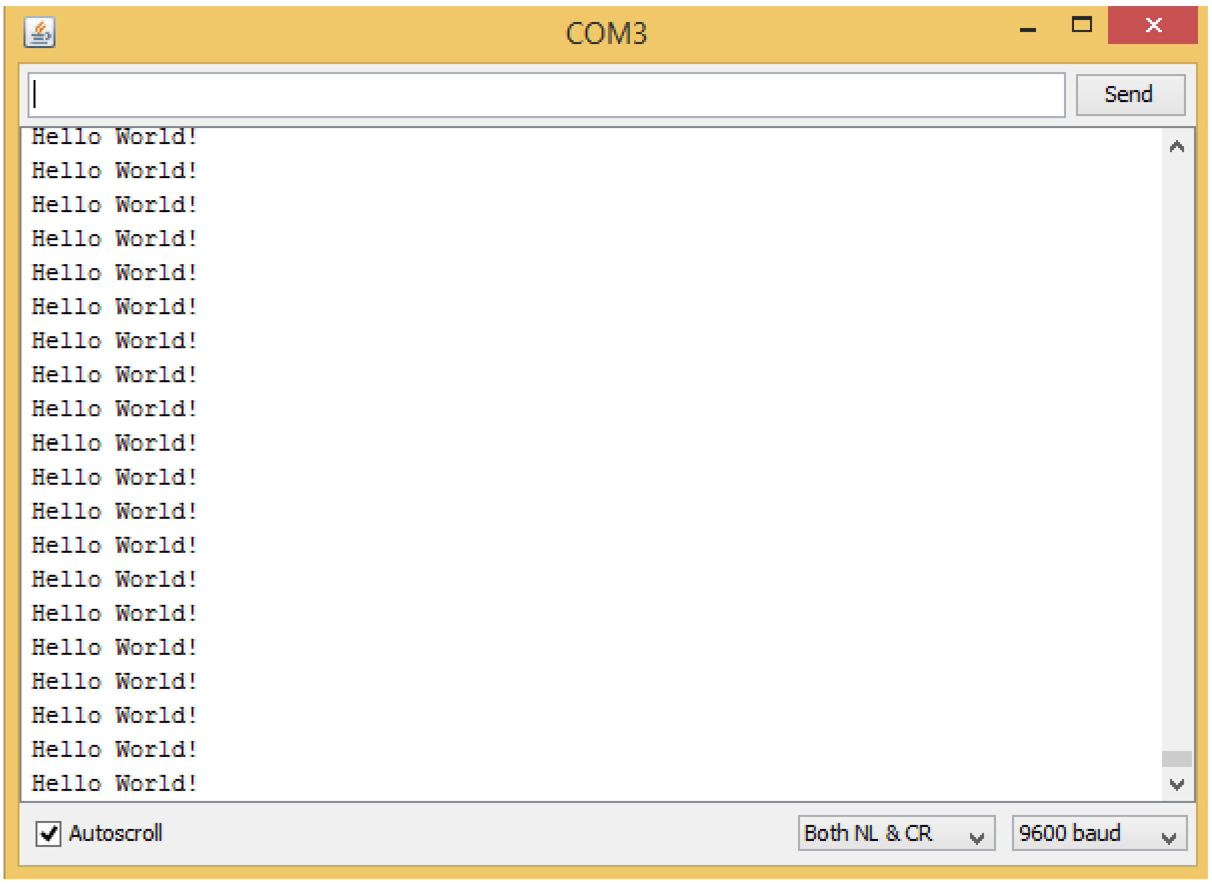
```
uint8_t buf[12];

uint8_t buflen = sizeof(buf);
```

Then, we check if we received a valid message. If we receive a valid message, we print it in the serial monitor.

```
if (driver.recv(buf, &buflen)) {

    int i;

    // Message with a good checksum received, dump it.

    Serial.print("Message: ");

    Serial.println((char*)buf);

}
```

Demonstration

In this project the transmitter is sending a message “Hello World!” to the receiver via RF. Those messages are being displayed in receiver’s serial monitor. The following figure shows what you should see in your Arduino IDE serial monitor.



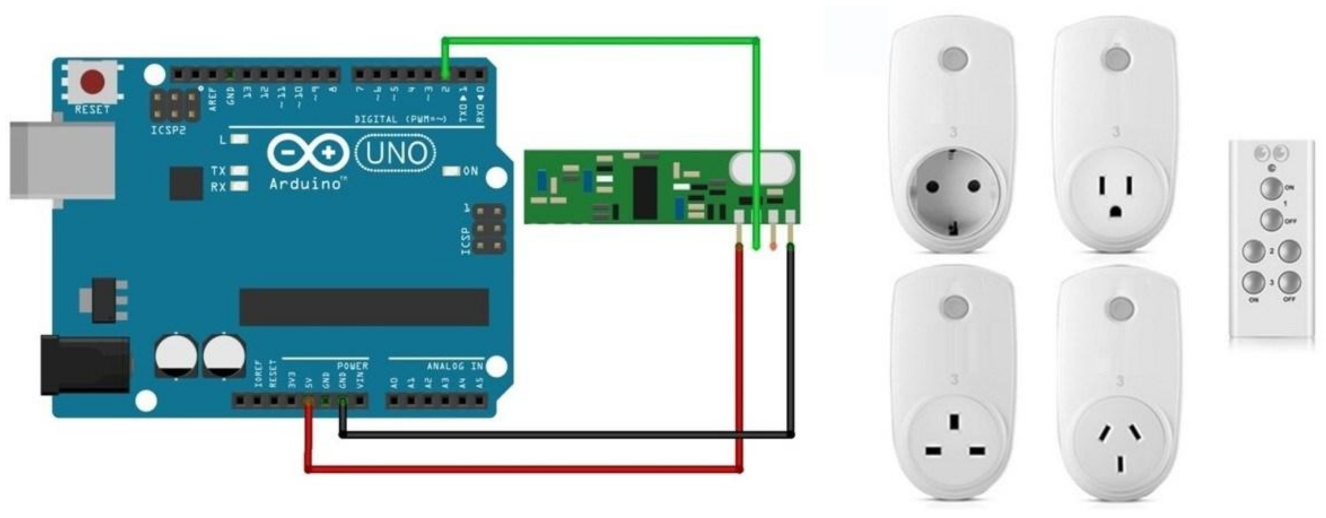
```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Wrapping Up

You need to have some realistic expectations when using this module. They work very well when the receiver and transmitter are close to each other. If you separate them too far you’ll lose the communication. The communication range will vary. It depends on how much voltage you’re supplying to your transmitter module, RF noise in your environment, and if you’re using an external antenna.

Decode and Send 433 MHz RF Signals with Arduino

This guide shows how to use an Arduino to decode 433 MHz signals from RF remotes, and send them with an Arduino and a 433 MHz transmitter to remotely control mains switches outlets.



Why Decoding RF Signals?

I've tried different methods of controlling the mains voltage, but some of the methods require:

- Experience dealing with AC voltage
- Opening holes in your wall/ceiling/switches
- Modifying the electrical panel
- Knowing the electrical rules for each country

It's difficult to come up with a solution that is safe and works for everyone. One of the easiest and safest ways to remotely control appliances connected to mains voltage is using radio frequency (RF) controlled outlets. Why? Using remote controlled outlets have 5 benefits:

1. Fairly inexpensive
2. Easy to get
3. Works with ESP8266 and Arduino

4. Safe to use
5. Works in any country

Parts Required

For this tutorial, you need the following parts:

- Arduino UNO – read Best Arduino Starter Kits
- 433 MHz RF Remote controlled sockets
- 433 MHz transmitter/receiver
- Breadboard
- Jumper wires

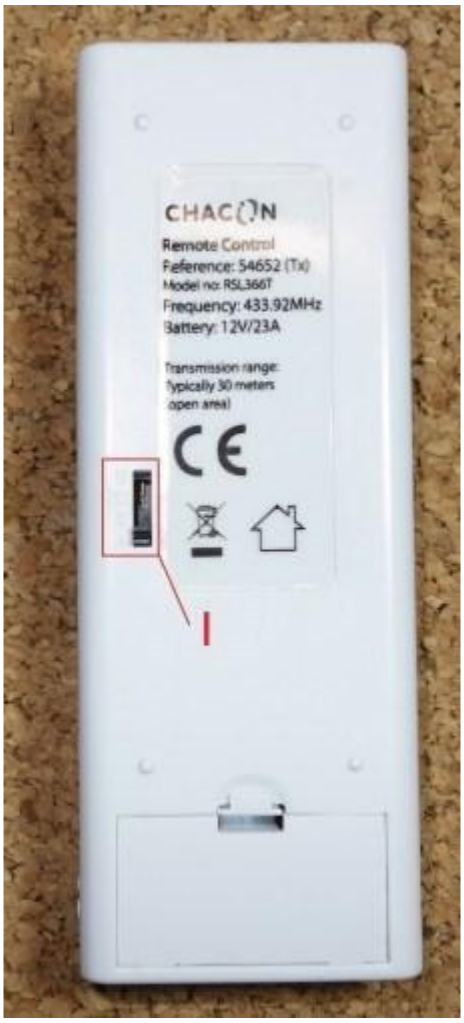
Note: you need to buy remote controlled outlets that operate at a RF of 433MHz. They should say the operating RF either in the product page or in the label.

Example

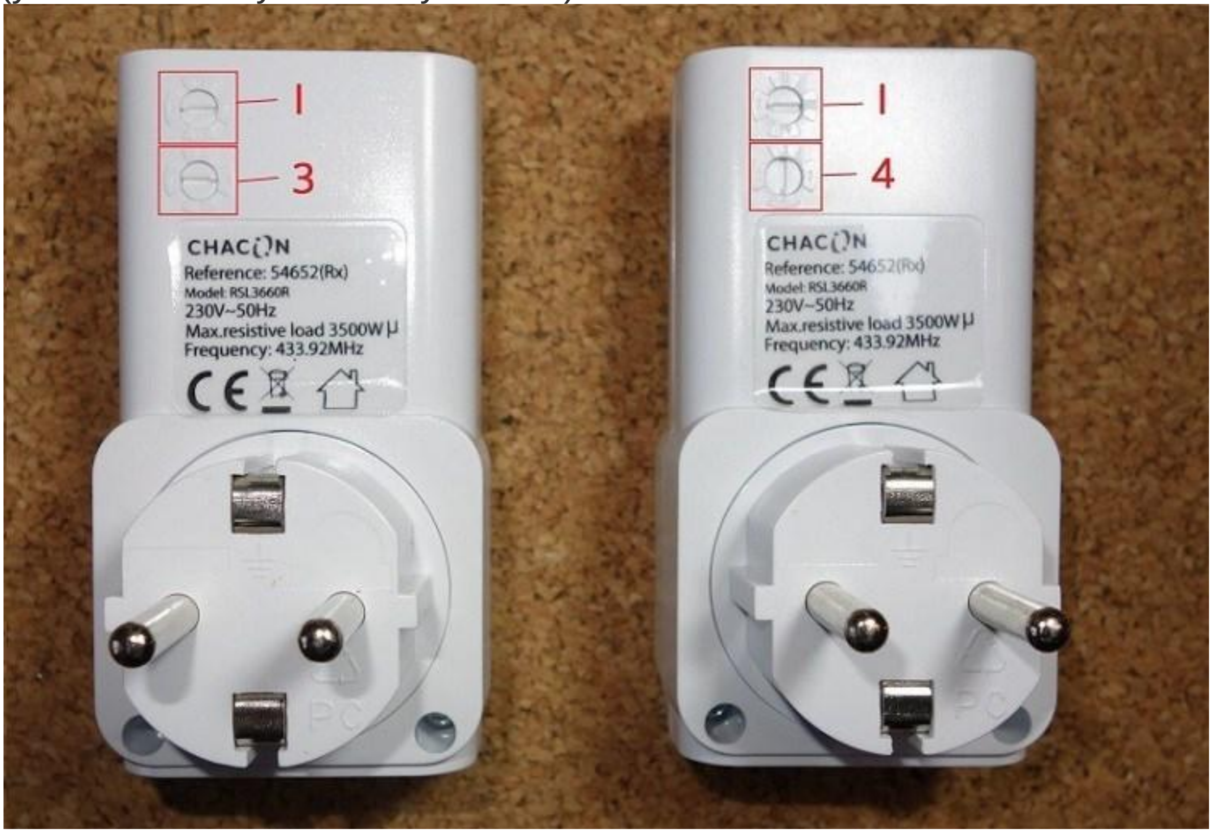
Here's how they look:

Setting the RF Channels

I've set my remote control to the **I** position.



The outlets must be both on the **I** position. I've selected channels **3** and **4** for the outlets (you can use any channel you want).



If you plug them to an outlet, you should be able to control the remote controlled outlets with your remote control.

Installing the RC Switch Library

The RC Switch library provides an easy way of using your ESP8266, ESP32, or Arduino to operate remote radio controlled devices. This will most likely work with all popular low-cost power outlet sockets.

1. Click here to download the RC Switch library. You should have a .zip folder in your **Downloads** folder
2. Unzip the .zip folder and you should get **rc-switch-master folder**
3. Rename your folder from ~~**rc-switch-master**~~ to **rc_switch**
4. Move the **rc_switch** folder to your Arduino IDE installation libraries folder
5. Then, re-open your Arduino IDE

Opening the Decoder Sketch

You need to decode the signals that your remote control sends, so that the Arduino or ESP8266 can reproduce those signals and ultimately control the outlets.

The library comes with several sketch examples. Within the Arduino IDE software, you need to go to **File** > **Examples** > **RC_Switch** > **ReceiveDemo_Advanced**. This next sketch opens:

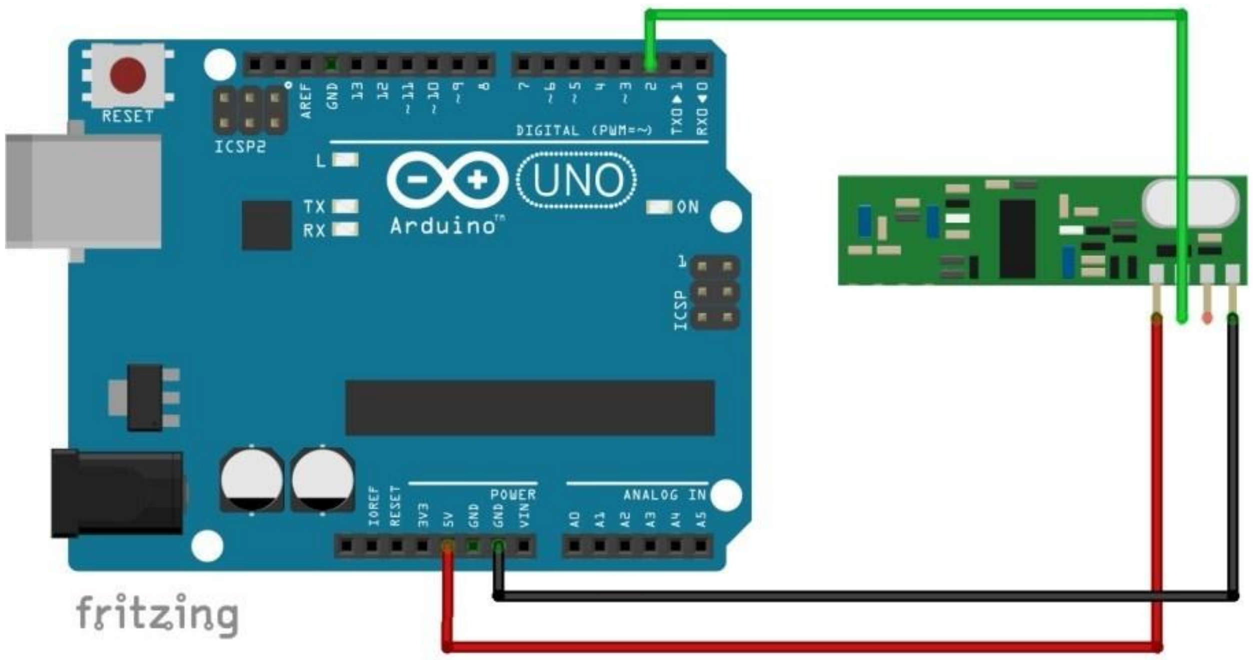
```
/*
  Example for receiving

  https://github.com/sui77/rc-switch/

  If you want to visualize a telegram copy the raw data and
  paste it into http://test.sui.li/oszi/
*/

#include

RCSwitch mySwitch = RCSwitch();

void setup() {
  Serial.begin(9600);
  mySwitch.enableReceive(0);  // Receiver on interrupt 0 => that is pin #2
}

void loop() {
  if (mySwitch.available()) {
    output(mySwitch.getReceivedValue(), mySwitch.getReceivedBitlength(), 
mySwitch.getReceivedDelay(), 
mySwitch.getReceivedRawdata(),mySwitch.getReceivedProtocol());
    mySwitch.resetAvailable();
  }
}
```

Having an Arduino board connected to your computer follow these instructions:

1. Go to the **Tools** tab
2. Select **Arduino UNO** board
3. Select the **COM** port
4. Press the **Upload** button.

Decoder – Schematics

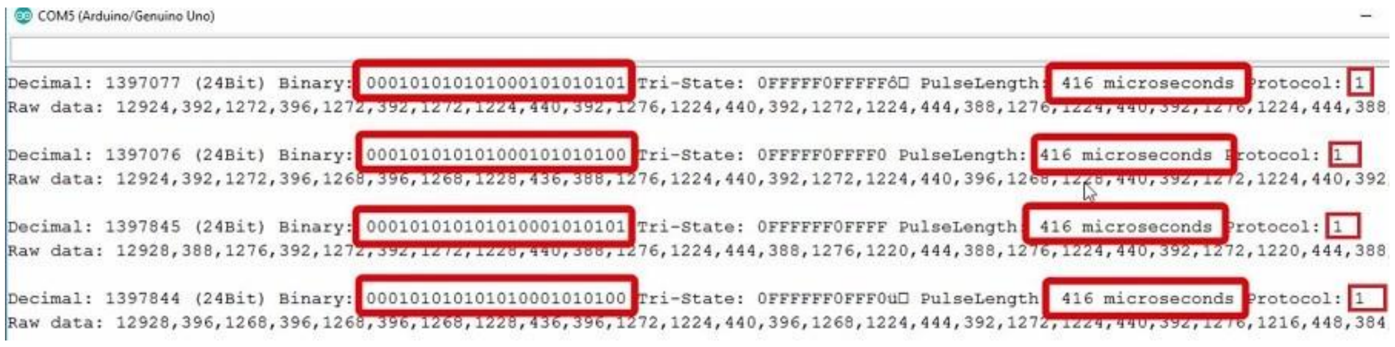
After uploading the sketch, connect an 433MHz RF receiver to Digital Pin 2 of your Arduino UNO board:



Decode RF Signals (codes)

Open the Arduino IDE serial monitor and start pressing the buttons. As shown in the video demonstration below:

After pressing each button one time, you can see the binary code for each button (it’s highlighted in red):



```
Decimal: 1397077 (24Bit) Binary: 000101010101000101010101 Tri-State: 0FFFFF0FFFFFô□ PulseLength: 416 microseconds Protocol: 1
Raw data: 12924,392,1272,396,1272,392,1272,1224,440,392,1276,1224,440,392,1272,1224,444,388,1276,1224,440,392,1276,1224,444,388

Decimal: 1397076 (24Bit) Binary: 000101010101000101010100 Tri-State: 0FFFFF0FFFF0 PulseLength: 416 microseconds Protocol: 1
Raw data: 12924,392,1272,396,1268,396,1268,1228,436,388,1276,1224,440,392,1272,1224,440,396,1268,1228,440,392,1272,1224,440,392

Decimal: 1397845 (24Bit) Binary: 000101010101010001010101 Tri-State: 0FFFFFF0FFFF PulseLength: 416 microseconds Protocol: 1
Raw data: 12928,388,1276,392,1272,392,1272,1228,440,388,1276,1224,444,388,1276,1220,444,388,1276,1224,440,392,1272,1220,444,388

Decimal: 1397844 (24Bit) Binary: 000101010101010001010100 Tri-State: 0FFFFFF0FFF0ú□ PulseLength 416 microseconds Protocol: 1
Raw data: 12928,396,1268,396,1268,396,1268,1228,436,396,1272,1224,440,396,1268,1224,444,392,1272,1224,440,392,1276,1216,448,384
```

Save your binary codes for each button press (you can also use the Decimal or Tri-State codes):

- Button 3 ON = (24Bit) Binary: 000101010101000101010101
- Button 3 OFF = (24Bit) Binary: 000101010101000101010100
- Button 4 ON = (24Bit) Binary: 000101010101010001010101
- Button 4 OFF = (24Bit) Binary: 000101010101010001010100

Save your Pulse Length: **416 Microseconds** and Protocol: **1**.

Send the RF Signals (codes)

You’ll need to customize the next sketch with your binary codes, pulse length and protocol:

```
/*
  Based on the SendDemo example from the RC Switch library
  https://github.com/sui77/rc-switch/
*/

#include
RCSwitch mySwitch = RCSwitch();

void setup() {
  Serial.begin(9600);

  // Transmitter is connected to Arduino Pin #10
  mySwitch.enableTransmit(10);

  // Optional set pulse length.
  mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);
  // Optional set protocol (default is 1, will work for most outlets)
  mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

  // Optional set number of transmission repetitions.
  // mySwitch.setRepeatTransmit(15);
}

void loop() {
  // Binary code - button 3
  mySwitch.send("000101010101000101010101");
  delay(1000);
  mySwitch.send("000101010101000101010100");
  delay(1000);

  // Binary code - button 4
  mySwitch.send("000101010101010001010101");
  delay(1000);
  mySwitch.send("000101010101010001010100");
  delay(1000);
}
```

In my case, the pulse length and protocol looks like this:

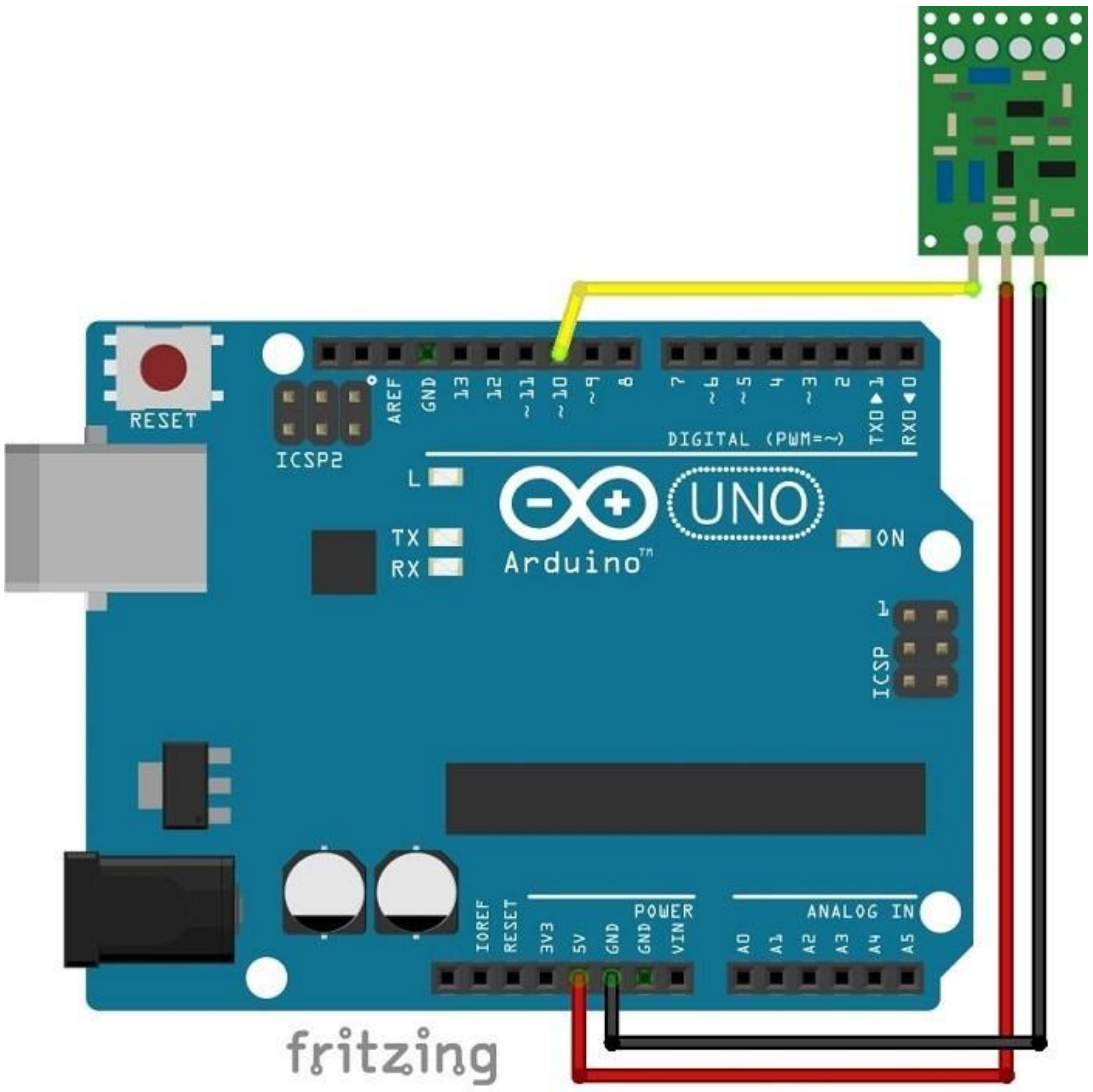
```
// Optional set pulse length.
mySwitch.setPulseLength(416);

// Optional set protocol (default is 1, will work for most outlets)
mySwitch.setProtocol(1);
```

Here's a binary sender example (you have to replace with your own binary codes):

```
mySwitch.send("000101010101000101010101");
```

Sender Schematics

After uploading the sketch to your Arduino board, assemble this circuit:



Both of your outlets should be turning on and off continuously.