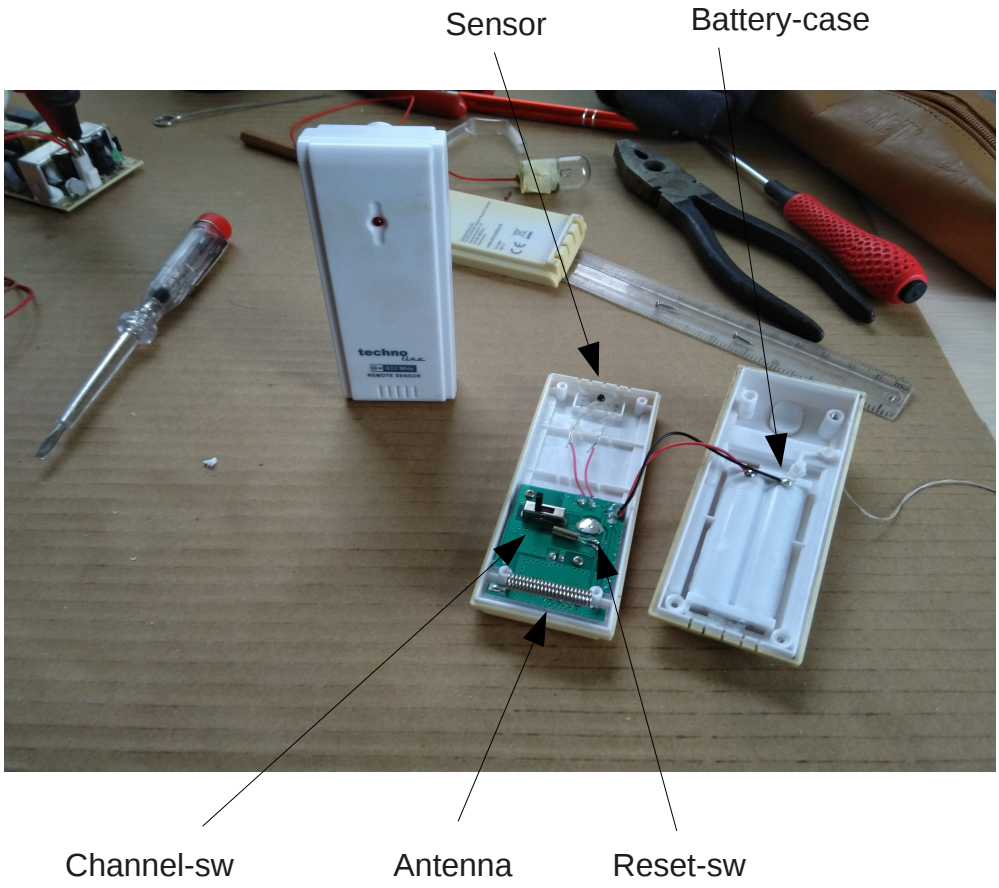
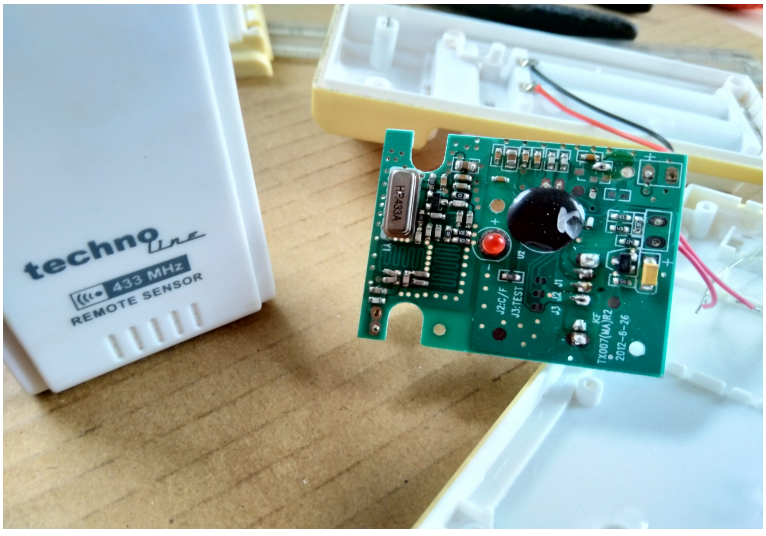
Christian Schaller Sep. 2020

Decoding wireless sensor devices temperature signal

My sensor is a **TechnoLine** temperature sensor, working on 433 MHz.



The main intent of decoding the signal with a microcontroller is, that i do not have any base-station which is able to receive and display these sensor values.



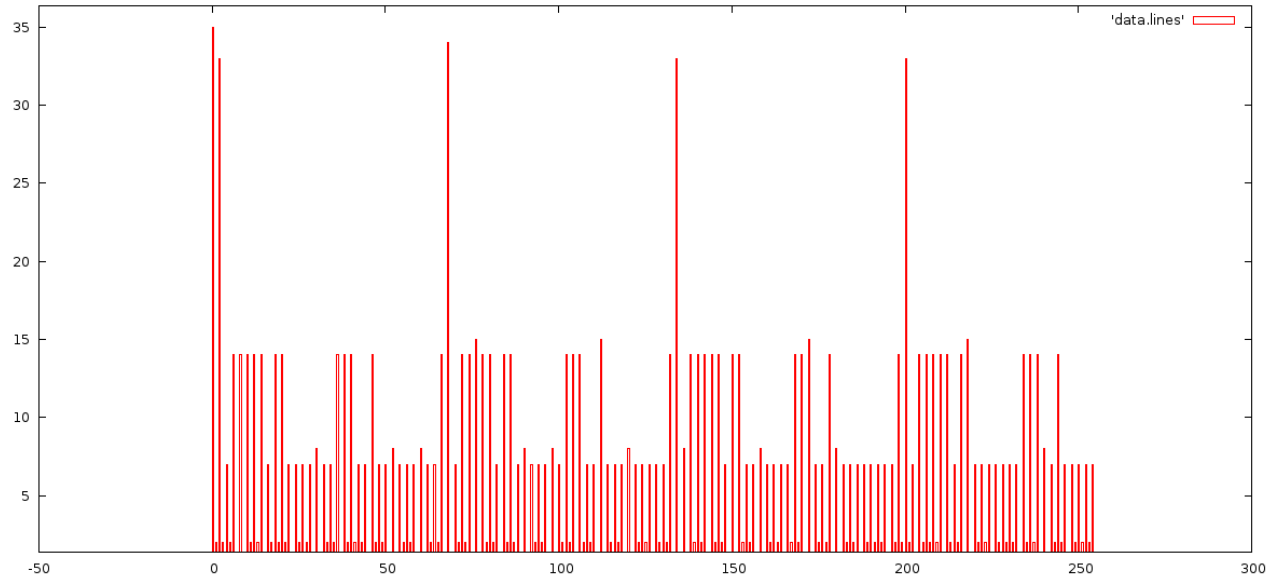
The first thing i did, was writing a simple program, which snaps received data, using a timer and external interrupt. This way i tracked all flanks belonging to the sensors signal.

Whenever the red LED flashed, i got a print out of hundreds of values over serial line.

After capturing the timed-data, i went to analyze the data, mainly with the help of gnuplot and some bash commands.

Christian Schaller Sep. 2020

Here you see a histogram of the timed data. The values on y-axis are the time between two flanks in the signal. In this example the timer is running at a speed of 3600 Hz.



As you can see, there are three distinct classes of timing periods. A very long period, with a value above 30. A shorter Period, having its value somewhere between 10 and 15. The next timing period lies between 5 and 10 and at least the smallest gap on the bottom of the diagram has a value of about 1 or 2.

In times:

class I : 8 ms < T

class II: 2ms < T < 4ms

class III: 1,5 ms < T < 2ms

class IV: T < 0,5ms

How to distinguish the timed-data?

Let's take a look on the longest timing period. This period appears twice at start of the transmission and repeats three more times. Furthermore, one can recognize, it divides the transmission into four parts. At a closer look, the four parts are nearly equal, which means, they provide the same data.

The class I timing shall mark the beginning of a complete transmission, therefor give time for preparation, as well as give a little pause time inbetween, to process received data.

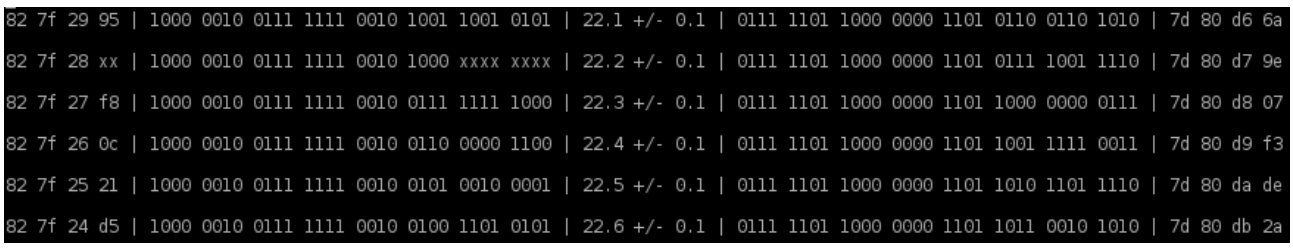
The class II and III timed data, are probably the desired zeros and ones (databits) and the class IV data, is again the timing gap between two databits.

Which means in conclusion, to reproduce the binary coded signal, you just use the class II and III timed data values.

My first assumption for bitcoding was assigning class II data zero and class III one. The next picture shows the timed data of only one of the four equal parts, which is 32 bits long each. Under the times, you can see the assigned values of zeros and ones, furthermore the decimal and hexadecimal values accordingly.

This picture shows 5 different transmissions. In all examples the first ten bits are equal. The first three examples have the same channel number, but different temperature. The last three examples have nearly the same temperature, but different channel values. The sensor allows three different channels, which need at least two bit. It must be bit number eleven and twelve, reading from the left. This means, the first twelve bit are identifying the sensor. (Note: Resetting the sensor does not change identification here.)

The other 20 bits should be the temperature and possibly some error correction. Further tests have as well to verify, which bit assigning is correct.



The above image shows further examples of continous ascending temperature values. Next to the sensor, I have used a normal thermometer, to assing a nearby value for each example. On the left you see the coding as described above. On the right, the coding is inverted. One can see, that the temperature value is somehow counting down on the left, if you ignore the last databyte. On the right hand side, the temperature value is slightly counting up, as expected.

When varying the sensors channel, while the temperatue is constant, only the two bits for the channel and the last databyte change values and return accordingly to the same values.

This leads to may be 12 bit temperature value and the last byte could be some error correction stuff.

Christian Schaller Sep. 2020

Since I now know the temperature at a given point and decided to switch the bitcoding, so that zeros are now class III and ones belong to class II data-timing. This has the opportunity, that I can simply use the difference to a known temperature point for calculating the measured centigrade.

Another interesting argument leads to the correct decision for this type of coding. If you look to the values on the right and interpret the 12 bit temperatre value as integer number. The integer value is quiet near the real temperature read from the normal thermometer. (Note: The decimal point can be thought to be before the last decimal digit.) The difference stays nearly constant, so that I only need to decode the twelve temperature bit as integer and add a correction, in my example 7 centigrade.

In the last example above, the measured integer temperature value is 219. The verification-thermometer showed 22.6 centigrade. Now I simply add the difference of 7 to the measured value from the sensor and get 226.

The second sensor is of the same type. Now I activated both sensors, which use the same protocol, but the second one needs a slightly different correction term!

At last I tried to find out how the last byte could be combined with the others. But no match until now. May be this is CRC.

If you have an idea, I would be really interested.

Please e-mail, if you find this helpful.

cs@sprintmail.de

With regards,

Christian