# WS3000 Hack

4 August, 2020
0:20

## WS3000 hack

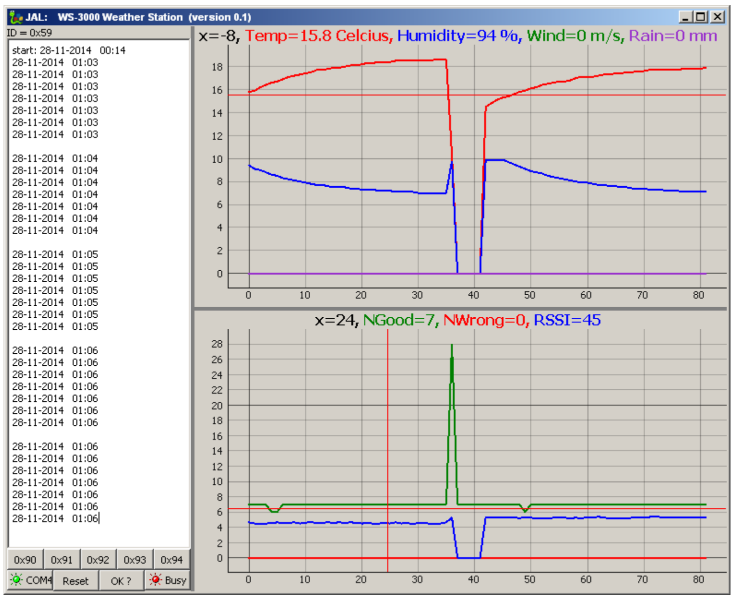
last updated, dec 2014, Stef Mientki

### Introduction

The Alecto WS3000 is a non-expensive wheather station, with outdoor temperature, humidity, windspeed (windgust) and rainmeter. Indoor it has temperature, humidity and pressure sensor. THe outdoor unit also contains a DCF receiver, which gets the time from an atomic clock, transmitted by longwave radio transmitter in Frankfurt (Germany).
    http://en.wikipedia.org/wiki/DCF77
The WS3000 is also sold under different names, like WH1070, WH1080.
There are several versions, having different communication protocols. The older versions works with OOK signals (On-Off-Keying), the newer ones uses FSK (Frequency Shift Keying).
This document describes the receiving and decoding of an outdoor WS3000 (with FSK) with an SI4432 transceiver, and a PIC coded in JAL.
For optimal reception you should use an antenna of 17.2 cm (= 1/4 wavelength).

### Desktop program

In the top graph the sensor data is shown. Around the whole hour, there's a dip in all the sensor data, because data is not send in that period.
In the bottom graph the RF-signal strength and the number of received lines per package is shown. When the signal strength is good, you'll almost sees 7 good lines per package (sometimes 6, because sometimes the JAL program misses one of the lines. The large peek in the number of good lines is caused by a battery exchange, in which case the transmitter sends a whole lot of lines.


JAL: WS-3000 Weather Station (version 0.1)
ID = 0x59
x=-8, Temp=15.8 Celcius, Humidity=94 %, Wind=0 m/s, Rain=0 mm
x=24, NGood=7, NWrong=0, RSSI=45
0x90
0x91
0x92
0x93
0x94
COM4
Reset
OK ?
Busy

```
start: 28-11-2014  00:14
28-11-2014  01:03
28-11-2014  01:03
28-11-2014  01:03
28-11-2014  01:03
28-11-2014  01:03
28-11-2014  01:03
28-11-2014  01:03

28-11-2014  01:04
28-11-2014  01:04
28-11-2014  01:04
28-11-2014  01:04
28-11-2014  01:04
28-11-2014  01:04
28-11-2014  01:04

28-11-2014  01:05
28-11-2014  01:05
28-11-2014  01:05
28-11-2014  01:05
28-11-2014  01:05
28-11-2014  01:05
28-11-2014  01:05

28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06

28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
28-11-2014  01:06
```

### Package transmission

Each 48 seconds a package is send by the transmitter, containing the sensor data.
Around the whole hour, xx:58 .. xx:05 there's nothing transmitted (probably to get a clean DCF receiver timing).
A few minutes after an whole hour, not the sensor data is send but a number of packages containg the actual time is sent. In the paragraph "Desktop Program" a complete set of time packages is shown.
Each package contains 7 repetitions of the sensor data (or time data).
Between each repetition there's a space of 7 bits no RF signal.
The 7 repetitions are often identical, but if a sensor value changes within the transmit window, the values will instantanously change, like in the example below.

```
537 1 package of sensor data
538 52 40 D0 45 00 00 03 0C BB
539 52 40 D0 45 00 00 03 0C BB
540 52 40 D0 45 00 00 03 0C BB
541 52 40 D0 45 00 00 03 0C BB
542 52 40 D0 45 00 00 03 0C BB
543 52 40 D0 45 00 00 03 0C BB
544 52 40 D1 44 00 00 03 0C 3F
```

Sensor data, one line from a package (including JAL sync bytes)

| Byte | bits | Description |
|---|---|---|
| 0xCC | | JAL sync byte |
| 0xBB | | JAL sync byte |
| 0xAA | | JAL sync byte |
| 0 | 7:4 | Message Type: 5 = Sensor Data, 6 = Time Data |
| | 3:0 | MSB of ID |
| 1 | 7:4 | LSB of ID |
| | 3 | Sign of Temperature |
| | 2:0 | MSB of Temperature |
| 2 | | LSB of Temperature |
| 3 | | Humidity |
| 4 | | Windspeed (*1.22 ??) |
| 5 | | Wind gust ? |
| 6 | | MSB of Rain (starts at 0x030C) |
| 7 | | LSB of Rain (stepsize = 0.3 mm) |
| 8 | | CRC |
| CRC | | JAL CRC calculation |
| RSSI | | Max RSSI |
| 7 bits 0 | | 7 bits space between messages |

The JAL sycn bytes are added by JAL to serve as sync bytes to the desktop viewer.
The ID is a random number of 8 bits, which will change after a battery exchange.
Time data, one line from a package (including JAL sync bytes).

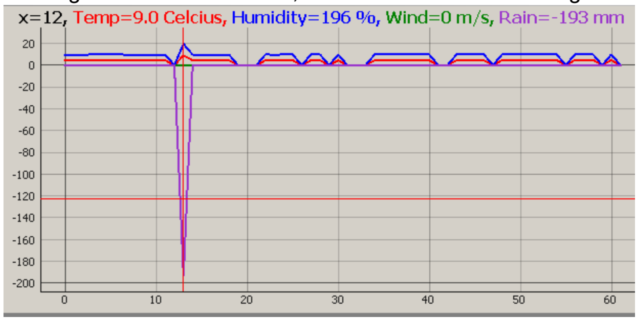
| Byte | bits | Description |
|---|---|---|
| 0xCC | | JAL sync byte |
| 0xBB | | JAL sync byte |
| 0xAA | | JAL sync byte |
| 0 | 7:4 | Message Type: 5 = Sensor Data, 6 = Time Data |
| | 3:0 | ???? |
| 1 | | ???? |
| 2 | 7:6 | ???? |
| | 5:0 | Hour (BCD) |
| 3 | | Minute (BCD) |
| 4 | | Second (BCD) |
| 5 | | Year (last 2 digits) ( BCD) |
| 6 | 7:6 | ???? |
| | 5:0 | Month ( BCD) |
| 7 | | Day (BCD) |
| 8 | | CRC |
| CRC | | JAL CRC calculation |
| RSSI | | Max RSSI |
| 7 bits 0 | | 7 bits space between messages |

## WS3000 interface specifications

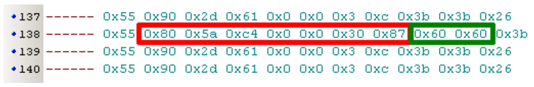
Carrier Frequency = 868.3 MHz
Frequency deviation = 67 kHz
Bitrate = 17.24 kHz ( or should it be 19.2 kHz ??? )

## Special problems

Although the CRC code is ok, sometimes still set of wrong values is received.

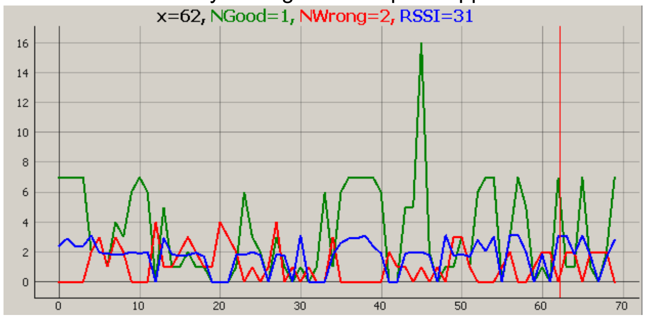


while the CRC code is ok (green block), the data is rubish



```
•137 ------ 0x55 0x90 0x2d 0x61 0x0 0x0 0x3 0xc 0x3b 0x3b 0x26
•138 ------ 0x55 0x80 0x5a 0xc4 0x0 0x0 0x30 0x87 0x60 0x60 0x3b
•139 ------ 0x55 0x90 0x2d 0x61 0x0 0x0 0x3 0xc 0x3b 0x3b 0x26
•140 ------ 0x55 0x90 0x2d 0x61 0x0 0x0 0x3 0xc 0x3b 0x3b 0x26
```

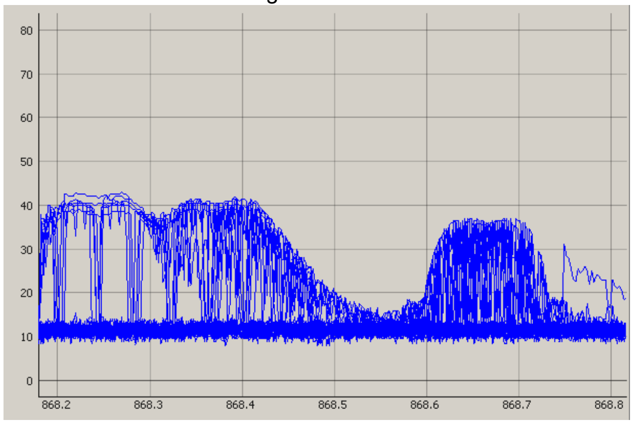
Don't understand why the high NGood peek appears.



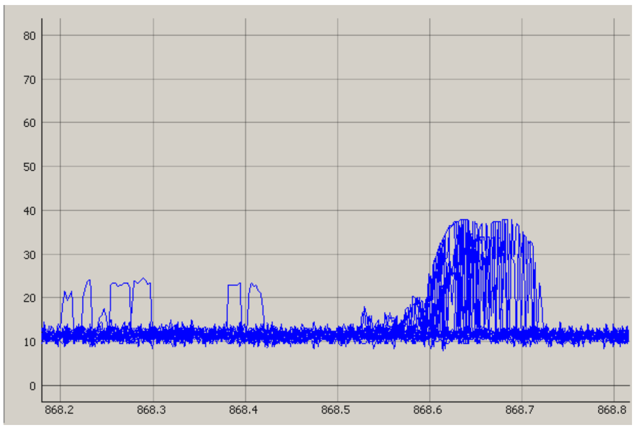
## From here, this document should be read bottom up.

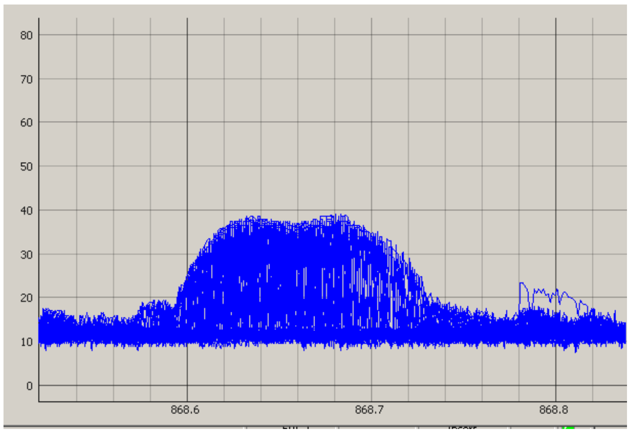
## Interesting

What's that second FSK signal ?



to be sure



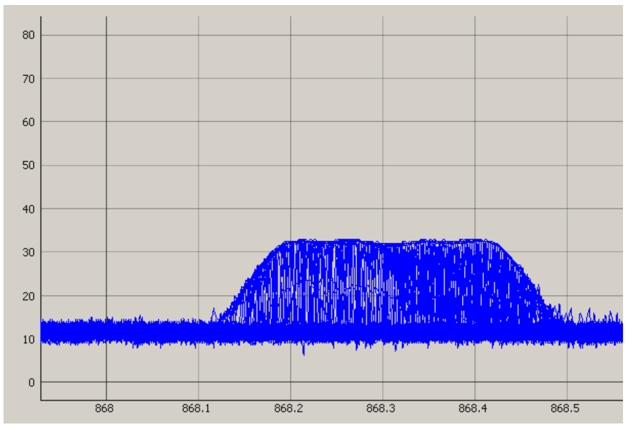


```
Start Frequency = 868.52 MHz
Center Frequency = 868.68 MHz
End Frequency = 868.84 MHz
DeltaTime = 196.95237624
DeltaTime = 0.46291128
DeltaTime = 0.44633212
DeltaTime = 0.45416548
DeltaTime = 411.16738552
DeltaTime = 0.44779528
DeltaTime = 0.43870576
DeltaTime = 0.46591592
DeltaTime = 427.07110588
DeltaTime = 0.476807759999
DeltaTime = 0.44462336
DeltaTime = 377.49958872
DeltaTime = 0.44943972
DeltaTime = 0.4635336
DeltaTime = 0.44938056
DeltaTime = 220.59665432
DeltaTime = 0.450216599999
DeltaTime = 0.456704480001
DeltaTime = 0.45530632
DeltaTime = 238.32654436
DeltaTime = 0.4627464
DeltaTime = 0.44753528
DeltaTime = 0.44767688
DeltaTime = 279.25219536
DeltaTime = 0.47346388
DeltaTime = 0.44916456
DeltaTime = 98.70207012
DeltaTime = 459.8221028
DeltaTime = 0.4489698
DeltaTime = 0.46272316
DeltaTime = 0.44689752
DeltaTime = 262.43175732
DeltaTime = 0.46423224
DeltaTime = 0.448893
DeltaTime = 0.46168792
DeltaTime = 82.7743566
DeltaTime = 0.44784464
DeltaTime = 0.459976960001
DeltaTime = 0.450116079999
DeltaTime = 196.4906952
DeltaTime = 0.44671384
DeltaTime = 0.465313879999
DeltaTime = 0.448005640001
DeltaTime = 140.08798076
DeltaTime = 0.44613304
DeltaTime = 0.46323304
DeltaTime = 0.44819164
DeltaTime = 151.01516484
DeltaTime = 0.4440772
DeltaTime = 0.45295044
DeltaTime = 0.46276
DeltaTime = 401.60525204
DeltaTime = 0.44971536
DeltaTime = 0.45154452
DeltaTime = 0.46013204
DeltaTime = 138.72836372
DeltaTime = 0.44447004
DeltaTime = 0.45569536
DeltaTime = 0.45373028
DeltaTime = 257.88360684
DeltaTime = 0.45756408
DeltaTime = 0.45242096
DeltaTime = 36.37693564
DeltaTime = 0.478957
DeltaTime = 0.434837559999
```

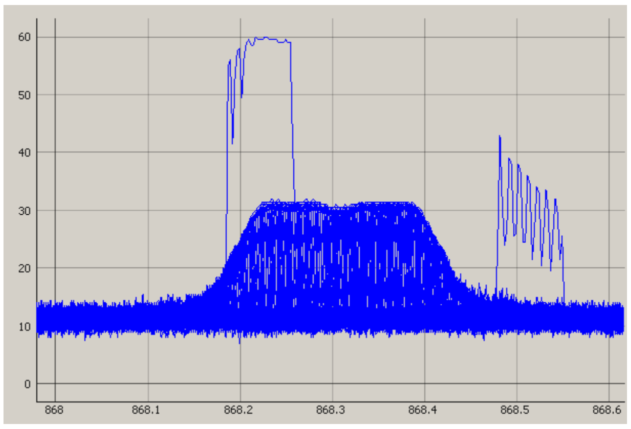
DeltaTime = 0.446786320001

## Frequency Deviation

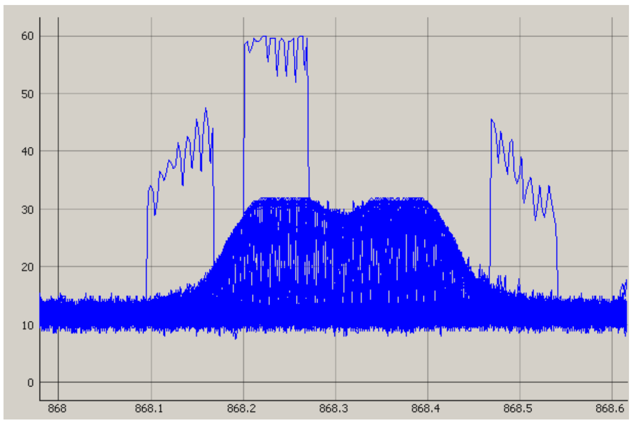
On the web they speak about a frequency deviation of 134 kHz. From the pictures below we concluse that probably the wrong definition of frequency deviation  is used, so half of it (67 kHz) looks more realistic.
at 67 kHz : bandwidth is a little too high (and this picture is taken with the low Baudrate of 17.24 kHz)



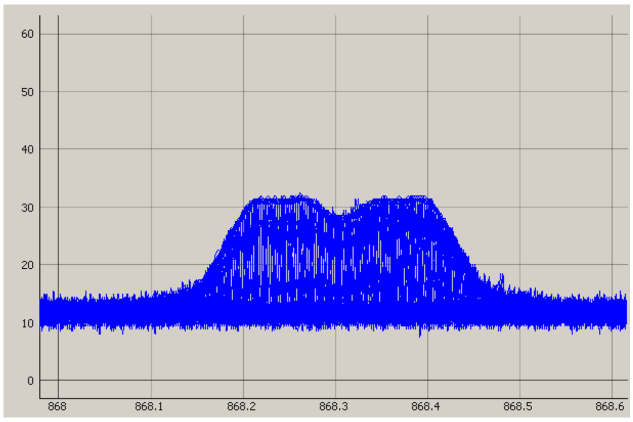
at 50 kHz: bandwith is a little too low



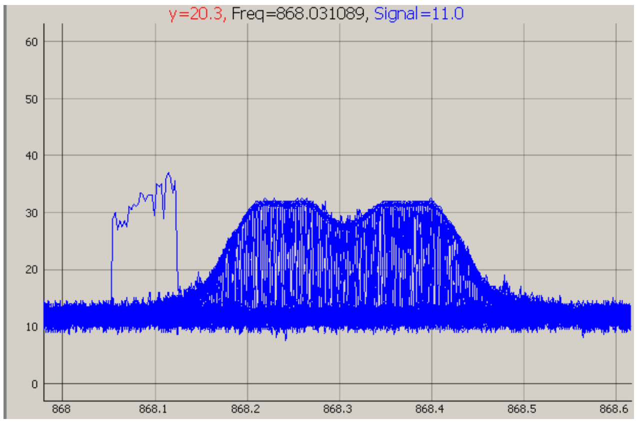
at 60 kHz: still a bit too low



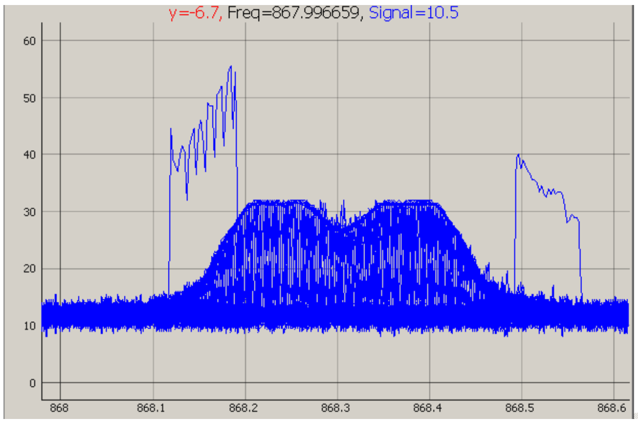
at 62 kHz:



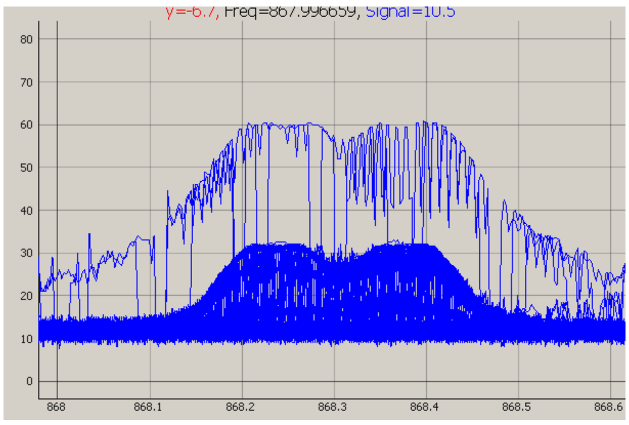
at 64 kHz :



at 67 kHz : this looks perfect now. The previous measurement at 17.24 kHz Bitrate is probaly wrong.
The central frequency of the WS3000 is about 15 to 20 kHz above the 868.3 kHz.



WS3000 + UHF-generator:



## Baudrate

For the greatest sensitivity it's necessary to know both the freqeuncy-deviation and the Baudrate. As in the pictures below, we can see that with the values found on the internet ( 76 kHz and 17.24 kHz) the bandwidth from the UHF generator is larger than from the WS3000. Interesting to see that everyone is talking about 17.24 kHz and no one is talking how it's measured or derived. So I guess one guy measured it and everyone is copying that measurement. Also the suggested accuray of +/- 0.01 kHz is doubtfull, especially when you realize that it's a small burst with very low duty-cycle.
Measurements were done with the following JAL program (GPIO2 was programmed for Rx clock output)

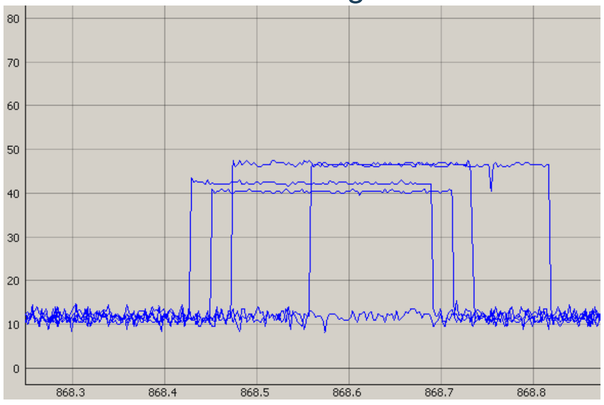
```
procedure SI4432_Baudrate_Loop ( byte in Treshold ) is
  var word N
  var byte RS232
  var bit  Data_Old
  alias Data is GPIO2
  var byte ARR [200]
  var byte i

forever loop
    while SI4432_Read ( 0x26 ) < Treshold loop
    end loop
    serial_hw_write (0xEE)
    serial_hw_write (0xEE)
    serial_hw_write (SI4432_Read ( 0x26 ))

    for count(ARR) using i loop
      N = 0
      Data_Old = Data
      while Data == Data_Old loop
        N = N + 1
      end loop
      Arr[i] = N
    end loop

    for count(ARR) using i loop
      serial_hw_write ( ARR[i] )
    end loop

    while SI4432_Read ( 0x26 ) > Treshold loop
    end loop
  end loop
```

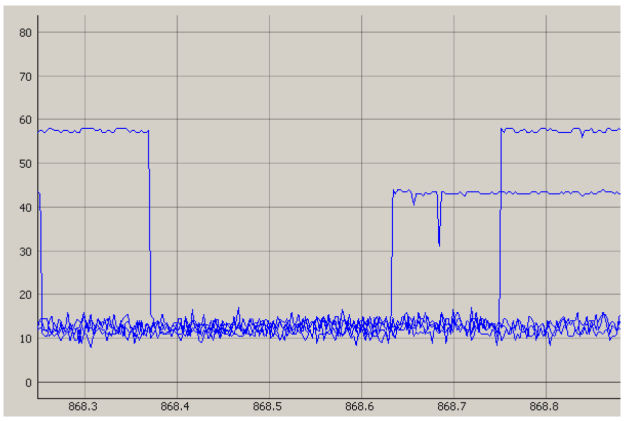
No absolute measurements were done, but the average half bit time was compared (always over 200 samples) between the WS3000 and the UHF-generator set at different Baudrates.
The average half bit time of the WS3000 was 6.17.
With the UHF-generator programmed at 17.24 kHz we measured a mean half-bit time of 5.7, so that's about 8% too low.
With the standard Baudrate of 19.2 kHz, we measured 6.06, which is still 1.5 % too low, but much better than the 17.24 kHz, and more logical. In general an error of 2.5% timing error in serial communication is acceptable.

## View with spectrum analyzer

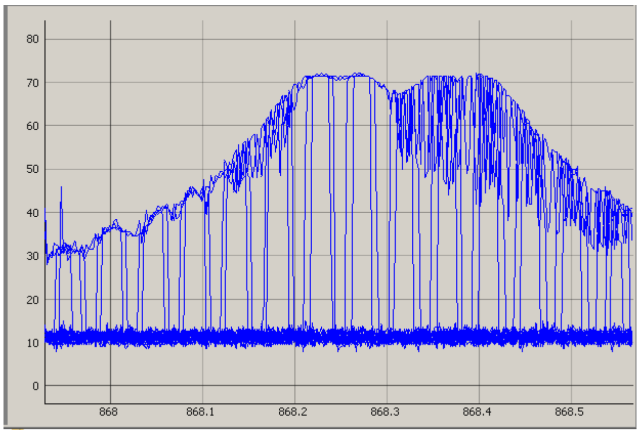
...plaatje
f-carrier = 434 MHz
probably FSK with f-deviation = 50 kHz
Repetion_Rate = 1.1 sec
Packet duration = 150 msec

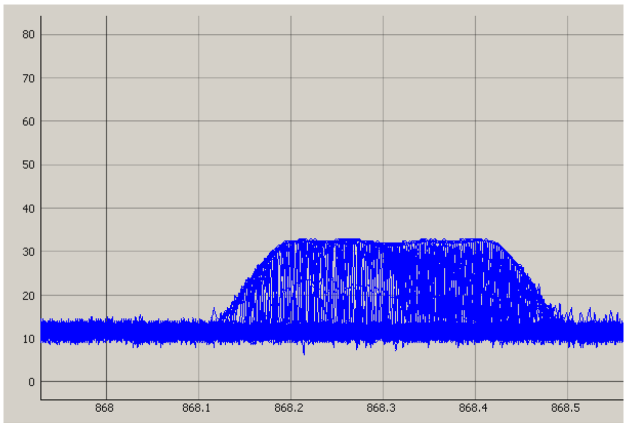
## Antenna Switch and length





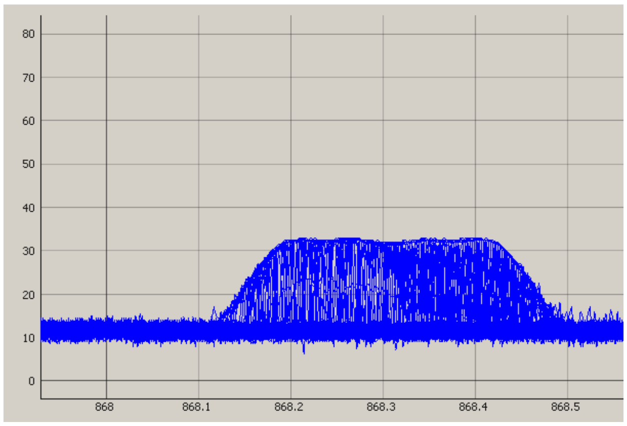
## WS3000





|  |  |
|---|---|
| -- Modulation Type | : FSK |
| -- Frequency Deviation | : 76.0 [kHz] |
| -- Manchester | : OFF |
| -- Carrier Frequency | : 868.3 [MHz] |
| -- Data Rate | : 17.24 [kb/s] |

The UHF Generator is set at maximum output power, distance between transmitter and receiver is 20 cm, and still that low signal !!
But looking at the shape, it's somewhat too small.
The bandwidth looks a litlle bit too large.



## Frequency mismatch

AFC is enabled

+/-20 kHz

```
00 00 00 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
00 00 00 D4 00 00 10 08 89 09 8A 0A 8B 0B 8C 0C 80 00 00 00 00 00 00
00 00 00 D4 00 00 20 22 24 26 28 2A 2C 2E 30 32 00 00 00 00 00 00 00
00 00 00 D4 00 00 20 22 24 26 28 2A 2C 2E 30 34 00 00 00 00 00 00 00
00 00 00 D4 00 00 08 08 89 09 85 05 45 85 C6 03 20 00 00 00 00 00 00
00 00 00 D4 00 00 08 08 89 09 8A 0A 8B 0B 8C 06 40 00 00 00 00 00 00
```

```
EE EE 21 20 52 A5 EE EE
EE EE 21 20 52 A6 EE EE
FF FF 21 20 D2 AA FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 A5 EE EE
EE EE 21 20 52 A6 EE EE
EE EE 21 20 52 A6 EE EE
EE EE 21 20 52 A6 EE EE
EE EE 21 20 52 A6 EE EE
EE EE 21 20 52 A5 EE EE
EE EE 21 20 52 A5 EE EE
EE EE 21 20 52 A6 EE EE
EE EE 21 20 52 A5 EE EE
FF FF 21 20 D2 A7 FF FF
EA 00 00 04 04 44 84 C2 82 A2 C2 F1 81 93 FF FF FF FF
```

+/-10 kHz, detection OK

```
EE EE 21 20 52 AC EE EE
FF FF 21 20 D2 AC FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AC FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF
```

## AFC enabled / disabled

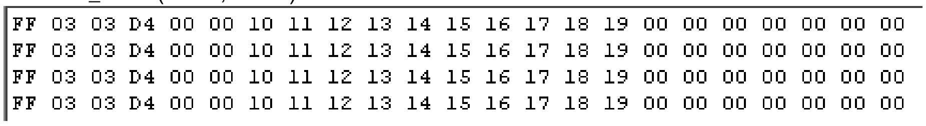
Disabling the AFC doesn't show any changes.
If AFC is enabled (default), registers 2B,73,74 should hold the frequency offset (after the sync word is detected) ???

```
00 00 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
00 00 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
00 00 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
00 00 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
```

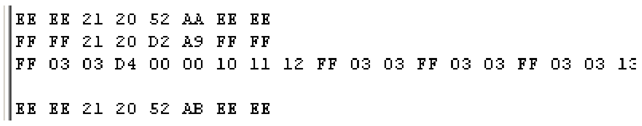
```
EE EE 21 20 52 AA EE EE
FF FF 21 20 D2 AB FF FF
00 00 03 D4 00 00 10 11 12 00 00 03 00 00 03
00 00 03 13 14 15 16 17 18 00 00 03 00 00 03 00 00 03 19 3F FF FF FF FF

EE EE 21 20 52 AA EE EE
```

after intializing these registers
  SI4432_Write ( 0x2B, 0xFF )

```
SI4432_Write ( 0x73, 0xFF )
SI4432_Write ( 0x74, 0xFF )
```



```
FF 03 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
FF 03 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
FF 03 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
FF 03 03 D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
```



```
EE EE 21 20 52 AA EE EE
FF FF 21 20 D2 A9 FF FF
FF 03 03 D4 00 00 10 11 12 FF 03 03 FF 03 03 FF 03 03 13

EE EE 21 20 52 AB EE EE
```

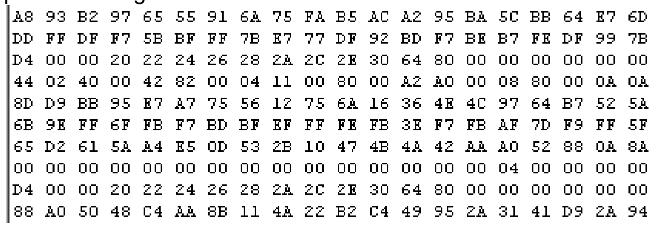
## Minimal Preamble length

We need at least 3 nibbles to get a resonable error free detection.

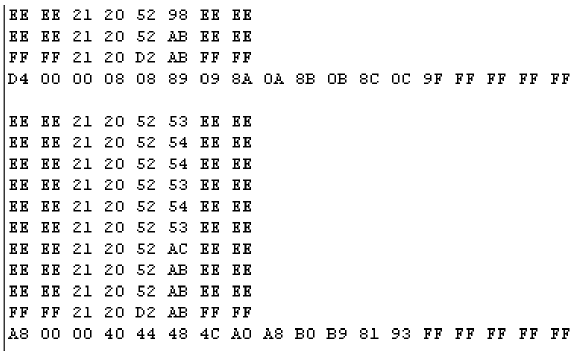
## Packet Hacker 2

Probably as done in Packet_Hacker can be achieved with a normal read procedure, so we implemented the normal read procedure in Packet_Hacker_2. We made a comparison between those two procedures, by testing low preamble tresholds for both the procedures, on the left Packet_Hacker_2 and on the right Packet_Hacker.

preamble length = 1

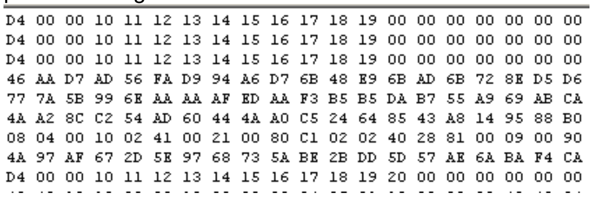


```
A8 93 B2 97 65 55 91 6A 75 FA B5 AC A2 95 BA 5C BB 64 E7 6D
DD FF DF F7 5B BF FF 7B E7 77 DF 92 BD F7 BE B7 FE DF 99 7B
D4 00 00 20 22 24 26 28 2A 2C 2E 30 64 80 00 00 00 00 00 00
44 02 40 00 42 82 00 04 11 00 80 00 A2 A0 00 08 80 00 0A 0A
8D D9 BB 95 E7 A7 75 56 12 75 6A 16 36 4E 4C 97 64 B7 52 5A
6B 9E FF 6F FB F7 BD BF EF FF FE FB 3E F7 FB AF 7D F9 FF 5F
65 D2 61 5A A4 E5 0D 53 2B 10 47 4B 4A 42 AA A0 52 88 0A 8A
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00
D4 00 00 20 22 24 26 28 2A 2C 2E 30 64 80 00 00 00 00 00 00
88 A0 50 48 C4 AA 8B 11 4A 22 B2 C4 49 95 2A 31 41 D9 2A 94
```

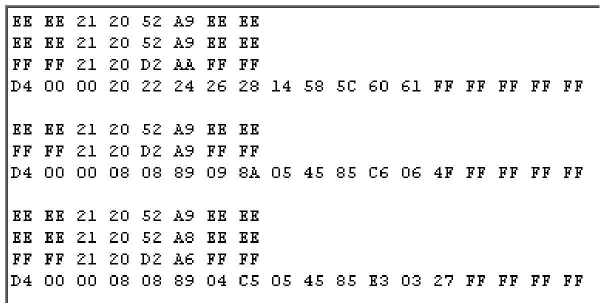


```
EE EE 21 20 52 98 EE EE
EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 08 08 89 09 8A 0A 8B 0B 8C 0C 9F FF FF FF FF

EE EE 21 20 52 53 EE EE
EE EE 21 20 52 54 EE EE
EE EE 21 20 52 54 EE EE
EE EE 21 20 52 53 EE EE
EE EE 21 20 52 54 EE EE
EE EE 21 20 52 53 EE EE
EE EE 21 20 52 AC EE EE
EE EE 21 20 52 AB EE EE
EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AB FF FF
A8 00 00 40 44 48 4C A0 A8 B0 B9 81 93 FF FF FF FF FF
```

preamble length = 2

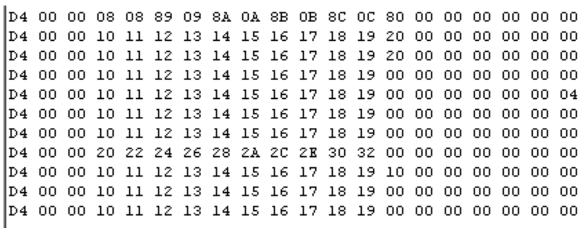


```
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
46 AA D7 AD 56 FA D9 94 A6 D7 6B 48 E9 6B AD 6B 72 8E D5 D6
77 7A 5B 99 6E AA AA AF ED AA F3 B5 B5 DA B7 55 A9 69 AB CA
4A A2 8C C2 54 AD 60 44 4A A0 C5 24 64 85 43 A8 14 95 88 B0
08 04 00 10 02 41 00 21 00 80 C1 02 02 40 28 81 00 09 00 90
4A 97 AF 67 2D 5E 97 68 73 5A BE 2B DD 5D 57 AE 6A BA F4 CA
D4 00 00 10 11 12 13 14 15 16 17 18 19 20 00 00 00 00 00 00
```

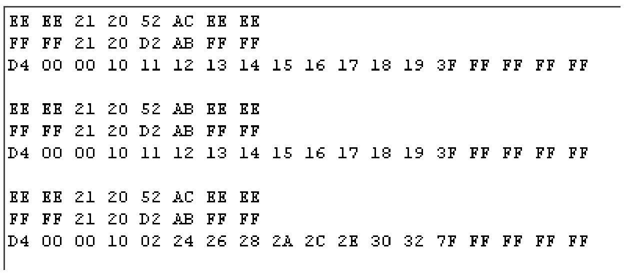


```
EE EE 21 20 52 A9 EE EE
EE EE 21 20 52 A9 EE EE
FF FF 21 20 D2 AA FF FF
D4 00 00 20 22 24 26 28 14 58 5C 60 61 FF FF FF FF FF

EE EE 21 20 52 A9 EE EE
FF FF 21 20 D2 A9 FF FF
D4 00 00 08 08 89 09 8A 05 45 85 C6 06 4F FF FF FF FF

EE EE 21 20 52 A9 EE EE
EE EE 21 20 52 A8 EE EE
FF FF 21 20 D2 A6 FF FF
D4 00 00 08 08 89 04 C5 05 45 85 E3 03 27 FF FF FF FF
```

preamble_length = 3

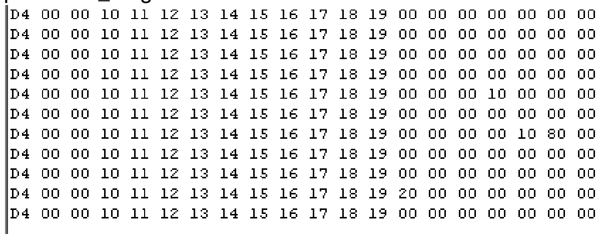


```
D4 00 00 08 08 89 09 8A 0A 8B 0B 8C 0C 80 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 20 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 20 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 04
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 20 22 24 26 28 2A 2C 2E 30 32 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 10 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
```

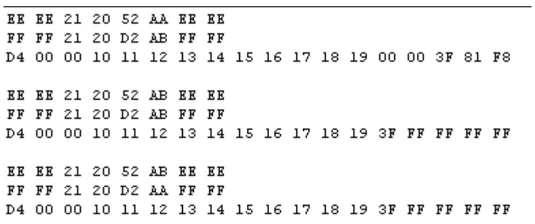


```
EE EE 21 20 52 AC EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 AC EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 10 02 24 26 28 2A 2C 2E 30 32 7F FF FF FF FF
```

preamble_length = 4



```
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 10 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 10 80 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 20 00 00 00 00 00 00
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00 00
```



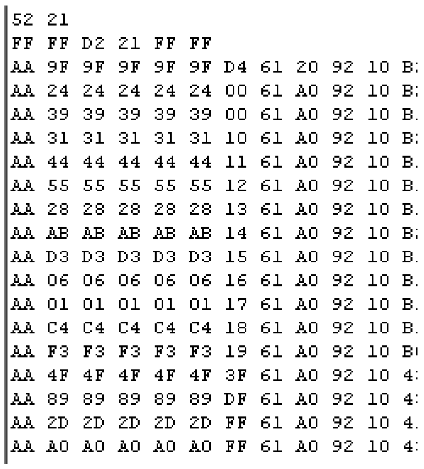
```
EE EE 21 20 52 AA EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 00 00 3F 81 F8

EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AA FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF
```

## Packet Hacker

Because we don't know anything about the contents of the package, and hope it will at least contains a preamble and 1 sync byte (0x4D which is  often the standard), we want to see if we can read it.
Because we want to use the Rx-FIFO, we mimick the behaviour of the packet handler, i.e.

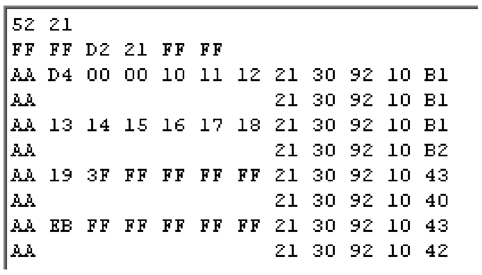
- wait for valid preamble
- wait for valid sync word
- read all data
- reset the receiver if the RSSI becomes too low

```
-- test if RX FIFO not empty
if ( R02 & 0x20 ) == 0 then
  serial_hw_write ( 0xAA )
  while ( R02 & 0x20 ) == 0 loop
    serial_hw_write ( SI4432_Read ( 0x7F ) )
    R02 = SI4432_Read ( 0x02 )
  end loop
  serial_hw_write ( R02 )
  serial_hw_write ( R03 )
  serial_hw_write ( R04 )
  serial_hw_write ( SI4432_Read ( 0x31 ))
  serial_hw_write ( SI4432_Read ( 0x26 ))
end if
```



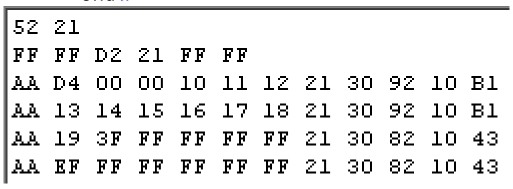
```
52 21
FF FF D2 21 FF FF
AA 9F 9F 9F 9F 9F D4 61 20 92 10 B
AA 24 24 24 24 24 00 61 A0 92 10 B
AA 39 39 39 39 39 00 61 A0 92 10 B
AA 31 31 31 31 31 10 61 A0 92 10 B
AA 44 44 44 44 44 11 61 A0 92 10 B
AA 55 55 55 55 55 12 61 A0 92 10 B
AA 28 28 28 28 28 13 61 A0 92 10 B
AA AB AB AB AB AB 14 61 A0 92 10 B
AA D3 D3 D3 D3 D3 15 61 A0 92 10 B
AA 06 06 06 06 06 16 61 A0 92 10 B
AA 01 01 01 01 01 17 61 A0 92 10 B
AA C4 C4 C4 C4 C4 18 61 A0 92 10 B
AA F3 F3 F3 F3 F3 19 61 A0 92 10 B
AA 4F 4F 4F 4F 4F 3F 61 A0 92 10 4
AA 89 89 89 89 89 DF 61 A0 92 10 4
AA 2D 2D 2D 2D 2D FF 61 A0 92 10 4
AA A0 A0 A0 A0 A0 FF 61 A0 92 10 4
```

```
-- test if RX FIFO not empty
if ( R03 & 0x10 ) != 0 then
  serial_hw_write ( 0xAA )
  while ( R02 & 0x20 ) == 0 loop
    serial_hw_write ( SI4432_Read ( 0x7F ) )
    R02 = SI4432_Read ( 0x02 )
  end loop
  serial_hw_write ( R02 )
  serial_hw_write ( R03 )
  serial_hw_write ( R04 )
  serial_hw_write ( SI4432_Read ( 0x31 ))
  serial_hw_write ( SI4432_Read ( 0x26 ))
end if
```



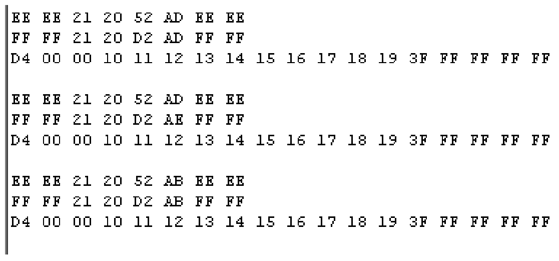
```
52 21
FF FF D2 21 FF FF
AA D4 00 00 10 11 12 21 30 92 10 B1
AA                   21 30 92 10 B1
AA 13 14 15 16 17 18 21 30 92 10 B1
AA                   21 30 92 10 B2
AA 19 3F FF FF FF FF 21 30 92 10 43
AA                   21 30 92 10 40
AA EB FF FF FF FF FF 21 30 92 10 43
AA                   21 30 92 10 42
```

```
-- test if RX FIFO not empty
if (( R03 & 0x10 ) != 0 ) & (( R02 & 0x20 ) == 0) then
  serial_hw_write ( 0xAA )
  while ( R02 & 0x20 ) == 0 loop
    serial_hw_write ( SI4432_Read ( 0x7F ) )
    R02 = SI4432_Read ( 0x02 )
  end loop
  serial_hw_write ( R02 )
  serial_hw_write ( R03 )
  serial_hw_write ( R04 )
  serial_hw_write ( SI4432_Read ( 0x31 ))
  serial_hw_write ( SI4432_Read ( 0x26 ))
end if
```



```
52 21
FF FF D2 21 FF FF
AA D4 00 00 10 11 12 21 30 92 10 B1
AA 13 14 15 16 17 18 21 30 92 10 B1
AA 19 3F FF FF FF FF 21 30 82 10 43
AA EF FF FF FF FF FF 21 30 82 10 43
```

```
-- test if RX FIFO not empty
if (( R03 & 0x10 ) != 0 ) & (( R02 & 0x20 ) == 0) then
  ;serial_hw_write ( 0xAA )
  while (( R02 & 0x20 ) == 0) loop
    serial_hw_write ( SI4432_Read ( 0x7F ) )
    R02  = SI4432_Read ( 0x02 )
    RSSI = SI4432_Read ( 0x26 )
  end loop
  ;serial_hw_write ( R02 )
  ;serial_hw_write ( R03 )
  ;serial_hw_write ( R04 )
  ;serial_hw_write ( SI4432_Read ( 0x31 ))
  ;serial_hw_write ( SI4432_Read ( 0x26 ))
  if RSSI < 0x80 then
    SI4432_Write ( 0x07, 0x03 )
    SI4432_Write ( 0x08, 0x03 )
    SI4432_Write ( 0x08, 0x00 )
    delay_1ms ( 1 )            -- minimal 200 us
    SI4432_Write ( 0x07, 0x07 )
  end if
end if
```



```
EE EE 21 20 52 AD EE EE
FF FF 21 20 D2 AD FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 AD EE EE
FF FF 21 20 D2 AE FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF

EE EE 21 20 52 AB EE EE
FF FF 21 20 D2 AB FF FF
D4 00 00 10 11 12 13 14 15 16 17 18 19 3F FF FF FF FF
```